Linear Methods for Regression

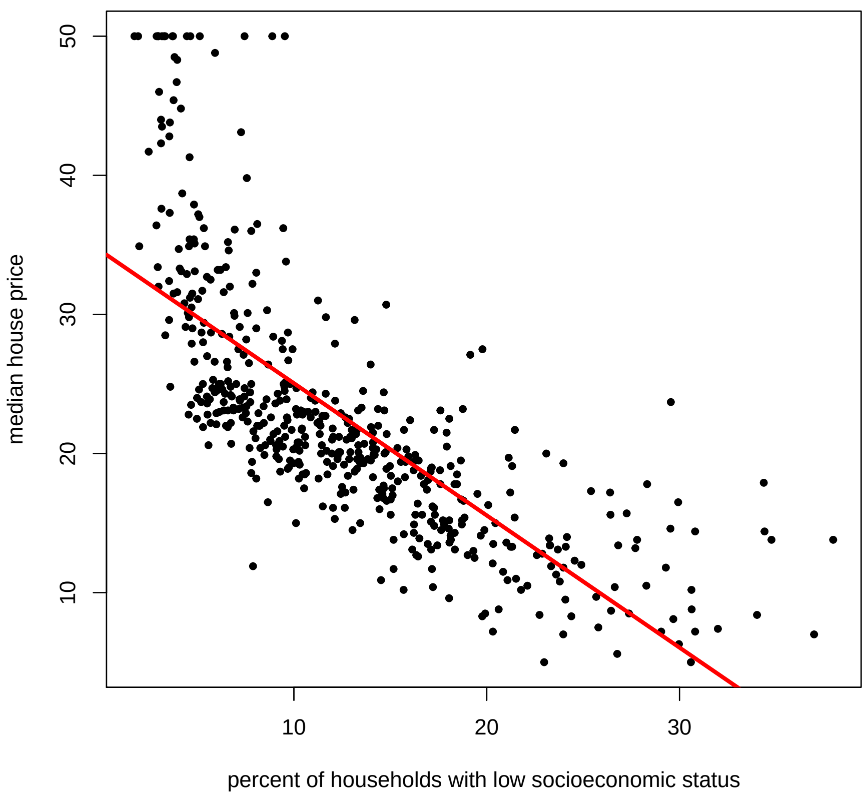
a.h.thiery@nus.edu.sg

Version: 0.1

1 Linear regression with one predictor

We consider the *Boston* data-set. As a first step, let us fit a simple linear model of the type $y = \beta_0 + \beta_1 x$ to try to explain $y =$ (median house price) from $x =$ (percent of households with low socioeconomic status).

```
set.seed(1) #for reproducibility
library(MASS)
#attach(Boston)
names(Boston)

##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"

#lstat  = percent of households with low socioeconomic status
#medv   = median house value

#fit a linear model
lm.fit = lm(medv ~ lstat, data=Boston)
print(lm.fit)

##
## Call:
## lm(formula = medv ~ lstat, data = Boston)
##
## Coefficients:
## (Intercept)        lstat
##       34.55        -0.95

#plot the fitted line
plot(Boston$lstat, Boston$medv, pch = 20,
     xlab="percent of households with low socioeconomic status",
     ylab="median house price")
abline(lm.fit, col="red", lwd=3)
```

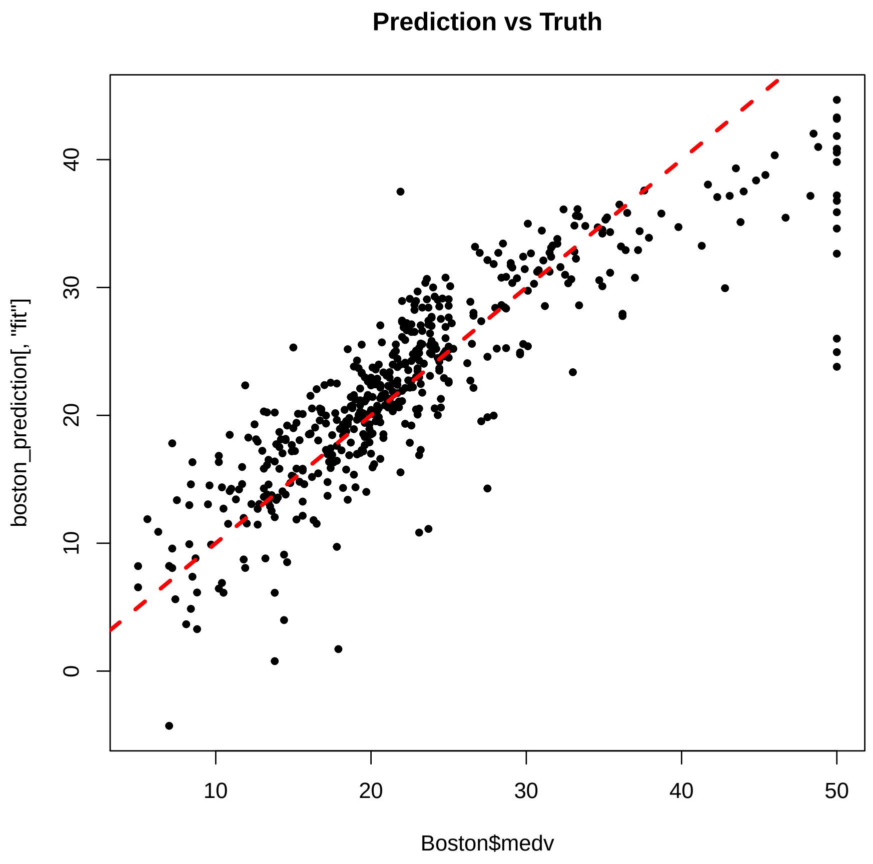


Let us fit a full linear regression. The covariates are:

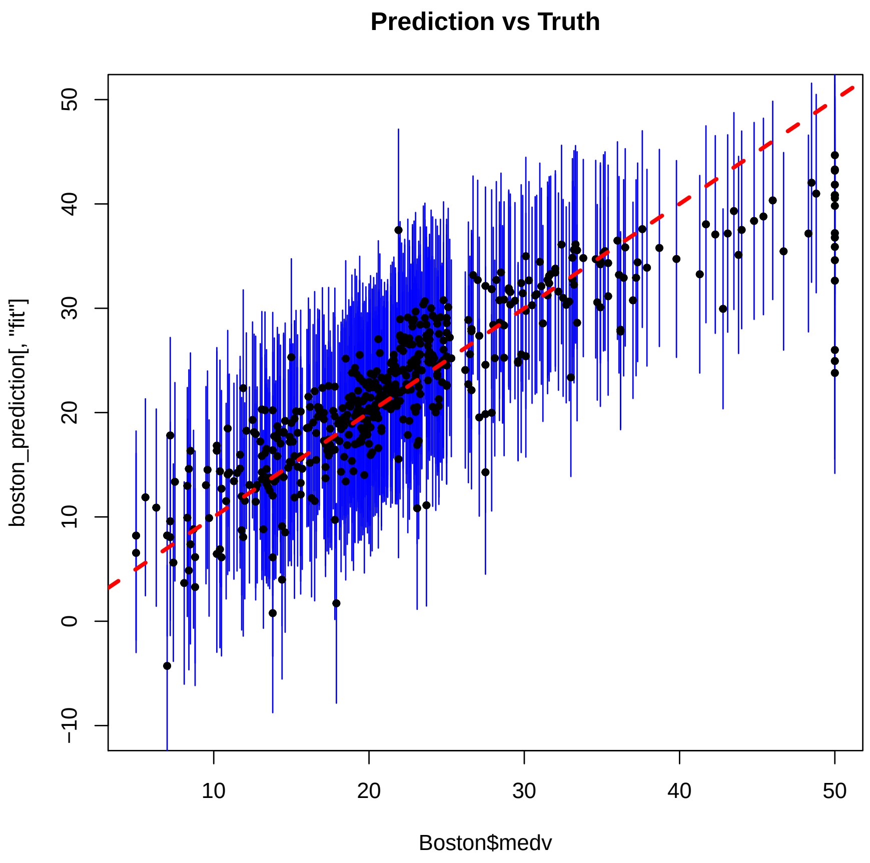
1. CRIM: per capita crime rate by town
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS: proportion of non-retail business acres per town
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX: nitric oxides concentration (parts per 10 million)
6. RM: average number of rooms per dwelling
7. AGE: proportion of owner-occupied units built prior to 1940
8. DIS: weighted distances to five Boston employment centres
9. RAD: index of accessibility to radial highways

