ENSAE ParisTech

Projet de Statistiques

Introduction à la régression pénalisée et à l'estimateur lasso

par Mehdi Abbana Bennani
et Julien Mattei

supérvisés par
Pr. Edwin Grappin

7 janvier 2017

1 Régression OLS

Question 1 :

Afin de calculer l'estimateur par moindre carrés, on minimise l'erreur quadratique :

$$\underset{\beta}{min} \left\| y - X\beta \right\|^2$$

Le gradient de l'erreur quadratique est nul en $\beta^\star$

$$\begin{aligned}\nabla_\beta \left\| y - X\beta \right\|^2 (\beta^\star) = 0 \iff & \nabla_\beta((y - X\beta)^T (y - X\beta))(\beta^\star) = 0 \\ \iff & -2X^T y + 2X^T X\beta^\star = 0 \\ \iff & \boxed{\beta^\star = (X^T X)^{-1} X^T y} \end{aligned} \tag{1}$$

Dans la suite on notera $\hat{\beta}$ l'estimateur des moindres carrés.

Question 2 :
Le code est disponible en annexe. Nous avons divisé le dataset en un Train Set et en Test Set. La proportion du Train Set est de 90.% Ceci va nous permettre de mesurer l'erreur de prédiction.
La métrique que nous avons choisie pour mesurer cette erreur est la RMSE (Root Mean Square Error), définie par :

$$RMSE(\mathbf{y_{pred}}, \hat{\mathbf{y}}) = \frac{1}{|PredictionSet|} \sum_{i=1}^{|PredictionSet|} (\hat{y}_i - y_i)^2 \tag{2}$$

Nous définissons aussi le R^2 la part de variance expliquée par le modèle comme suit :

$$R^2 = \frac{\sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{3}$$

Nous avons obtenu une RMSE de 4.53 sur le test set, et de 3.76 sur le train set. L'écart type estimé sur le Train Set est de l'ordre de 9. On obtient donc une RMSE de l'ordre de 40% de l'écart type. Il est normal que l'erreur sur le test set soit supérieure à celle mesurée sur le train set, car celui-ci a été utilisé pour estimer le paramètre. Nous avons aussi mesuré le R^2 soit la part de variance expliquée sur le train set, nous avons obtenu 96%, ce qui nous conforte sur la capacité prédictive du modèle. On remarque aussi qu'il n'y a aucun coefficient nul dans le paramètre $\hat{\beta}$ estimé.

Question 3 :
On ne peut pas utiliser la régression par moindres carrés si le nombre de variables est supérieur au nombre d'observations

Démonstration.
Soit X est une matrice de dimensions (n,p)

$X^T X$ est une matrice carrée de dimension p

On a $rg(X) \leq min(n, p)$

Or $rg(X^T X) = rg(X)$

Donc $rg(X^T X) \leq min(n, p)$

Par hypothèse, le nombre de variables est plus grand que le nombre d'observations, soit $p > n$

On en déduit que $rg(X^T X) < n$ car $min(n, p) = n$

Donc $rg(X^T X) < p$ car $n < p$

Ainsi $X^T X$ est une matrice de taille p dont le rang est strictement inférieur à sa dimension .

Donc $X^T X$ n'est pas inversible.

Conclusion : on ne peut pas utiliser la méthode des moindres carrées dans ce cas. □

2 Régression pénalisée : le lasso

Question 4 :

Sous l'hypothèse de p=1, on résout le problème d'optimisation :

$$\underset{\beta}{min} \frac{1}{2n} \|\mathbf{y} - \mathbf{X}\beta\|_2 + \lambda\|\beta\|_1$$

En développant la norme 2 on obtient :

$$\underset{\beta}{min} \frac{1}{2n} \sum_{i=0}^{n} (y_i - x_i\beta)^2 + \lambda|\beta|$$

A partir de l'équation (1), on retrouve la valeur de l'estimateur par moindres carrés $\hat{\beta} = \frac{y}{x}$

On injecte $\hat{\beta}$ dans le problème d'optimisation :

$$\underset{\beta}{min} \frac{1}{2n} \sum_{i=0}^{n} x_i^2\beta^2 - \frac{1}{n} \sum_{i=0}^{n} x_i^2\beta\hat{\beta} + \frac{1}{2n} \sum_{i=0}^{n} y_i^2 + \lambda|\beta|$$

