

# Regression - Regularized Regression

## A. Burnetas

In this note we will apply least squares regression and two regularized methods, ridge and lasso, on the ozone data set, using both direct matrix calculations (for the LSE and ridge methods) as well as appropriate R packages.

We first import the dataset:

```
ozone=read.csv("ozone.data", sep="\t")
```

Split dataset to training sample and testing sample, sizes=71 and 40, respectively.

```
Ntrain=71;
Ntest=40;
trainindex=sample(1:Ntrain+Ntest, Ntrain)
ozonetrain=ozone[trainindex,]
ozonetest=ozone[-trainindex,]
```

## 1 Ordinary Least Square Estimation

### 1.1 Direct computation

Create  $Y$  and  $X$  from training set(with column of 1)

```
y=as.vector(ozonetrain$ozone)
X=as.matrix(ozonetrain[,2:4])
N=dim(X)[1]
p=dim(X)[2]
X1=as.matrix(cbind(rep(1,N),X))
```

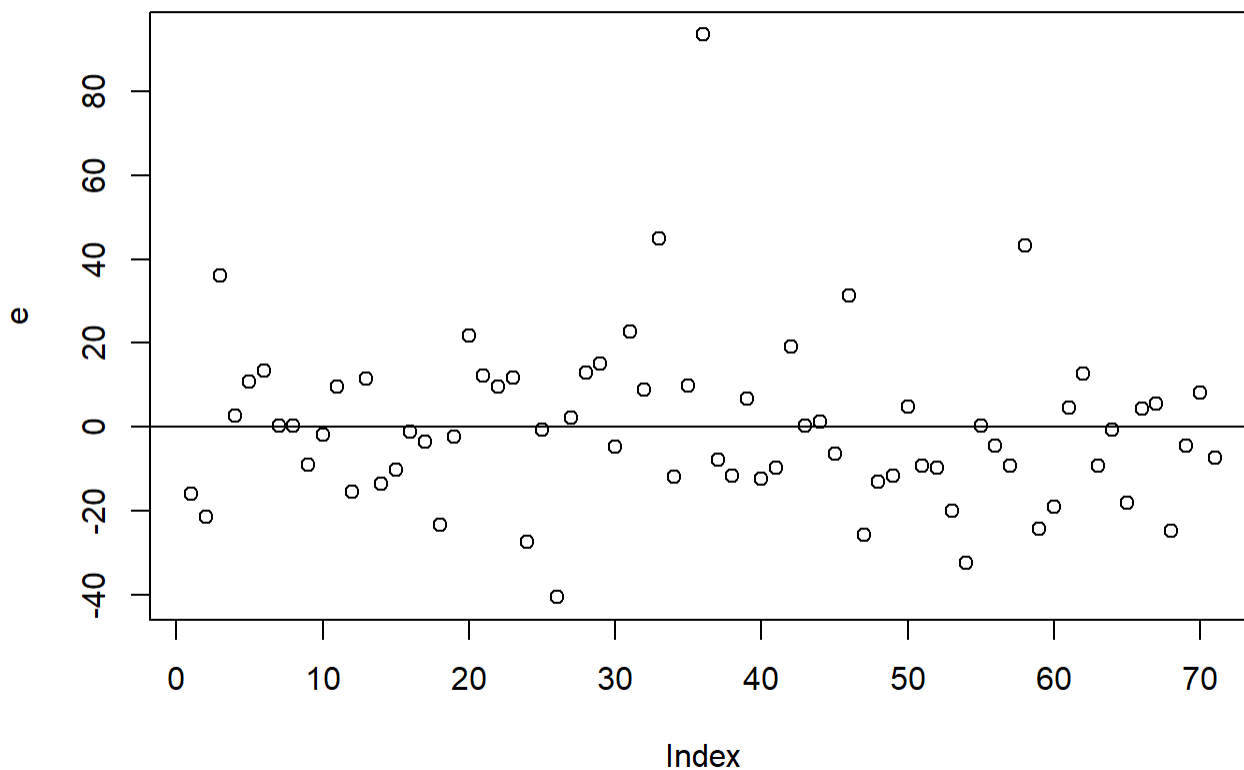
Standard LSE method (no scaling no centering). Model  $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \epsilon$ .