10. TAX: full-value property-tax rate
11. PTRATIO: pupil-teacher ratio by town
12. BLACK: $1000 \times (Bk - 0.63)^2$ where Bk is the proportion of blacks by town
13. LSTAT: percentage lower status of the population
14. MEDV: Median value of owner-occupied homes

```r
lm.fit = lm(medv ~ ., data=Boston)
#let us do some prediction on the same dataset on has just used to fit the data
boston_prediction = predict( lm.fit, Boston, interval = "prediction")

#prediction versus truth
plot(Boston$medv, boston_prediction[,"fit"], pch=20, main="Prediction vs Truth")
abline(a=0, b=1, col="red", lwd=3, lty=2)
```



```r
#prediction versus truth
plot(Boston$medv, boston_prediction[,"fit"], pch=20,
     main="Prediction vs Truth", ylim=c(-10, 50))
segments(x0=Boston$medv, y0=boston_prediction[,"lwr"],
         x1=Boston$medv, y1=boston_prediction[,"upr"],
         col="blue", cex=0.1)
points(Boston$medv, boston_prediction[,"fit"], pch=20,
       main="Prediction vs Truth", ylim=c(-10, 50))
abline(a=0, b=1, col="red", lwd=3, lty=2)
```



1.1 Minimizing the Residual Sum of Square (RSS)

The coefficients $\beta = (\beta_0, \beta_1)$ are chosen such that the RSS

$$\mathbf{RSS}(\beta_0, \beta_1) = \sum_{i=1}^{n} \left(y_i - [\beta_0 + \beta_1 \, x_i]\right)^2$$

is minimum. Note that the function $(\beta_0, \beta_1) \mapsto \mathbf{RSS}(\beta_0, \beta_1)$ is a simple quadratic function. Setting the partial derivative of the function $\mathbf{RSS}(\beta_0, \beta_1)$ to zero yields that [**Exercise**]

$$\begin{cases} \beta_0 + \beta_1 \, \overline{x} & = \overline{y} \\ \beta_0 \, \overline{x} + \beta_1 \, \overline{xx} & = \overline{xy} \end{cases}$$

with $\overline{x} = n^{-1} \sum x_i$ and $\overline{y} = n^{-1} \sum y_i$ and $\overline{xx} = n^{-1} \sum x_i^2$ and $\overline{xy} = n^{-1} \sum x_i y_i$.

2 Linear regression with several predictors

Consider observations $y = \{y_i\}_{i=1}^n$ and p-dimensional predictors (or covariates) $x_i = (x_{i,1}, \ldots, x_{i,p})$. One is looking for $\beta = (\beta_0, \beta_1, \ldots, \beta_p) \in \mathbb{R}^{p+1}$ such that the **RSS** is minimum, with

$$\mathbf{RSS}(\beta) = \sum_{i=1}^{n} \left(y_i - [\beta_0 + \beta_1 \, x_{i,1} + \ldots + \beta_p \, x_{i,p}] \right)^2 .$$

In other words, the Ordinary Least Square (OLS) estimate $\widehat{\beta}$ is defined as

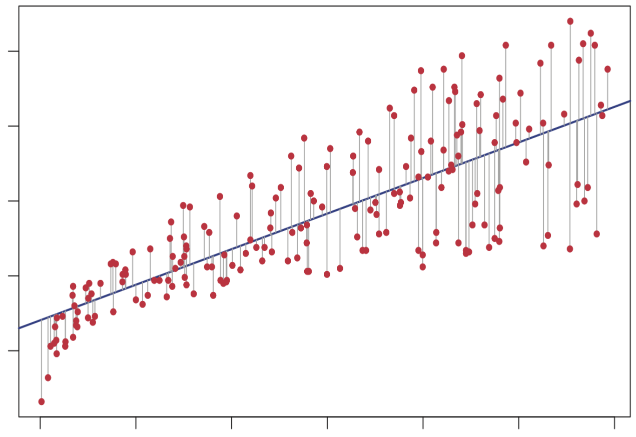


Figure 1: Residual Sum of Square (**RSS**)

$$\widehat{\beta} \;=\; \textbf{argmin} \; \left\{ \; \mathbf{RSS}(\beta) \; : \; \beta \in \mathbb{R}^{p+1} \; \right\}. \qquad (1)$$

One could try, and we will need to do so in similar situations, to numerically optimize the function $\beta \mapsto \mathbf{RSS}(\beta)$ using for example a gradient descent algorithm. Nevertheless, in this case the function $\beta \mapsto \mathbf{RSS}(\beta)$ is a simple quadratic function and one can write down an explicit expression for $\widehat{\beta}$. It is more convenient to introduce matrix notations and write

$$\mathbf{RSS}(\beta) = \|y - X\,\beta\|^2 = \langle y - X\,\beta, y - X\,\beta \rangle$$

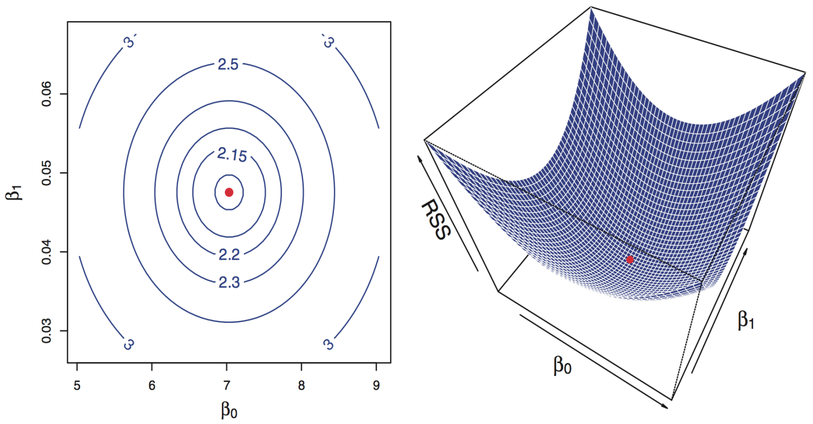


Figure 2: Ordinary Least Square (OLS) estimate $\widehat{\beta}$

with column vector $\beta \in \mathbb{R}^{p+1}$ and matrix $X \in \mathbb{R}^{n,p+1}$. The first and second derivatives (i.e. gradient and Hessian matrix) of **RSS** are given by [**Exercise**]

$$\partial_\beta \mathbf{RSS} = -2X^T\,(y - X\,\beta)$$
$$\partial^2_{\beta,\beta}\mathbf{RSS} = 2\,X^T\,X.$$

Since the matrix $X^T\,X$ is positive semi-definite [**Exercise**], this shows that the function $\beta \mapsto \mathbf{RSS}(\beta)$ is *convex*. If the matrix $X^T\,X$ is positive definite, which is the case if the matrix X has full rank, the function $\beta \mapsto \mathbf{RSS}(\beta)$ is strictly *convex* and it is then not hard to see that it has a *unique* minimum that can be found by setting the first derivative to zero. Let us assume that this is the case: this yields to [**Exercise**]

$$\widehat{\beta} = (X^T\,X)^{-1}\,X^T\,y \tag{2}$$

Note that the coefficient $\left[X^T\,X\right]_{i,j} = \langle x_i, x_j\rangle$; this remark will reveal useful in the chapter on *kernel methods*. Note that the **RSS** can be written as $\mathbf{RSS} = \sum_i (y_i - \widehat{y}_i)^2$ where the *fitted values* $\widehat{y}$ are defined by

$$\widehat{y} = \widehat{H}\,y \qquad \text{with} \qquad \widehat{H} \equiv X\,(X^T\,X)^{-1}\,X^T.$$

The matrix $\widehat{H}$ is called the *hat* matrix; that is a (orthogonal) projection in the sense that [**Exercise**]

$$\widehat{H}^2 = \widehat{H}.$$

Note that the solution $\widehat{\beta}$ is unique and the matrix $\widehat{H}$ is well defined only if the matrix $X^T\,X \in \mathbb{R}^{p+1,p+1}$ is invertible; it is never the case if $n \leq p$ i.e. when there is more covariates than observations [**Exercise**]. In this case, there exist infinitely many β's that minimize the **RSS**.

It should be noted that the OLS estimate is not necessary a sensible thing to consider if:

- the relationship between covariates and responses is not (approximately) linear
- the covariates are highly correlated; this often yield to highly unstable estimates.
- the variance of the noise is not (approximately) constant; if this is not the case, it may be worthwhile to consider a transformation of the covariates first.
- high correlation between the error terms
- presence of outliers (the square function is highly sensitive to outliers)

2.1 Property of $\widehat{\beta}$

Let us suppose, for simplicity, that the data are generated from a Gaussian model of the type $y_i = \beta_0 + \beta_1 x_{i,1} + \ldots + \beta_p x_{i,p} + \mathbf{N}\left(0, \sigma^2\right)$, for unknown parameter $\beta_\star = (\beta_0, \ldots, \beta_p)$. In this case $\widehat{\beta}$ is an unbiased estimate of $\beta_\star$ and [**Exercise**]

$$\widehat{\beta} \sim \mathbf{N}\left(\beta_\star, \sigma^2 \, (X^T X)^{-1}\right).$$

This shows that if the matrix $(X^T X)$ is almost singular then the variance of $\widehat{\beta}$ is very high and thus $\widehat{\beta}$ is an unreliable estimate of β. This is for example the case if:

- the covariates are highly correlated.
- the number of covariate is if the same order as the number of observations.