On supprime les constantes :

$$\min_{\beta} \frac{1}{2n} \sum_{i=0}^{n} x_i^2 \beta^2 - \frac{1}{n} \sum_{i=0}^{n} x_i^2 \beta \hat{\beta} + \lambda |\beta|$$

On remarque que si $\hat{\beta} < 0$ Alors forcément $\beta^{lasso} < 0$ Car les termes en β^2 et en $|\beta|$ ne dépendent pas du signe de β, et vu que $\hat{\beta} < 0$ alors$-\frac{1}{n}\sum_{i=0}^{n} x_i^2 \hat{\beta} > 0$ Donc pour minimiser la quantité, il faut que $\beta^{lasso} < 0$

De même si $\hat{\beta} > 0$ Alors forcément $\beta^{lasso} > 0$

On va traiter les deux cas :

Si $\hat{\beta} > 0$, on résout :

$$\min_{\beta} \frac{1}{2n} \sum_{i=0}^{n} x_i^2 \beta^2 - \frac{1}{2n} \sum_{i=0}^{n} x_i^2 \beta \hat{\beta} + \lambda \beta$$

On obtient :

$$\beta^{lasso} = \hat{\beta} - \frac{\lambda}{\sum_{i=0}^{n} \frac{x_i^2}{2n}}$$

Or β^{lasso} forcément positif, donc

$$\beta^{lasso} = max(0, \hat{\beta} - \frac{\lambda}{\frac{1}{2n}\sum_{i=0}^{n} x_i^2})$$

Si $\hat{\beta} < 0$, à l'aide d'un raisonnement similaire, la seule différence étant $-\lambda$ au lieu de λ :

$$\beta^{lasso} = max(0, \hat{\beta} + \frac{\lambda}{\frac{1}{2n}\sum_{i=0}^{n} x_i^2})$$

On en déduit que :

$$\boxed{\beta^{lasso} = sg(\hat{\beta})(|\hat{\beta}| - t)_+} \text{ avec } t = \frac{\lambda}{\frac{1}{2n}\sum_{i=0}^{n} x_i^2} \tag{4}$$

Le seuillage doux permet de régler l'intervalle sur lequel on veut estimer β^{lasso}. Ainsi, comme le montre la figure 1 si notre estimateur $\hat{\beta}$ est compris dans l'intervalle $[-t, t]$ il est ramené à 0. On ne retient que les valeurs de $\hat{\beta}$ en dehors de

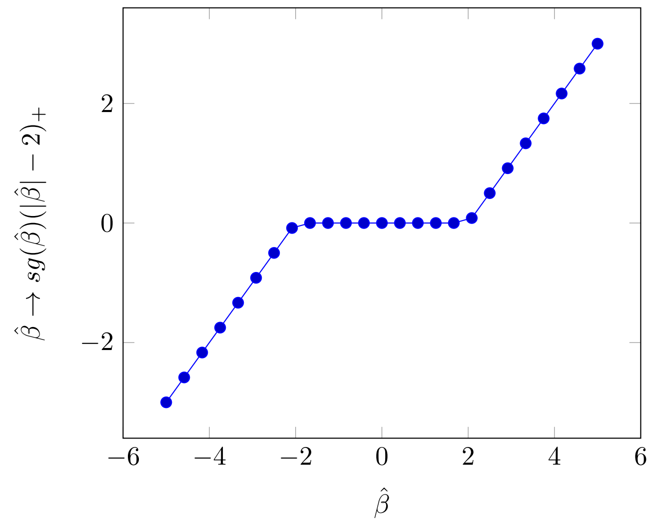


FIGURE 1 – β^{lasso} en fonction de $\hat{\beta}$ pout $t = 2$

cet intervalle (même principe qu'un filtre).
L'estimateur β^{lasso} est nul pour tout les $\hat{\beta}$ dans l'intervalle $[-t, t]$. Donc cela induit une estimation nulle plus souvent que l'estimateur des moindres carrés, celui-ci est nul seulement si $\hat{\beta} = 0$.
t est croissant en λ, donc en choisissant λ, nous pouvons modifier la taille du filtre, et donc régler le nombre de coefficients nuls dans notre estimateur, plus λ est grand plus le nombre de coefficients nuls est grand.