LSE formulas :  $\hat{\beta} = Gy$ ,  $\hat{y} = X\hat{\beta} = Hy$ , where  $G = (X^T X)^{-1} X^T$  and  $H = X(X^T X)^{-1} X^T$ .

```
xtx=t(X1)%*%X1
xtxin=solve(xtx)
Gls=xtxin%*%t(X1)
H=X1%*%Gls
blse=Gls%*%y
yhatlse=H%*%y
```

Calculate residuals in training set

```
e=y-yhatlse
plot(e)
abline(0,0)
```



Calculate predictions on testing set, and compute the mean prediction error:

```
ytest=as.vector(ozonetest$ozone)
Xtest=as.matrix(ozonetest[,2:4])
Ntest=dim(Xtest)[1]
Xtest1=as.matrix(cbind(rep(1,Ntest),Xtest))
ytesthatlse=Xtest1%*%blse
predresid=ytest - ytesthatlse
mpe=t(predresid)%*%predresid/Ntest
mpe
```

```
##      [,1]
## [1,] 639.27
```

## 1.2 R Linear Model function

Before we proceed, we repeat the previous computations using the `lm` function of R, which applies ordinary least squares estimation in a linear model. We first define a model formula and then call the `lm` function on the

model. There are several ways to do this. The fastest is

```
model1=y~X
```

This method works because X is an object of type matrix. This formula is equivalent to

```
model2=y ~ X[,1] + X[,2] + X[,3]
```

We can also use the column names of X:

```
colnames(X)
```

```
## [1] "radiation" "temperature" "wind"
```

and define

```
model3= y~ X["radiation"] + X["temperature"] + X["wind"]
```

Since X is a matrix and not a dataframe, we cannot use the notation X\$radiation to refer to the radiation column.

To check that all model definitions are equivalent, we call the lm function on each of them

```
lm(model1)
```

```
##  
## Call:  
## lm(formula = model1)  
##  
## Coefficients:  
## (Intercept)    Xradiation  Xtemperature      Xwind  
##   -104.80622      0.08292      2.12650     -3.71606
```

```
lm(model2)
```

```
##  
## Call:  
## lm(formula = model2)  
##  
## Coefficients:  
## (Intercept)      X[, 1]      X[, 2]      X[, 3]  
##   -104.80622      0.08292      2.12650     -3.71606
```

```
lm(model3)
```

```
##
## Call:
## lm(formula = model3)
##
## Coefficients:
##      (Intercept)      X[, "radiation"]      X[, "temperature"]      X[, "wind"]
##      -104.80622           0.08292           2.12650           -3.71606
```

A more appropriate way, from the point of view of R programming, to define the model is to consider the training and the test set as two subsets of the original data set ozone, i.e., as two dataframes that contain both the dependent and the independent variables. The two dataframes have the same number of columns and the same column names and differ in the number of rows. We can then define the model formula as a relationship between variables (column names), regardless of the dataset that they refer to:

```
model4 = ozone~radiation+temperature+wind
```

Now, when we call the lm function we must specify not only the model formula, but also the dataset that contains the actual data. This must be a dataframe that includes all the variables mentioned in the formula. This is done with the option data= inside the lm function:

```
l1 = lm(model4, data=ozonetrain)
l1
```

```
##
## Call:
## lm(formula = model4, data = ozonetrain)
##
## Coefficients:
## (Intercept)      radiation      temperature      wind
## -104.80622       0.08292       2.12650       -3.71606
```

Note that in all the model definitions above, we used the original X from the training set without adding the column of 1. This is so because the lm function automatically assumes that the regression model has a constant term and estimates it together with the slope coefficients.