An estimate of σ^2 is simply given by the empirical covariance of $\{(y_i - \widehat{y}_i)^2\}_{i=1}^n.$,

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

2.2 Maximum likelihood interpretation of the least square estimate

Continuing with the assumption of Gaussian noise i.e. that the data are generated from a Gaussian model of the type $y_i = \beta_0 + \beta_1 x_{i,1} + \ldots + \beta_p x_{i,p} + \mathbf{N}\left(0, \sigma^2\right)$ for unknown parameter $\beta_\star = (\beta_0, \ldots, \beta_p)$, the log-likelihood is given by

$$\ell(\beta) = -\frac{1}{2\sigma^2} \left\| y - X\beta \right\|^2 + (\text{irrelevant additive constants})$$

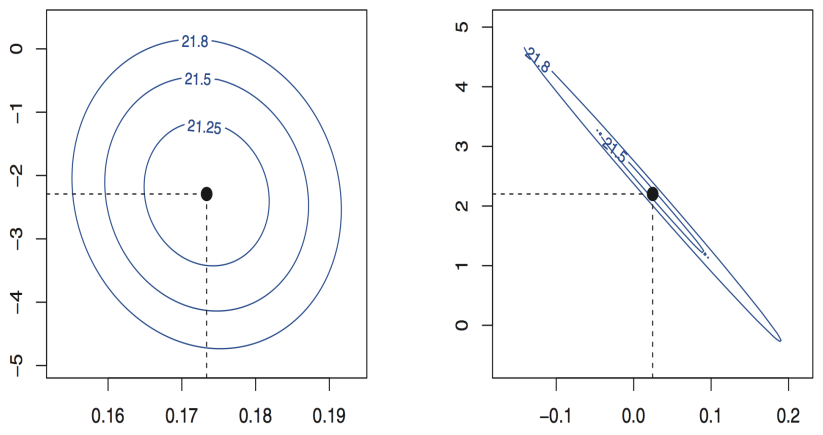


Figure 3: Left: no correlation. Right: high correlation

It follows that [**Exercise**] the least square estimate is also the maximum-likelihood estimate of β.

3 Linear model selection and regularization

There are several reasons why the least square estimate of β can be unsatisfying.

- If $p \geq n$ there is not unique solution for the minimization of the **RSS**. This *large p small n* situation is extremely important in practice.Indeed, in this case, the formula $(X^T X)^{-1} X^T y$ is not valid anymore since $(X^T X)$ is not invertible!
- in the case where $p \leq n-1$ but still large, even if the least square estimate $\widehat{\beta}$ is well-defined, as explained before, the variability of $\widehat{\beta}$ can be very important (because $(X^T X)$ is almost singular). This can yield to *overfitting* and/or *unreliable* conclusions.
- When p is large, one may want to find a solution with as many zero coefficient as possible (the one corresponding to irrelevant covariates) in order to obtain a model that is easier to interpret. Typically, all the coefficient of the least square estimate $\widehat{\beta}$ are non-zero.

3.1 Shrinkage methods

Subset selection procedures involve using a least squares approach and then select, according to some criterion, a subset of covariates: subsets with a lot of predictors are penalized. The approaches described in this section directly penalize the size of the coefficients or equivalently for the coefficient to live in a given subset. For a *regularization* parameter $\lambda > 0$ to be optimized (typically by cross-validation), the associated coefficients $\widehat{\beta}(\lambda)$ are given by the solution of the following penalized least square approach,

$$\widehat{\beta}(\lambda) \; = \; \textbf{argmin} \left\{ \, \textbf{RSS}(\beta) + \lambda \times \Omega(\beta) \; : \; \beta \in \mathbb{R}^{p+1} \right\}. \tag{3}$$

In other words, the coefficient $\widehat{\beta}(\lambda)$ is given by a similar optimization problem as the one describes in Equation (3), with the important difference that a penalization term $\lambda \times \Omega(\beta)$ is added. The coefficient $\lambda > 0$ is quantifies the amount of regularization. The two most common penalization terms are

$$\Omega_{\text{Ridge}}(\beta) \equiv \sum_{j=1}^{p} \beta_j^2 \equiv \|\beta\|_2^2 \qquad \text{and} \qquad \Omega_{\text{Lasso}}(\beta) \equiv \sum_{j=1}^{p} |\beta_j| \equiv \|\beta\|_1.$$

Note that, typically, the intercept coefficient β_0 is *not* penalized; this is because we do not want the procedures to be dependent on the location of the covariates i.e. the procedure should yield identical results would all the covariates to be shifted $x_i \mapsto x_i + c_i$. It is important to note that the results of these shrinkage procedure typically do depend on the scale of the covariates; in other words, if we multiply some coordinates by a fixed constant, the resulting estimation procedure will typically yield a different result. This would not be the case for example for the least square estimate. In practice, the responses and covariates are first centered/normalized i.e. $x_i \mapsto (x_i - \bar{x}_i)/\widehat{\sigma}(x)$ and $y \mapsto y - \bar{y}$, before applying a shrinkage procedure.

One can show (keyword: Lagrange duality) that $\widehat{\beta}(\lambda)$, solution of the optimization (3), is also solution of the following constrained optimization problem

$$\begin{cases} \text{Minimize} & \textbf{RSS}(\beta) \\ \text{Subject to} & \Omega(\beta) \leq T(\lambda) \end{cases} \tag{4}$$

for some threshold $T(\lambda)$ that depends on λ.

3.1.1 Ridge regression

For ridge regression, the coefficients are penalized by $\Omega_{\text{Ridge}}(\beta) \equiv \sum_{j=1}^{p} \beta_j^2$. In other words, the $\widehat{\beta}$ is solution of the following optimization problem,

$$\beta_{\text{Ridge}}(\lambda) \; = \; \left\{ \sum_{i=1}^{n} (y_i - [\beta_0 + \beta_1 \, x_{i,1} + \ldots + \beta_p \, x_{i,p}])^2 + \lambda \times \sum_{j=1}^{p} \beta_j^2 \; : \; \beta \in \mathbb{R}^{p+1} \right\}$$

In practice, the covariates are first normalized $x_i \mapsto (x_i - \bar{x})/\widehat{\sigma}(x)$ before applying a shrinkage procedure. Because of the normalization, one can forget about β_0 and suppose that $X \in \mathbb{R}^{n,p}$ and $\beta \in \mathbb{R}^p$: the vector $\beta_{\text{Ridge}}(\lambda)$ minimizes the function

$$\beta \mapsto \left\| y - X^T \beta \right\|^2 + \lambda \, \|\beta\|_2^2$$

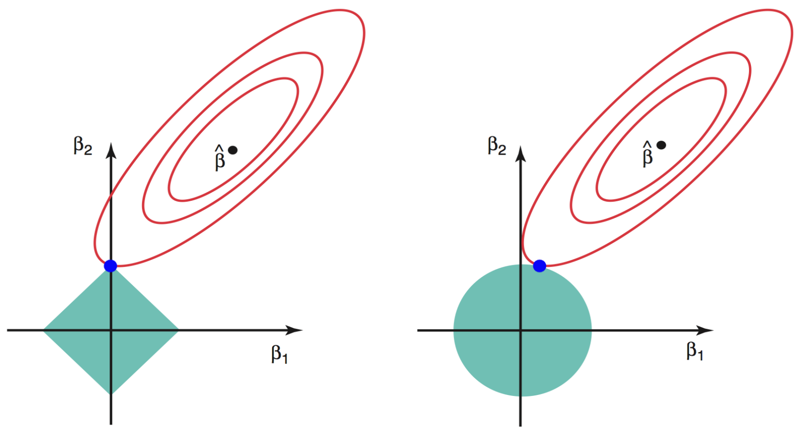


Figure 4: Left: Lasso. Right: Ridge regression

The gradient of this function [**Exercise**] reads $-2X^T\,(y - X\,\beta) + 2\lambda\,\beta$; setting this gradient to zero yields that

$$\widehat{\beta}(\lambda) = (X^T\,X + \lambda\,I)^{-1}\,X^T\,y. \tag{5}$$

Note [**Exercise**] that the matrix $(X^T\,X + \lambda\,I)$ is always invertible (in fact, positive definite) as soon as $\lambda > 0$. The case $\lambda = 0$ is the usual linear regression of Section 2. Note also that as $\lambda \to \infty$ we have $\widehat{\beta}(\lambda) \to 0$. Indeed, Equation (5) is a regularized version of (2). It is important to remark that ridge regression is well defined even for $p \geq n+1$, contrarily to the standard linear regression. The value of the regularization parameter $\lambda > 0$ is usually found by cross-validation.