Question 5 :
Dans la régression pénalisée, en plus du coût de l'erreur de prédiction, on ajoute un coût lié aux valeurs de β pour limiter certaines de leurs valeurs possibles.
Dans le cas du lasso par exemple, toutes les coefficients de l'estimateur par moindres carrées inférieures à t en valeur absolue sont ramenées zéros, l'intervalle dans lequel β^{lasso} peut être non nul est $]-\infty, -t] \cup \{0\} \cup [t, \infty[$. Dans le cas de la régression Ridge, on pénalise les coefficients trop grands de β^{Ridge}.

L'estimateur Lasso est adapté à notre problème car nous allons pouvoir réduire la taille de l'espace des paramètres jusqu'à ce que le nombre de coefficients de beta non nuls soit égal à n (le nombre d'observations), en contrôlant la valeur de λ. Plus celui-ci est grand, plus le nombre de paramètres nuls sera grand, comme le montre la figure 1, car le filtre grandit.

Question 6 :
La pénalisation Ridge a tendance à pénaliser les coefficients dans β trop grandes, mais n'a pas tendance à réduire à zéro ces coefficients, contrairement à la pénalisation lasso. Elle est en général utilisée pour limiter le phénomène d'overfitting,

en réduisant les valeurs dans β en valeur absolue, ce qui diminue la variance des prédictions, réduisant ainsi l'overfitting.
Comme le lasso, la pénalisation l_0 vise aussi à réduire la taille de l'espace des variables, toutefois le problème d'optimisation associé est NP complet, contrairement à la pénalisation lasso pour laquelle nous avons pu obtenir une expression analytique en dimension 1 et qui peut être approchée en plus grande dimension.

Question 7 :
Nous avons utilisé le package Glmnet pour calculer l'estimation de β^{lasso}. La démarche est la même que celle de la question 2. La fonction glmnet cherche par défaut un λ optimal, nous avons imposé que la régression lasso soit effectuée pour $\lambda = 0.1$, le choix est arbitraire.
On obtient un R^2 de 0.84 sur le train set, qui laisse penser que le modèle a une bonne capacité prédictive, une RMSE sur le Test set de 3.77, et une RMSE de 2.61 sur le train set. L'écart type sur le Train set est de l'ordre de 10. Donc la qualité de la prédiction est raisonnable en comparaison avec l'écart type de l'échantilllon.

3 Le choix du paramètre λ

Question 8 :
Le paramètre*lambda* permet de réduire la taille de l'espace des variables en régleant le nombre de coefficients nuls dans β^{lasso}. t étant croissant en fonctino de λ, plus on augmente λ plus le filtre est large, plus le nombre de zéros augmente et plus la taille de l'espace des variables est petit.

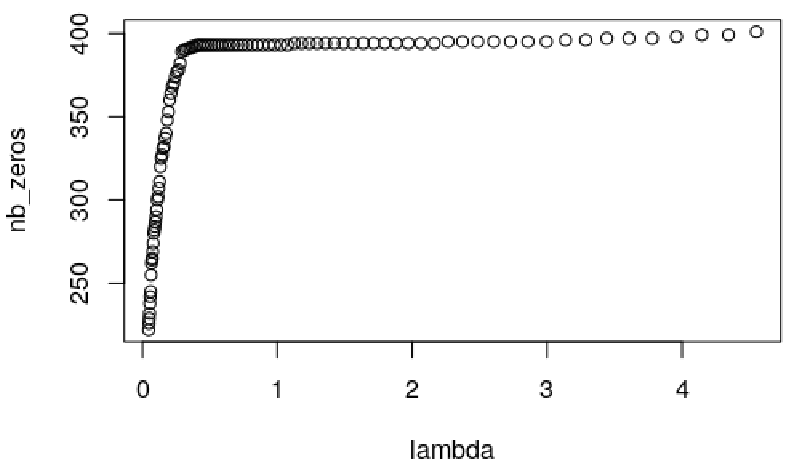
Question 9 :
Dans la méthode de cross validation, on divise le dataset en trois parties : Train Set, Validation Set, et Test Set. On entraine le modèle sur le train set puis on estime la qualité de la prédiction sur le validation set pour plusieurs valeurs de λ, on choisit le lambda le plus performant sur le validation set. Esuite on estime le modèle avec cette valeur de λ sur le train set comprenant le train set précédent et le validation set précédent.