We verify that all give the same estimates, which also coincide with those that we obtained with direct computation

```
blse
```

```
##              [,1]
##      -104.80621538
## radiation    0.08292363
## temperature  2.12649779
## wind        -3.71605662
```

The lm function returns much more information in addition to the LS estimates:

```
names(l1)
```

```
## [1] "coefficients" "residuals"      "effects"      "rank"
## [5] "fitted.values" "assign"         "qr"           "df.residual"
## [9] "xlevels"      "call"          "terms"        "model"
```

```
bhat=l1$coefficients
ytrainhat=l1$fitted.values
```

If combined with the summary function it gives further information, related to statistical inference:

```
ls1=summary(l1)
names(ls1)
```

```
## [1] "call"          "terms"          "residuals"      "coefficients"
## [5] "aliased"       "sigma"          "df"             "r.squared"
## [9] "adj.r.squared" "fstatistic"     "cov.unscaled"
```

The outputs of `lm` and `summary(lm)` are lists, the elements of which can be retrieved directly. For example the `coefficients` element of `l1` gives the vector of LS estimates, however the `coefficients` element of `ls1` is a matrix with the estimates as well as their standard deviations, the t-statistics and the p-values of the individual significance tests:

```
l1$coefficients
```

```
## (Intercept) radiation temperature wind
## -104.80621538 0.08292363 2.12649779 -3.71605662
```

```
ls1$coefficients
```

```
##           Estimate Std. Error t value Pr(>|t|)
## (Intercept) -104.80621538 34.98993575 -2.995325 3.839780e-03
## radiation    0.08292363 0.03090748 2.682963 9.183172e-03
## temperature  2.12649779 0.37776869 5.629100 3.866905e-07
## wind         -3.71605662 0.86889511 -4.276761 6.155631e-05
```

The output of `lm` by default refers to the training set. For example `l1$fitted.values` and `l1$residuals` compute  $\hat{y}$  and  $y - \hat{y}$  for the training set. To compute the predictions for the test set we must use the function `predict`, after running `lm`. To specify that the predictions must be computed for the test set we use the `newdata` option:

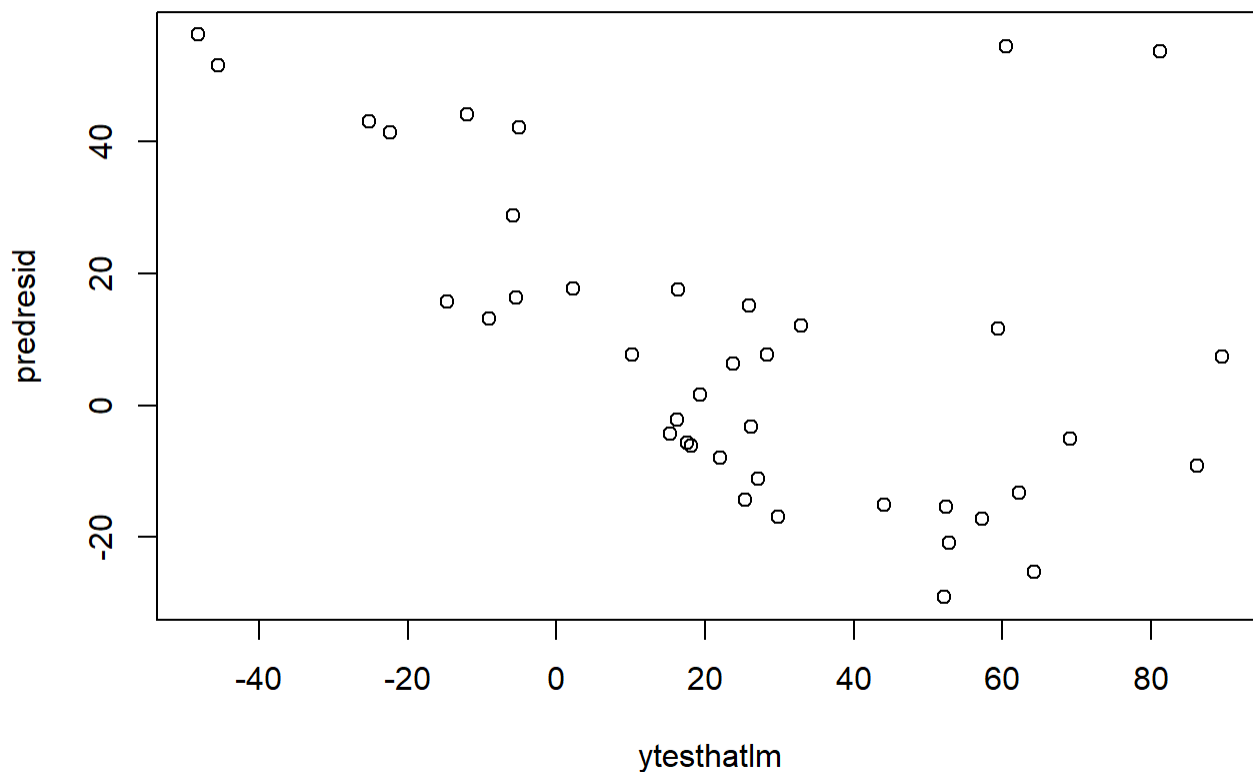
```
ytestthatlm=predict(l1, newdata = ozonetest)
```