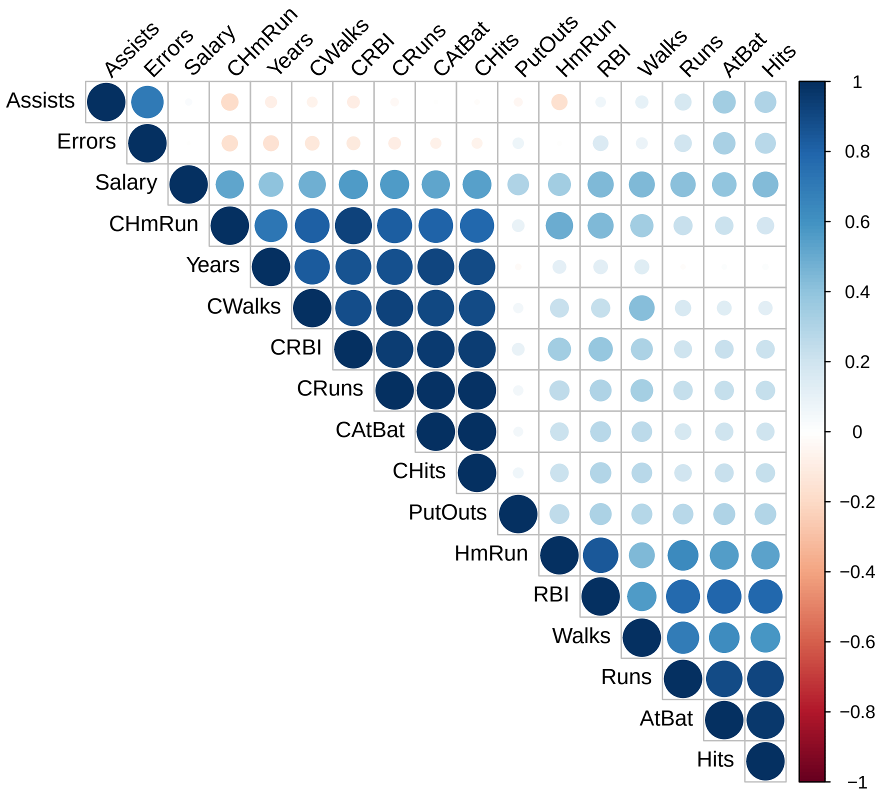
3.2 Ridge regression: Hitters dataset

We consider in this section a dataset of cricket players. We will be traying to use a linear model to predict the salaries of these players. Fir, let us load the dataset and quickly visualize it.

```
library(ISLR)  #for "Hitters" dataset
attach(Hitters)
#for simplicity, delete rows with N.A. entries
Hitters = na.omit(Hitters)
names(Hitters)

##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"
##  [6] "Walks"     "Years"     "CAtBat"    "CHits"     "CHmRun"
## [11] "CRuns"     "CRBI"      "CWalks"    "League"    "Division"
## [16] "PutOuts"   "Assists"   "Errors"    "Salary"    "NewLeague"
```

```
#For simplicity, drop non numerical values
Hitters = Hitters[,!(names(Hitters)
                        %in% c("League", "Division", "NewLeague") ) ]
```

```
#Let us look at the correlation between the variables
library(corrplot)

## Warning:  package 'corrplot' was built under R version 3.2.5

corrplot(cor(Hitters), type="upper",
         order="hclust", tl.col="black", tl.srt=45)
```



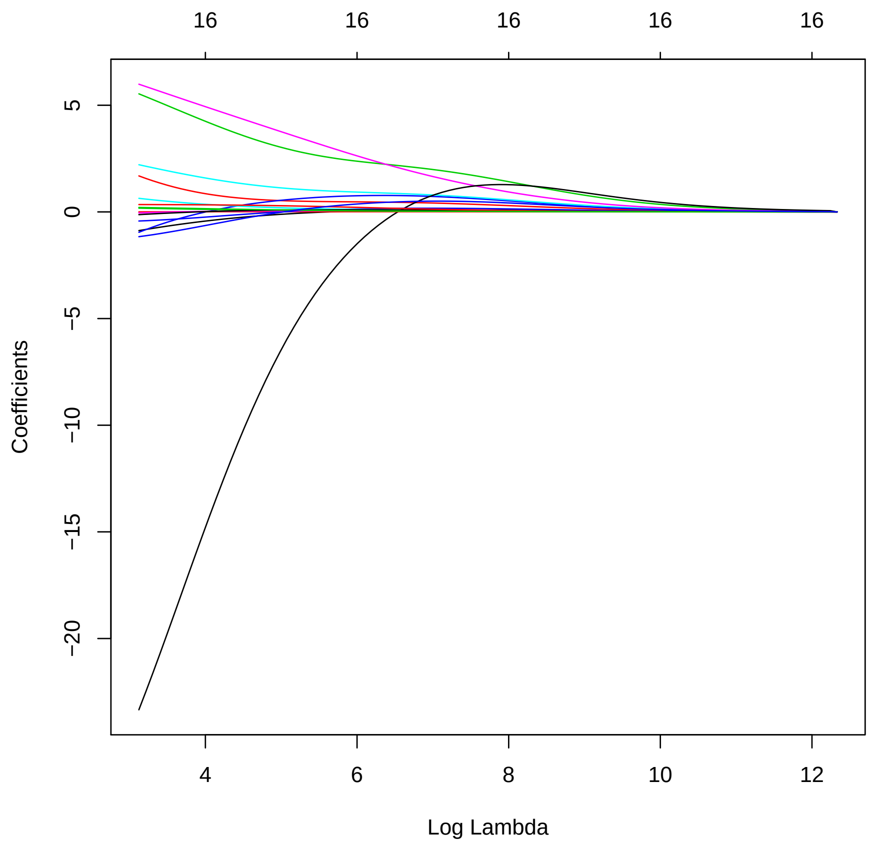
We can see that a lot of the variables are highly correlated. Variable selection, in this case, can be useful (highly correlated variables yield to a matrix $X^T X$ that is almost singular – the standard least square estimate $\widehat{\beta}$ is unstable). Dimension reduction techniques, which will be discussed in subsequent chapters, may also reveal useful in such situations.

First, let us extract the X matrix and split it as a training / test set.

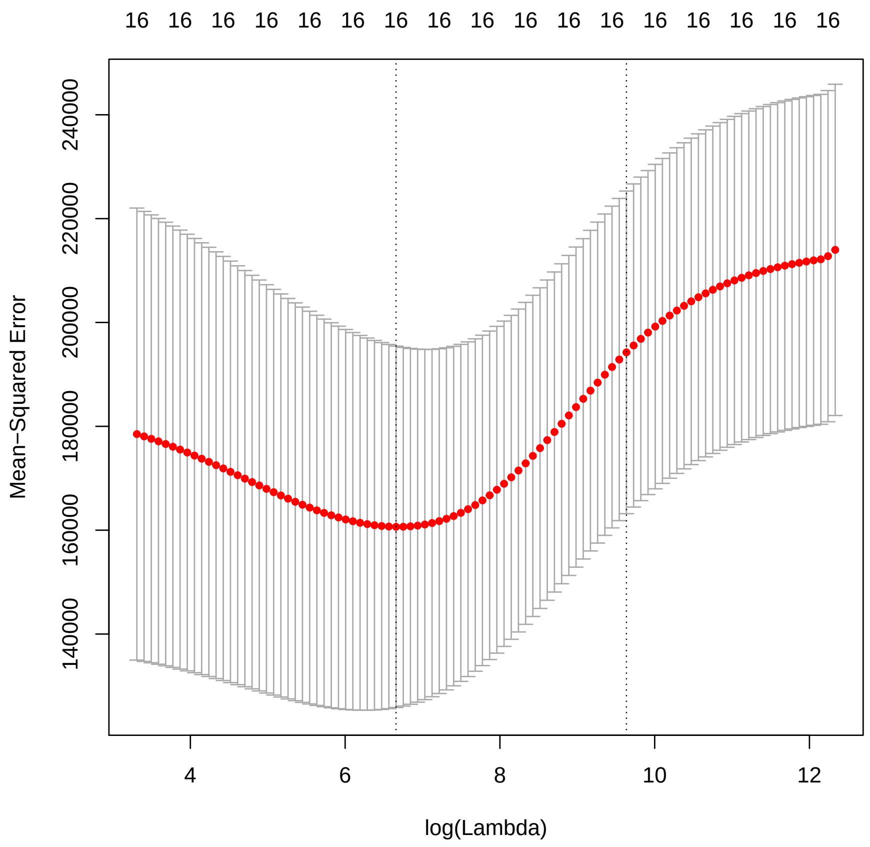
```
library(glmnet) #for ridge regression and Lasso, etc..

## Warning:  package 'glmnet' was built under R version 3.2.4
## Loading required package:  Matrix
## Warning:  package 'Matrix' was built under R version 3.2.5
## Loading required package:  foreach
## Loaded glmnet 2.0-5

#by default GLMET standardize the variable
#to turn that off, use standardize=FALSE

#create the covariate matrix (but delete the intercept)
x = model.matrix(Salary ~ ., Hitters)[,-1]
y = Hitters$Salary

#split training / test
n_data = length(y)
train_index = sample(1:n_data, 0.5*n_data, replace=FALSE)
x_train = x[train_index,]
y_train = y[train_index]
x_test = x[-train_index,]
y_test = y[-train_index]
```