Question 10 :
Pour évalueer l'erreur du modèle, nous avons divisé la base de données en Train Set et Test Set, les données du Train Set sont utilisées pour estimer les paramètres, les données du test set sont ensuite utilisés pour évaluer l'erreur d'estimation. La mesure que nous utiliserons est la Root Mean Square Error, RMSE, définie par :

$$RMSE(\mathbf{y_{test}}, \hat{\mathbf{y}}) = \frac{1}{|TestSet|} \sum_{i=1}^{|TestSet|} (\hat{y}_i - y_i)^2 \tag{5}$$

Comme le montre la figure 2, le nombre de zéros augmente avec lambda, on retrouve l'explication que l'on avait fourni dans la question 6. Lorsque le paramètre λ est trop grand, tous les coefficients de β^{lasso} sont nuls.

FIGURE 2 – Nombre de zéros dans β^{lasso} en fonction de λ, avec $\beta^{lasso} detaile 400$



Pour cette question, nous avons implémenté deux méthodes de cross- validation, l'une automatique à l'aide du package Gmlnet, et l'autre manuelle en définissant notre propre jeu de paramètres et en divisant la base manuellement.
L'implémentation qui se base sur le package est très simple, il suffit de choisir le nombre de λ que l'on veut tester et ensuite lancer l'estimation.
Pour l'implémentation manuelle, nous avons d'abord choisi un jeu de paramètres λ sur une échelle logarithmique. Ensuite nous avons divisé le dataset en Train, Cross-Validation et Test set. Enfin nous avons itéré les estimations de l'erreur sur le Cross-Validation set ensuite nous avons conservé le paramètre avec l'erreur la plus basse.
Comparaison des deux méthodes ?