We can now compute the prediction errors:

```
predresid=ozonetest$ozone - ytestthatlm
```

and draw a plot of the residuals against the predicted values:

```
plot(predresid~ytestthatlm)
```



We can also compute the mean square prediction error

```
mpe=t(predresid)%*%predresid/length(predresid)
mpe
```

```
##      [,1]
## [1,] 639.27
```

Note that the results are identical to those obtained by direct calculation, as expected.

## 1.3 Data Preprocessing: Scaling and Centering the independent variables

Scaling the independent variables is useful, since it makes the regression coefficients independent of the particular units of each variable. Scaling means applying a linear transformation to a variable such that the regression coefficient does not change when the units of measurement change.

There are two frequently used scaling methods: normalization and standardization. Normalizing the vector of values of a variable  $X$  means applying the transformation

$$X'_i = \frac{X_i - \min_j X_j}{\max_j X_j - \min_j X_j}, i = 1, \dots, N$$

where  $N$  is the length of the training set.

Standardization is the usual transformation applied to normal random variables, now using the sample mean and standard deviation of  $X$ :

$$X'_i = \frac{X_i - \bar{X}}{s(X)}$$

Centering is another transformation that serves a different purpose. Centering an independent variable is equivalent to taking the differences of the variable values from the from its mean:

$$X_i^c = X_i - \bar{X}, i = 1, \dots, N.$$

Centering is useful if we recall the property of the LS estimates in a regression model, that the regression hyperplane passes through the center of the training set. This means that in the regression model

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p + \epsilon$$

the LS estimates  $\hat{\beta}$  satisfy the relation:

$$\bar{Y} = \hat{\beta}_0 + \hat{\beta}_1 \bar{X}_1 + \dots + \hat{\beta}_p \bar{X}_p.$$

Therefore if we center the variables and apply the regression model:

$$Y = \beta_0^c + \beta_1^c X_1^c + \dots + \beta_p^c X_p^c + \epsilon$$

then  $\bar{X}_j^c = 0, j = 1, \dots, p$ , thus  $\bar{Y} = \hat{\beta}_0^c$ . In other words in a dataset with centered independent variables the estimate of the intercept is equal to the mean of  $Y$ , thus only the  $p$  coefficients need to be estimated. Furthermore it is easy to see that  $\hat{\beta}_j^c = \hat{\beta}_j, j = 1, \dots, p$ .

Before we finish the discussion on rescaling transformations we must make the following observations:

**(1)** First, if some independent variables are defined as functions of other variables (for example in a polynomial regression model  $Y = \beta_0 + \beta_1 X + \beta_2 X^2$  we have  $X_2 = X_1^2$ ), then we must center each variable *separately*.

In the above example the correct centering is  $X_1^c = X_1 - \bar{X}_1$  and  $X_2^c = X_2 - \bar{X}_2 = X_2 - \overline{X_1^2}$  and not  $X_2^c = X_1^2 - \bar{X}_1^2$ . There is a difference because  $\overline{X_1^2} \neq \bar{X}_1^2$ .