Let us run ridge regression and find λ by cross-validation.

```
#create a grid of potential Lambdas
lambda.grid = 10^seq(5,-6,length=1000)
#for illustration, let us plot the coefficients
#as a function of lambda.
#alpha = 0 corresponds to ridge regression
ridge.fit = glmnet(x_train, y_train, alpha=0)
plot(ridge.fit, xvar = "lambda")
```



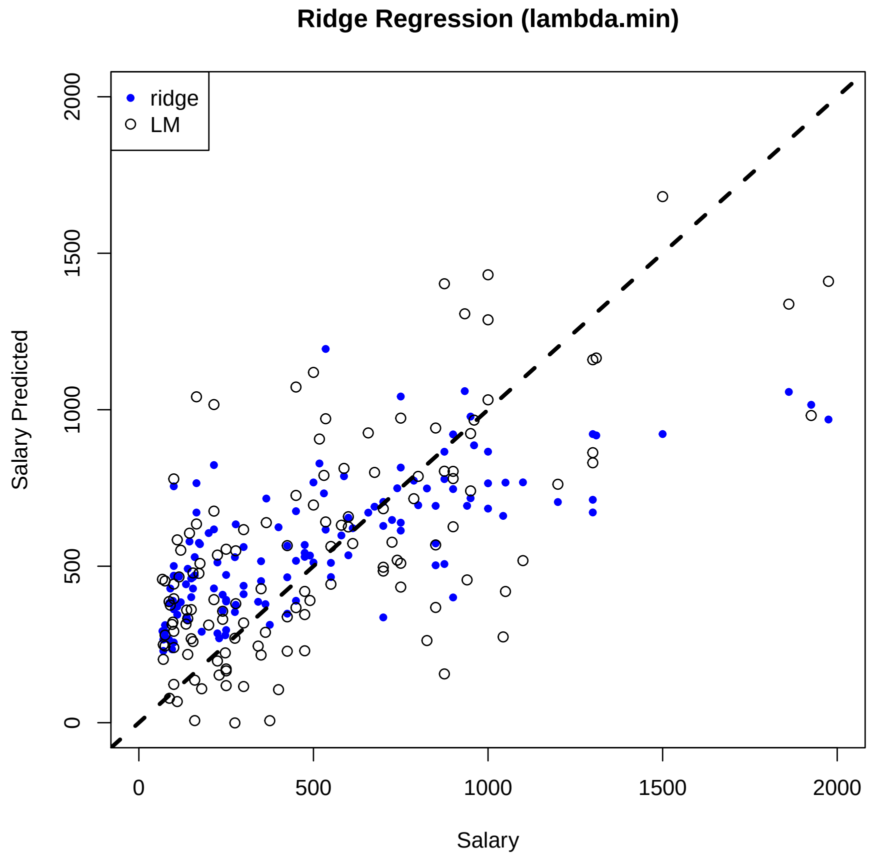
```
#Now, use a 10-fold cross validation
#to find a reasonable value of lambda
#we use MSE as measure of performance
ridge.cv = cv.glmnet(x_train, y_train, alpha=0,
      type.measure = "mse", nfolds = 10)
#plot the coefficients as a function of lambda
plot(ridge.cv)
```



```
#let us examine the coefficients
coef(ridge.cv, s = "lambda.min")

## 17 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept) 96.260221901
## AtBat        0.085185696
## Hits         0.442742211
## HmRun        2.128231407
## Runs         0.759842796
## RBI          0.847759753
## Walks        1.963936139
## Years        0.241632704
## CAtBat       0.008477867
## CHits        0.038323734
## CHmRun       0.169416123
## CRuns         0.090671974
## CRBI          0.056800450
## CWalks        0.088869930
## PutOuts       0.155604799
## Assists       0.038633655
## Errors        0.484947084
```

Note that none of the coefficients are exactly zero. We will see in Section 3.2.1 that this is very different when we use a LASSO shrinkage instead of a ridge shrinkage.

```
#let us compare prediction v.s. truth
# use "exact=TRUE" just to make sure
# that the coefficients at computed correctly
ridge.predict = predict(ridge.cv, newx = x_test,
      s = "lambda.min", exact=TRUE)
plot(y_test ,  ridge.predict,
      xlim=c(0,2000), ylim=c(0,2000), pch=20, col="blue",
      xlab="Salary", ylab="Salary Predicted",
      main="Ridge Regression (lambda.min)")
#superpose basic LM prediction for comparison
lm.fit = lm(y~., data=data.frame(y = y_train, x=x_train))
lm.predict = predict.lm(lm.fit, data.frame(x=x_test))
points(y_test , lm.predict,
      xlim=c(0,2000), ylim=c(0,2000), pch=21, col="black")
abline(a=0, b=1, lwd=3, lty=2)
legend("topleft", legend = c("ridge", "LM"),
       pch=c(20,21), col=c("blue","black"))
```



3.2.1 LASSO (Least Absolute Shrinkage and Selection Operator)

For LASSO regression, the coefficients are penalized by $\Omega_{\text{LASSO}}(\beta) \equiv \sum_{j=1}^{p} |\beta_j|$. In other words, the $\widehat{\beta}$ is solution of the following optimization problem,

$$\beta_{\text{Lasso}}(\lambda) \;=\; \left\{ \sum_{i=1}^{n} (y_i - [\beta_0 + \beta_1 \, x_{i,1} + \ldots + \beta_p \, x_{i,p}])^2 + \lambda \times \sum_{j=1}^{p} |\beta_j| \;:\; \beta \in \mathbb{R}^{p+1} \right\}$$

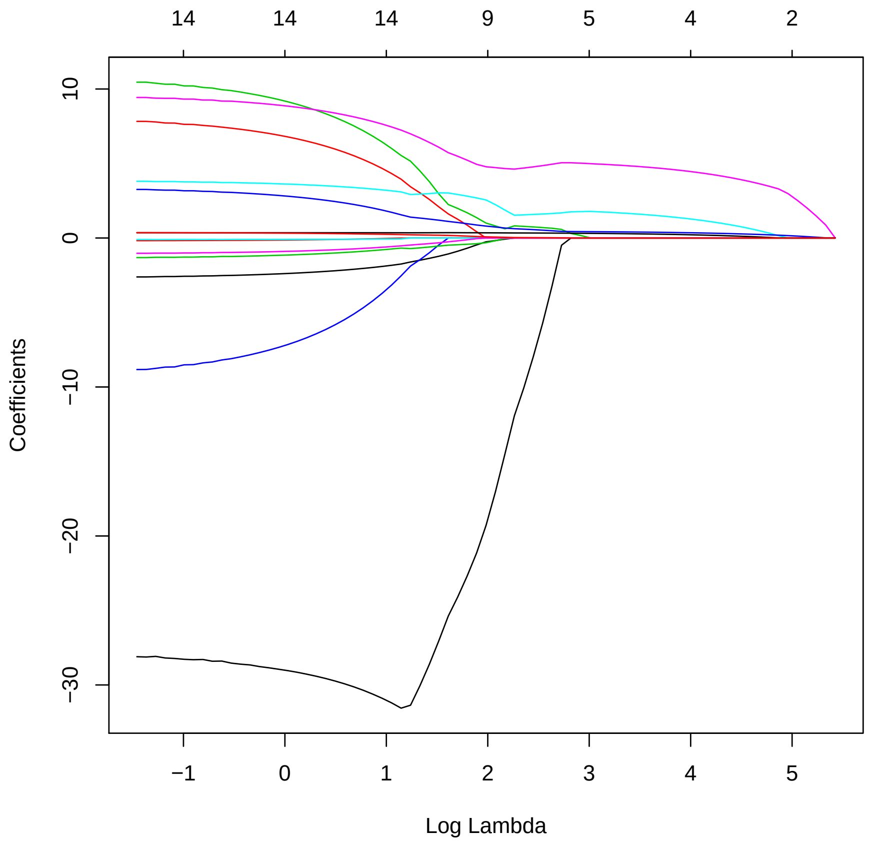
In practice, the covariates are first normalized $x_i \mapsto (x_i - \bar{x})/\widehat{\sigma}(x)$ before applying a shrinkage procedure. Because of the normalization, one can forget about β_0 and suppose that $X \in \mathbb{R}^{n,p}$ and $\beta \in \mathbb{R}^p$: the vector $\beta_{\text{Ridge}}(\lambda)$ minimizes the function

$$\beta \mapsto \left\| y - X^T \beta \right\|^2 + \lambda \, \|\beta\|_1 \tag{6}$$