4 Appendix

```r
# Pour les noms des variables, nous conservons les memes notations que l'enonce

# Global parameters
train_set_size = 0.90

# Computes the Root Mean Square Error of the predicted y
compute_rmse <- function(y, y_est){
  return(sum((y-y_est)^2)/length(y))
}

count_zeros <- function(x){
  return(sum(x == 0) )
}

# Question 2 : Multinlinear regression least squares estimator

# Estimate least sqaure OLS regression estimate of beta
estimate_beta <- function(X, y){
  M = t(X)%*%X
  M_inv = solve(M)
  beta = M_inv%*%t(X)%*%y
  return(beta)
}

# Returns the OLS regression prediction of y
predict_y <- function(X, beta){
  return(X%*%beta)
}

# Importating the data
ols_data_set = data.matrix(read.table("mysmalldata.txt", sep=','))

# Computing the train-test slicing coordinates
train_row_limit = train_set_size * nrow(ols_data_set)
test_row_start = train_row_limit + 1

# Splitting the data set into a train set and test set
X_train = ols_data_set[1:train_row_limit, 2:ncol(ols_data_set)]
y_train = ols_data_set[1:train_row_limit, 1]
X_test = ols_data_set[test_row_start:nrow(ols_data_set), 2:ncol(ols_data_set)]
y_test = ols_data_set[test_row_start:nrow(ols_data_set), 1]
```

```r

# Estimating the least square OLS regression estimate of beta using the train set
beta_hat = estimate_beta(X=X_train, y=y_train)

# Predicting and computing the error for the train set and the test set
y_est_train = predict_y(X=X_train, beta=beta_hat)
rmse_lse_train = compute_rmse(y=y_train, y_est=y_est_train)

y_est_test = predict_y(X=X_test, beta=beta_hat)
rmse_lse_test = compute_rmse(y=y_test, y_est=y_est_test)

nb_zeros_ols = count_zeros(beta_hat)

# Question 7 : Estimateur Lasso

# Package
install.packages("glmnet")
library("glmnet")

# Parameters
lambda = c(1)

# Importing the dataset
lasso_data_set = data.matrix(read.table("mydata.txt", sep=','))

# Computing the train-test slicing coordinates
train_row_limit = train_set_size * nrow(lasso_data_set)
test_row_start = train_row_limit + 1

# Splitting the data set into a train set and test set
X_train = lasso_data_set[1:train_row_limit, 2:ncol(lasso_data_set)]
y_train = lasso_data_set[1:train_row_limit, 1]
X_test = lasso_data_set[test_row_start:nrow(lasso_data_set), 2:ncol(lasso_data_set)]
y_test = lasso_data_set[test_row_start:nrow(lasso_data_set), 1]

# Estimating the lasso regression parameters using the glmnet package
# The parameter alpha is the weight between Lasso and Ridge regression, we set it to one for Lasso regression
fit = glmnet(x=X_train, y=y_train, alpha = 1, lambda = lambda)

# Predicting y using the lasso estimated parameters
```

```r
y_est_lasso = predict(fit, newx = X_test, type = "response",
    s=lambda)

# Computing the score for the lasso regression
rmse_lasso = compute_rmse(y_est_lasso, y_test)

# Conting the number of zeros
nb_non_zeros_ols = fit$df
nb_zeros_ols = ncol(lasso_data_set) - nb_non_zeros_ols

# Question 10 : Cross Validation

# Parameters
lambda_array = c(0.001, 0.01, 0.1, 1, 10)
train_size = 0.8
val_size = 0.1
test_size = 0.1
train_row_limit = train_size * nrow(ols_data_set)
validation_row_start = train_row_limit + 1
validation_row_end = (train_size + val_size) * nrow(ols_data_set)
test_row_start = validation_row_start + 1

# Train, validation, test split
X_train = lasso_data_set[1:train_row_limit,
    2:ncol(lasso_data_set)]
y_train = lasso_data_set[1:train_row_limit, 1]
X_val = lasso_data_set[validation_row_start:validation_row_end,
    2:ncol(lasso_data_set)]
y_val = lasso_data_set[validation_row_start:validation_row_end,
    1]
X_test = lasso_data_set[test_row_start:nrow(lasso_data_set),
    2:ncol(lasso_data_set)]
y_test = lasso_data_set[test_row_start:nrow(lasso_data_set), 1]

rmse_min = Inf
lambda_min = c(0)
# Estimating the lasso regression parameters using the glmnet
    package
for (lambda in lambda_array){
  lambda = c(lambda)
  # The parameter alpha is the weight between Lasso and Ridge
    regression, we set it to one for Lasso regression
  fit = glmnet(x=X_train, y=y_train, alpha = 1, lambda = lambda)
  # Predicting y using the lasso estimated parameters
  y_est_lasso = predict(fit, newx = X_val, type = "response",
    s=lambda)
```

```r
  # Computing the score for the lasso regression
  rmse_lasso = compute_rmse(y_est_lasso, y_val)
  if (rmse_lasso < rmse_min){
    rmse_min = rmse_lasso
    lambda_min = lambda
  }
}

# Merging the train and validation into a full train set
X_train = lasso_data_set[1:validation_row_end, 2:ncol(lasso_data_set)]
y_train = lasso_data_set[1:validation_row_end, 1]

# Predicting y using the lasso estimated parameters
y_est_lasso_cval = predict(fit, newx = X_test, type = "response", s=lambda)

# Computing the score for the lasso regression against the test set
rmse_lasso_cval = compute_rmse(y_est_lasso_cval, y_test)


# Alternative method for Cross Validation using the glmnet package
# Cross validation with 100 lambda trials
fit_auto = glmnet(X_train, y_train, alpha = 1, nlambda = 100)

# Predicting y using the lasso estimated parameters
y_est_lasso_auto = predict(fit_auto, newx = X_test, type = "response")

# Computing the score for the lasso regression
rmse_lasso_auto = compute_rmse(y_est_lasso_auto, y_test)

# The obtained value is slightly inferior to the value we obtained manually

# Plotting the number of zeros against lambdas
nb_zeros_array = sort(ncol(lasso_data_set) - fit_auto$df)
lambdas_array = sort(fit_auto$lambda)
```