**(2)** Second, when we apply the estimates to make predictions from a testing set, we must first apply *the same transformations* (normalization, standardization or centering) with the numerical values for  $\min(X), \max(X), \bar{X}, sd(X)$  that were used in the training set, and *not* apply the transformation from scratch. For example if in the training set  $\bar{X} = 5$ , in the testing set  $\bar{X} = 6$ , and we apply centering  $X^c = X - 5$  in the training set, then for making predictions we must also apply  $X = X - 5$  in the testing set, although it is not the mean of  $X$  for this set. Alternatively, if the training and test sets come from splitting a larger dataset, then any rescaling must be performed on the original data before splitting.

**(3)** If we need both rescaling and centering the independent variables, then standardization is the most

appropriate transformation, since it accomplishes both purposes in a single step.

To verify the above numerically, we can apply standardization to the training set of ozone dataset and rerun the `lm` function for training and prediction. First create the standardized training set:

```
ozonetrainc=ozonetrain
ozonetrainc$radiation=(ozonetrainc$radiation-mean(ozonetrain$radiation))/sd(ozonetrain$radiation)
ozonetrainc$temperature=(ozonetrainc$temperature-mean(ozonetrain$temperature))/sd(ozonetrain$temperature)
ozonetrainc$wind=(ozonetrainc$wind-mean(ozonetrain$wind))/sd(ozonetrain$wind)
```

Check that the independent variables are now standardized:

```
apply(ozonetrainc[,2:4], 2, mean)
```

```
##      radiation  temperature      wind
## -1.390226e-16 -5.394501e-16 -5.434754e-17
```

```
apply(ozonetrainc[,2:4], 2, sd)
```

```
##      radiation temperature      wind
##           1           1           1
```

We next run the linear model on the standardized training set  $Y = \beta_0^s + \beta_1^s X_1^s + \beta_2^s X_2^s + \beta_3^s X_3^s$ .

```
l1c=lm(model4, data=ozonetrainc)
betac=l1c$coefficients
betac
```

```
## (Intercept)  radiation temperature      wind
##  46.971831    6.786065   16.626547  -12.340529
```

We first verify that the new intercept is equal to the mean of the dependent variable:

```
mean(ozonetrainc$ozone)
```

```
## [1] 46.97183
```

We also compare the coefficients with those of the original model:

```
blse
```

```
##           [,1]
##      -104.80621538
## radiation    0.08292363
## temperature  2.12649779
## wind        -3.71605662
```

and verify that  $\beta_j^s = sd(X_j)\beta_j$ :

```
blse[2]*sd(ozonetrain$radiation)
```

```
## [1] 6.786065
```

```
blse[3]*sd(ozonetrain$temperature)
```

```
## [1] 16.62655
```

```
blse[4]*sd(ozonetrain$wind)
```

```
## [1] -12.34053
```

Finally, we use the estimates of the standardized training set to make predictions on the test set. We must first transform the test set using the standardization transformation of the training set, as discussed above:

```
ozonetestc=ozonetest
ozonetestc$radiation=(ozonetestc$radiation-mean(ozonetrain$radiation))/sd(ozonetrain$radiation)
ozonetestc$temperature=(ozonetestc$temperature-mean(ozonetrain$temperature))/sd(ozonetrain$temperature)
ozonetestc$wind=(ozonetestc$wind-mean(ozonetrain$wind))/sd(ozonetrain$wind)
```

(Pay attention where we use ozonetestc and ozonetrain in the above commands.)