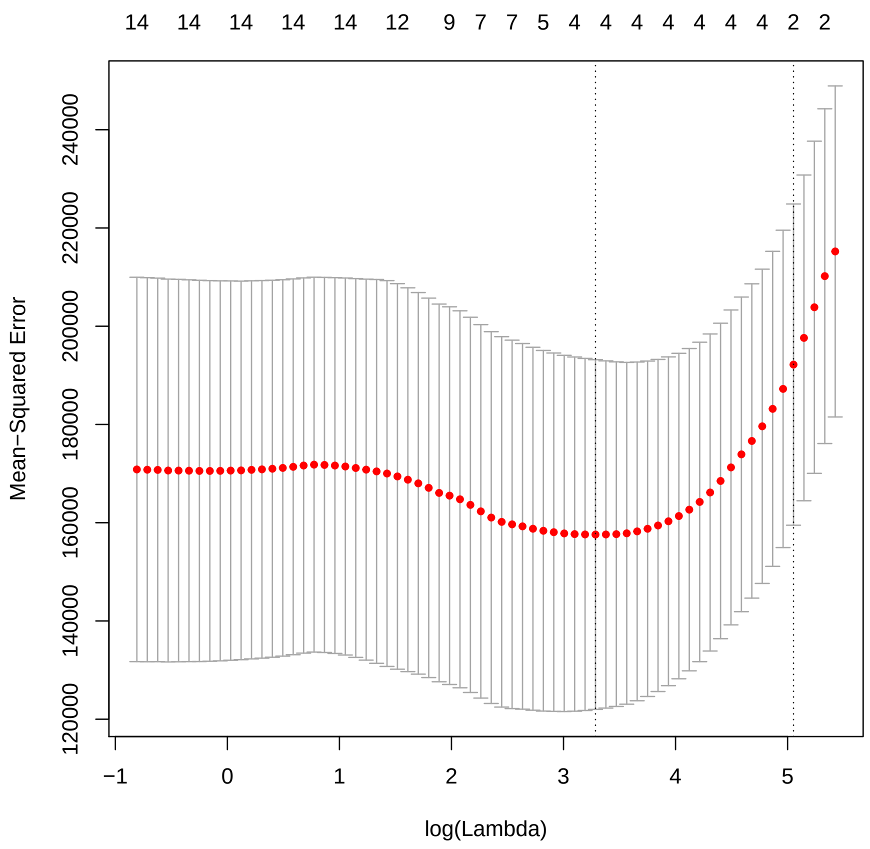
Contrarily to ridge regression, it is not possible in general to give an explicit expression for $\beta_{\text{Lasso}}(\lambda)$; indeed, the function (6) is not differentiable everywhere;

if β has some coefficient equal to zero, the function (6) is in general not differentiable at β. On the other hand, we will see that the minimum of (6) typically does have some coefficient equal to zero. To compute $\beta_{\text{Lasso}}(\lambda)$, one thus have to use some numerical optimization procedure. Note that a naive gradient descent algorithm would not work because of the lack of differentiability. The description of the minimization of (6) is beyond the scope of this lecture, but it is important to know that there exist very efficient method to do so.

```
#for, for illustration, let us plot the coefficients as a function of lambda
#alpha = 1 corresponds to LASSO
lasso.fit = glmnet(x_train, y_train, alpha=1)
plot(lasso.fit, xvar = "lambda")
```



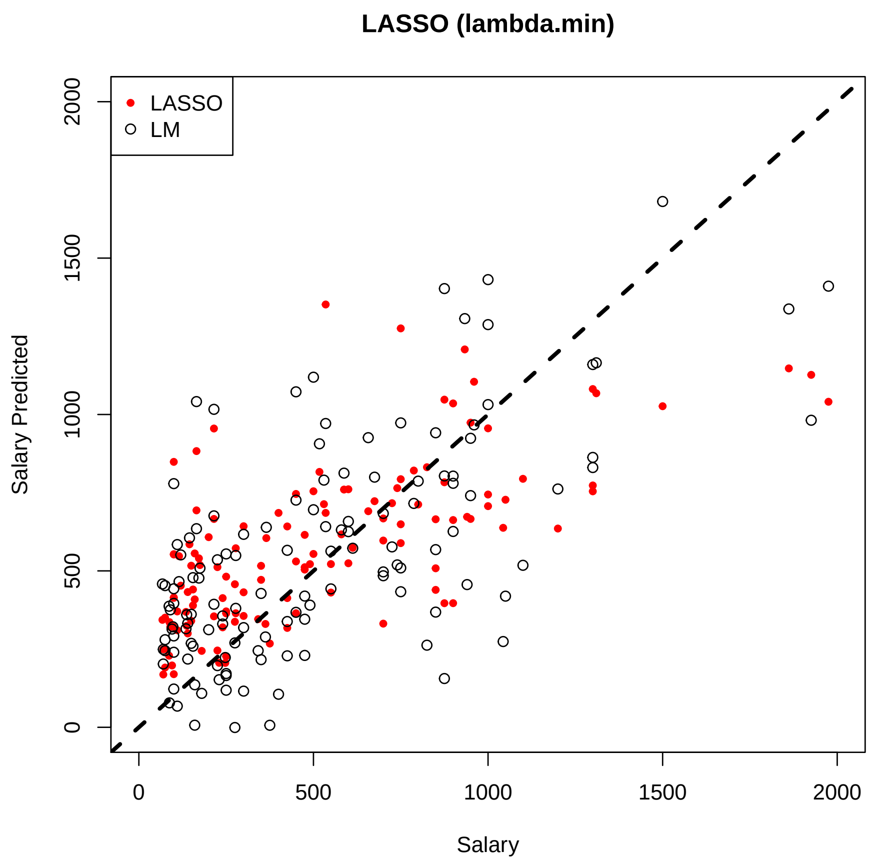
```
#Now, use a 10-fold cross validation to find a reasonable value of lambda
#we use MSE as measure of performance
lasso.cv = cv.glmnet(x_train, y_train, alpha=1,
      type.measure = "mse", nfolds = 10)
#plot the coefficients as a function of lambda
plot(lasso.cv)
```



```
#let us examine the coefficients
coef(lasso.cv, s = "lambda.min")

## 17 x 1 sparse Matrix of class "dgCMatrix"
##                      1
## (Intercept) 42.6216249
## AtBat        .
## Hits         .
## HmRun        .
## Runs         .
## RBI         1.6904781
## Walks       4.8900315
## Years          .
## CAtBat         .
## CHits          .
## CHmRun         .
## CRuns          0.4111395
## CRBI           .
## CWalks         .
## PutOuts        0.2936944
## Assists        .
## Errors         .
```

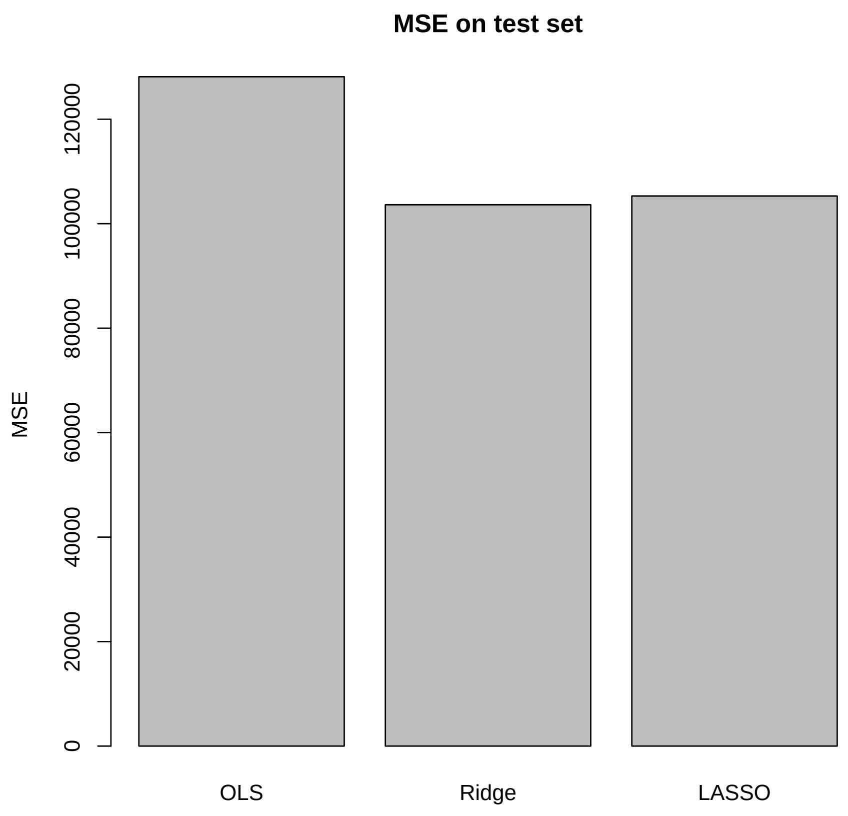
Note that many coefficients are exactly equal to zero: these variables are not relevant. To understand why the LASSO estimate yields sparse estimate (i.e. estimate with a lot of coefficient exactly set to zero), it suffices to look at the geometry of the set $\{\beta \,:\, \Omega_{\text{Lasso}}(\beta) \leq T(\lambda)\}$.

```
#let us compare prediction v.s. truth
#use "exact=TRUE" just to make sure
#that the coefficient at computed correctly
lasso.predict = predict(lasso.cv, newx = x_test,
      s = "lambda.min", exact=TRUE)
plot(y_test , lasso.predict, pch=20, col="red",
      xlim=c(0,2000), ylim=c(0,2000),
      xlab="Salary", ylab="Salary Predicted",
     main="LASSO (lambda.min)")
#superpose basic LM prediction for comparison
points(y_test , lm.predict,
      xlim=c(0,2000), ylim=c(0,2000), pch=21, col="black")
abline(a=0, b=1, lwd=3, lty=2)
legend("topleft", legend = c("LASSO", "LM"),
       pch=c(20,21), col=c("red","black"))
```



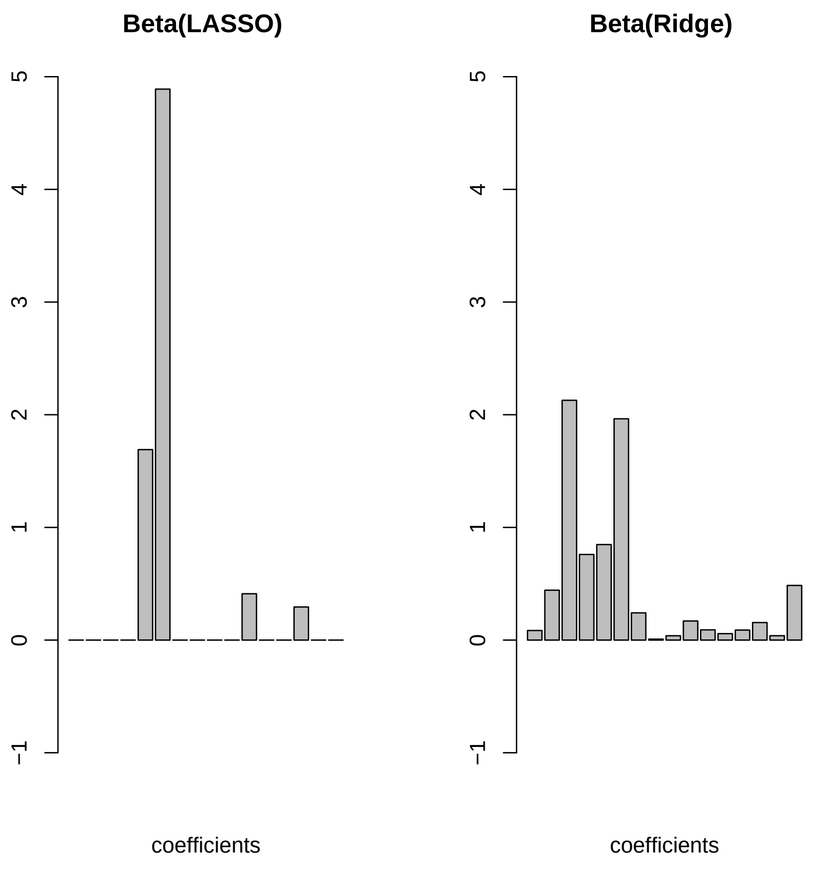
And let us compare the MSE on the test set.

```
lm.MSE = mean( (y_test - lm.predict)**2 )
ridge.MSE = mean( (y_test - ridge.predict)**2 )
lasso.MSE = mean( (y_test - lasso.predict)**2 )

#create barplot
barplot(c(lm.MSE,ridge.MSE,lasso.MSE),
        main="MSE on test set", ylab="MSE",
        names.arg=c("OLS", "Ridge", "LASSO"))
```



Let us plot a bar-plot of the coefficients the β_{Lasso} and β_{Ridge}.

```r
#the intercept coefficient is not displayed
par(mfrow=c(1,2))
barplot(as.vector(coef(lasso.cv, s = "lambda.min"))[2:17],
        main = "Beta(LASSO) ", xlab="coefficients", ylim=c(-1,5))

barplot(as.vector(coef(ridge.cv, s = "lambda.min"))[2:17],
        main = "Beta(Ridge) ", xlab="coefficients", ylim=c(-1,5))
```



3.3 Comparison of the methods

Let us consider a case where the number of potential predictors $p = 200$ is of the same order of magnitude as the number of $n = 400$ but where only a small numbers are actually informative.

```
#create the covariate matrix (but delete the intercept)
n = 400
p = 200
p_effective = 5
#create the matrix of predictos
x = matrix(rnorm(n*p, mean=0, sd = 1), nrow=n, ncol=p)
#create a beta coeeficient where only the first "p_effective"
#coefficients are non-zero
beta = c(rnorm(p_effective, mean=0, sd = 1), rep(0, p-p_effective))
#create noisy observations
```

```
sd_noise = 1
y = x %*% matrix(beta, ncol=1) + rnorm(n, mean=0, sd=sd_noise)
```

Let us split the data into a train set and a test set. We will keep 70% of the data as training set.

```
#split training / test
n_data = length(y); percent_train = 0.7
train_index = sample(1:n_data, percent_train*n_data, replace=FALSE)
x_train = x[train_index,]; y_train = y[train_index]
x_test = x[-train_index,]; y_test = y[-train_index]
```

Let us compare ridge regression, lasso, and standard linear regression. Let us start by running a ridge regression procedure.

```
lambda.grid = 10^seq(5,-5,length=100)
#run ridge regression
ridge.cv = cv.glmnet(x_train, y_train, alpha=0,
    type.measure = "mse", nfolds = 10)
ridge.predict = predict(ridge.cv, newx = x_test,
      s = "lambda.min", exact=TRUE)
ridge.test.MSE = mean( (ridge.predict - y_test) ** 2 )
cat("RIDGE: MSE on test set = ", ridge.test.MSE,"\n")

## RIDGE: MSE on test set =  1.658441
```

Let us run a LASSO procedure.

```
#run LASSO
lasso.cv = cv.glmnet(x_train, y_train, alpha=1,
    type.measure = "mse", nfolds = 10)
lasso.predict = predict(lasso.cv, newx = x_test,
      s = "lambda.min", exact=TRUE)
lasso.test.MSE = mean( (lasso.predict - y_test) ** 2 )
cat("LASSO: MSE on test set = ", lasso.test.MSE,"\n")

## LASSO: MSE on test set =  0.8031973
```

Let us see if the LASSO procedure has identified the true nonzero coefficients.

```
lasso.coef = coef(lasso.cv, s = "lambda.min")
#display the first 10 coefficients
cat(lasso.coef[1:10], "\n")

## -0.04819985 0.2025955 0.1230249 -0.8101585 -0.5924761 -0.9317421 0.03019895 0 0 0
```