The predictions are found in the same way as before and we see that they are identical with those in the nonstandardized model

```
ytestthatlmc=predict(l1c, newdata = ozonetestc)
ytestthatlmc
```

```
##      1      2      3      4      5      6      7
## 25.925807 28.358161 18.087928 10.257092 26.252219 -22.414988 -48.207039
##      8      9     10     11     12     13     14
## 27.104832 25.402770 22.011691 -25.131255 16.251484 16.407513 -45.503240
##     15     16     17     18     19     20     21
## 23.762391 -5.360462 -14.725206 15.277144 -9.062508 -12.053556 -5.845535
##     22     23     24     25     26     27     28
## 32.967616 60.497556 52.444490 44.052155 59.427780 64.248773 52.110848
##     29     30     31     32     33     34     35
## 19.403284 -5.070436 2.296595 17.644308 29.892771 81.261828 62.323436
##     36     37     38     39     40
## 52.822361 69.110876 57.226103 86.260623 89.561033
```

```
ytestthatlm
```

```
##      1      2      3      4      5      6      7
## 25.925807 28.358161 18.087928 10.257092 26.252219 -22.414988 -48.207039
##      8      9     10     11     12     13     14
## 27.104832 25.402770 22.011691 -25.131255 16.251484 16.407513 -45.503240
##     15     16     17     18     19     20     21
## 23.762391 -5.360462 -14.725206 15.277144 -9.062508 -12.053556 -5.845535
##     22     23     24     25     26     27     28
## 32.967616 60.497556 52.444490 44.052155 59.427780 64.248773 52.110848
##     29     30     31     32     33     34     35
## 19.403284 -5.070436 2.296595 17.644308 29.892771 81.261828 62.323436
##     36     37     38     39     40
## 52.822361 69.110876 57.226103 86.260623 89.561033
```

## 2 Regularized Regression Methods : Ridge and Lasso Regression

### 2.1 Ridge Regression - Direct Computations

For ridge regression we can apply the formula

$$\beta^r(\lambda) = (X^T X + \lambda I)^{-1} X^T Y$$

where  $X$  is the standardized matrix of independent variables (without a column of 1, since we do not estimate the constant term which is equal to  $\bar{Y}$ ).

The standardized matrix  $X$  may be obtained from the 2,3,4 columns of the training set, converted to matrix type:

```
Xtrain=as.matrix(ozonetrain[,2:4])
ytrain=ozonetrain$ozone
```

Since we are going to estimate the ridge coefficients and make predictions for various values of  $\lambda$ , we create two functions which perform the corresponding task.

```
betaridge=function(X,y,l)
{
  #Inputs : y independent variable vector
  #         X matrix of standardized independent variables
  #         l regularization parameter
  #Output  beta = vector of coefficients (without a constant term)
  dx=dim(X)
  N=dx[1]
  p=dx[2]
  beta = solve(t(X)%*%X + l*diag(p))%*%t(X)%*%y
  return(beta)
}
```

Applying the function to the training set data with  $\lambda = 0$  we obtain

```
br0=betaridge(Xtrainc,ytrain,0)
br0
```

```
##           [,1]
## radiation    6.786065
## temperature 16.626547
## wind        -12.340529
```

which is equal to  $\beta^c$  computed before:

```
betac
```

```
## (Intercept) radiation temperature      wind
##  46.971831    6.786065   16.626547  -12.340529
```

Applying the function to the training set data with  $\lambda = 2$  we obtain

```
br=betaridge(Xtrainc,ytrain,2)
```

The euclidian norm of the beta vector is

```
t(br)%*%br
```

```
##           [,1]
## [1,] 458.2601
```

compared to the euclidian norm of the original beta vector

```
t(betac[2:4])%*%betac[2:4]
```

```
##           [,1]
## [1,] 474.7814
```

We next create a function that takes as arguments  $y$  and  $Xc$  from the training set, as well as the corresponding data from the test set, computes the ridge betas and computes the predictions from the test set and the corresponding mean prediction error. **Note that** we must add  $\beta_0^c = \bar{y}_{train}$  to the prediction vector, because the coefficients we calculated do not include the constant term.