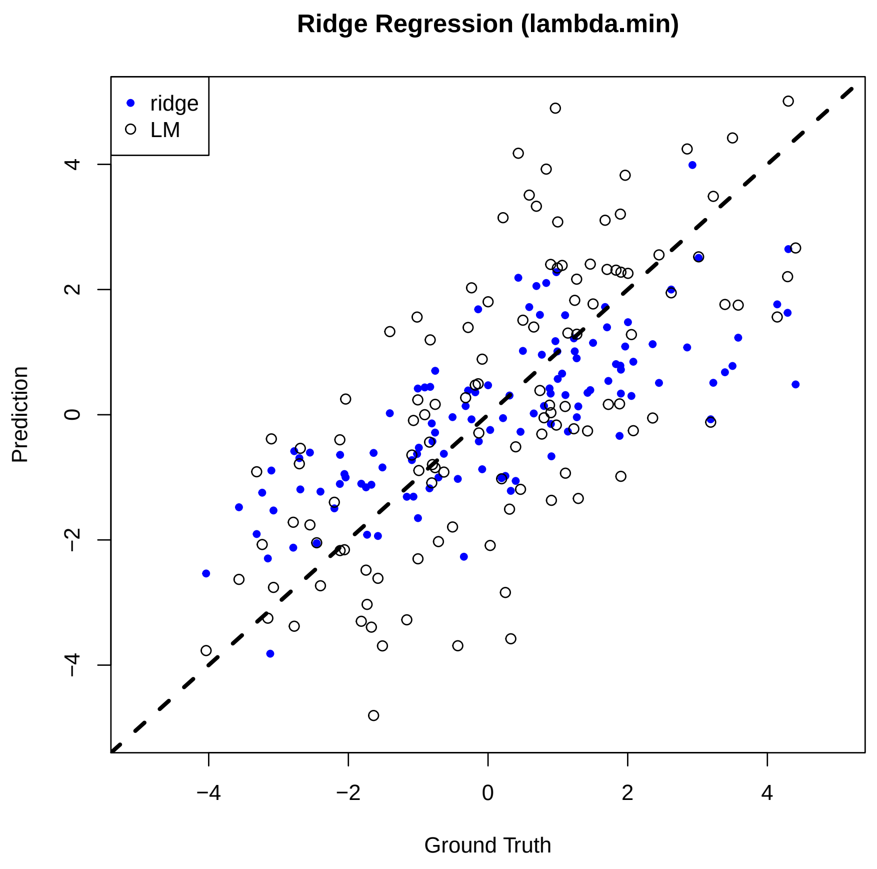
```
#count the number of nonzero coefficients
cat("LASSO: number of non-zero coefficients = ",
    sum(lasso.coef != 0), "\n")

## LASSO: number of non-zero coefficients =  28
```

Finally, let us run a standard linear regression.

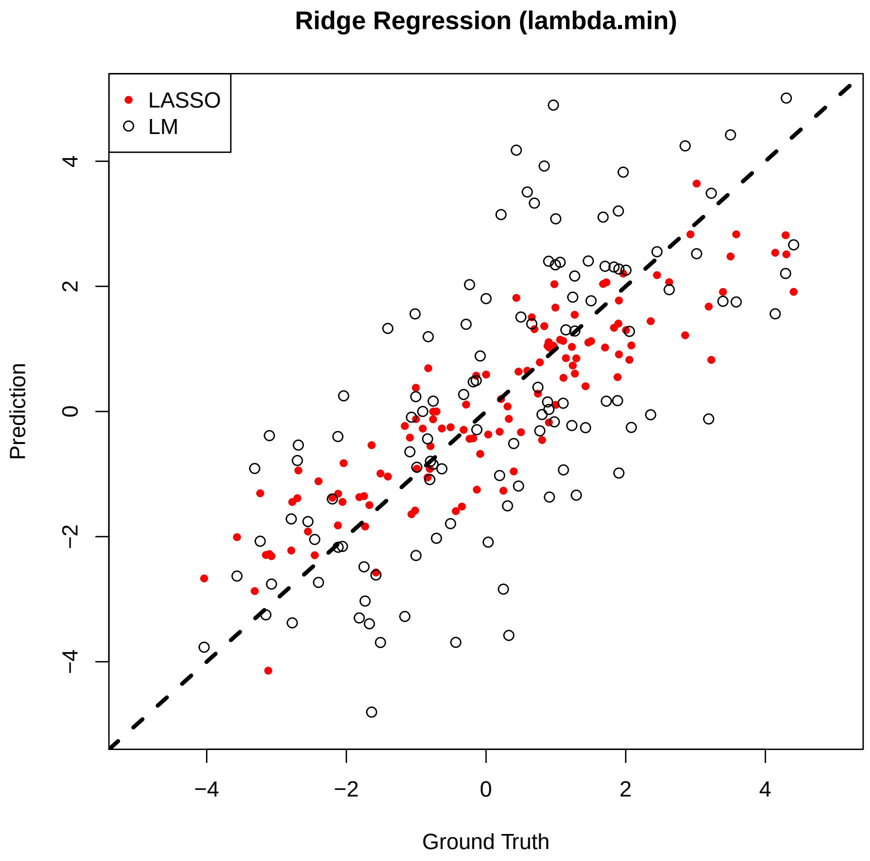
```
#run basic Linear regression
lm.fit = lm(y~., data=data.frame(y = y_train, x=x_train))
lm.predict = predict.lm(lm.fit, data.frame(x=x_test))
lm.test.MSE = mean( (lm.predict - y_test) ** 2 )
cat("STANDARD LM: MSE on test set = ", lm.test.MSE,"\n")

## STANDARD LM: MSE on test set =  3.425643
```

Let us plot the results on the test set.

```
#plot Ridge vs LM
plot(y_test , ridge.predict,
      xlim=c(-5,5), ylim=c(-5,5), pch=20, col="blue",
      xlab="Ground Truth", ylab="Prediction",
      main="Ridge Regression (lambda.min)")
#superpose basic LM prediction for comparison
points(y_test , lm.predict,
      xlim=c(-5,5), ylim=c(-5,5), pch=21, col="black")
abline(a=0, b=1, lwd=3, lty=2)
legend("topleft", legend = c("ridge", "LM"),
       pch=c(20,21), col=c("blue","black"))
```

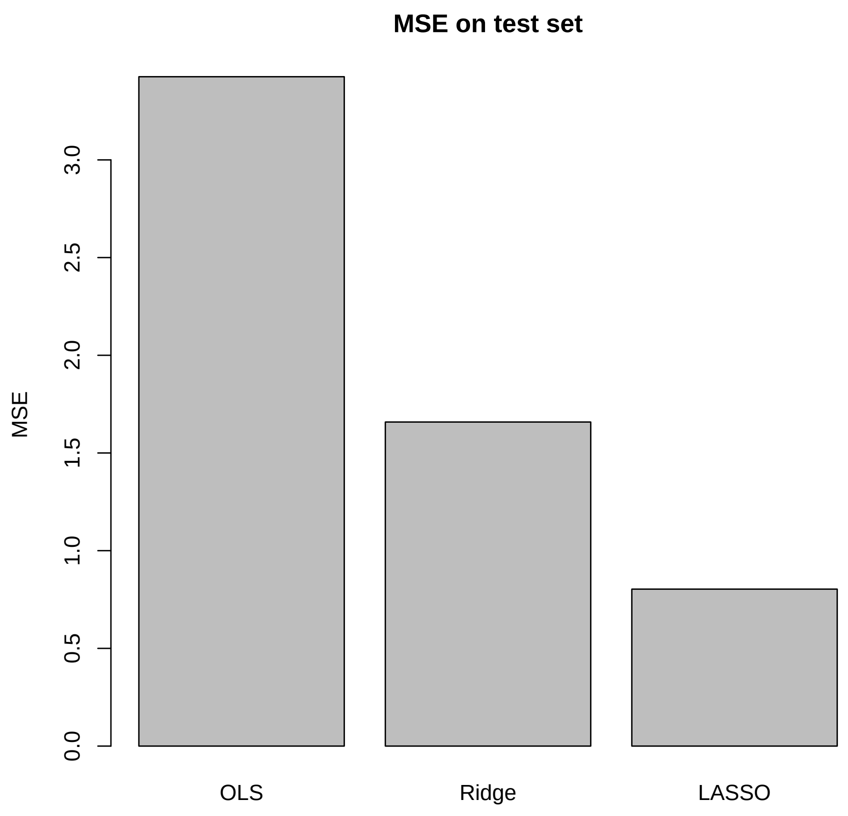


```
#plot LASSO vs LM
plot(y_test , lasso.predict ,
      xlim=c(-5,5), ylim=c(-5,5), pch=20, col="red",
      xlab="Ground Truth", ylab="Prediction",
      main="Ridge Regression (lambda.min)")
#superpose basic LM prediction for comparison
points(y_test , lm.predict,
      xlim=c(-5,5), ylim=c(-5,5), pch=21, col="black")
abline(a=0, b=1, lwd=3, lty=2)
legend("topleft", legend = c("LASSO", "LM"),
       pch=c(20,21), col=c("red","black"))
```



And let us compare the MSE on the test set.

```
barplot(c(lm.test.MSE,ridge.test.MSE,lasso.test.MSE),
        main="MSE on test set", ylab="MSE",
        names.arg=c("OLS", "Ridge", "LASSO"))
```



One can see that LASSO and Ridge are doing quite much better than OLS. On this data-set, LASSO is doing quite a lot better than ridge regression; that is far from always being the case, even in large p small n settings.