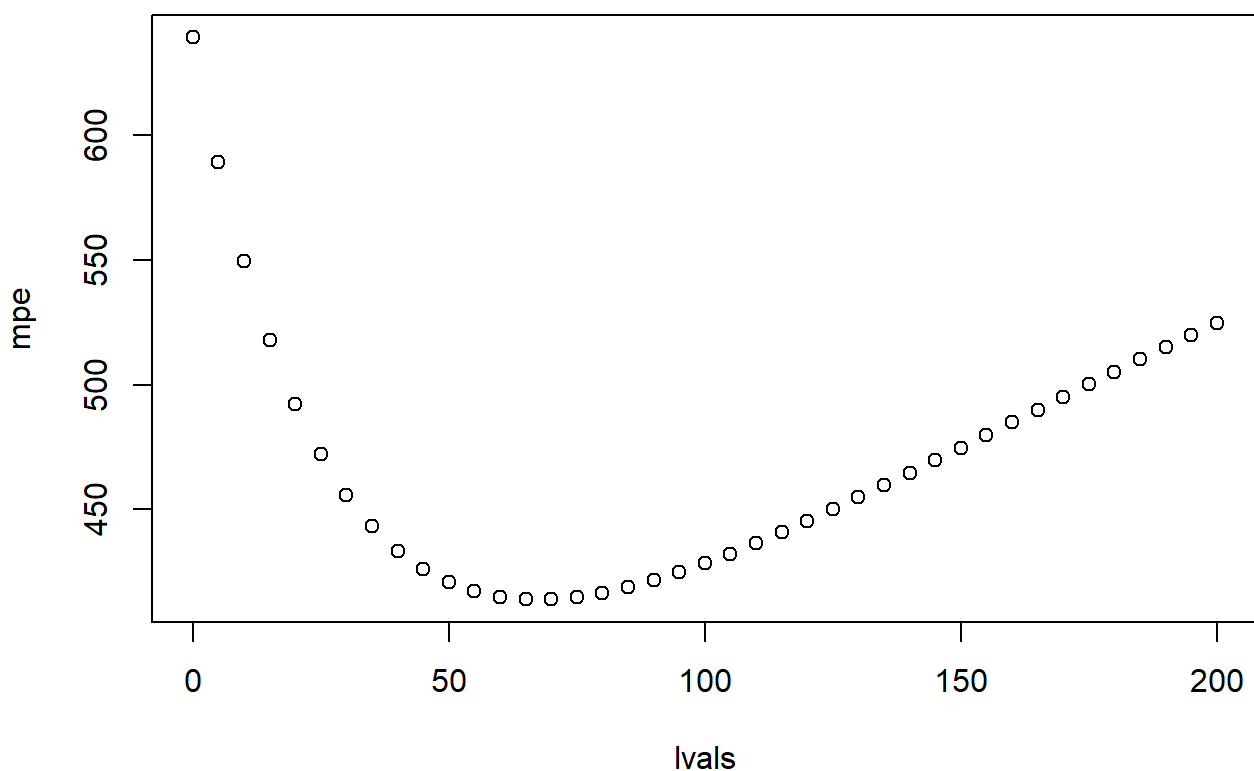
```
mperidge=function(Xtrain, ytrain, Xtest, ytest, l)
{
  beta=betaridge(Xtrain, ytrain, l)
  yhattest=mean(ytrain)+Xtest%*%beta
  predres=ytest-yhattest
  mpe=t(predres)%*%predres/length(predres)
  return(mpe)
}
```

We now apply the mperidge function for values of  $\lambda$  in the interval  $[0, 200]$  and plot the results:

```
Xtestc=as.matrix(ozonetestc[,2:4])
ytest=ozonetestc$ozone
lvals=seq(0,200,5)
nl=length(lvals)
mpe=rep(0,nl)
for (i in 1:nl)
{
  mpe[i]=mperidge(Xtrainc, ytrain, Xtestc, ytest, lvals[i])
}
mpe
```

```
## [1] 639.2700 589.3376 549.5914 517.7947 492.3394 472.0196 455.9041 443.2592
## [9] 433.4978 426.1441 420.8092 417.1724 414.9676 413.9726 414.0011 414.8960
## [17] 416.5244 418.7735 421.5472 424.7635 428.3521 432.2527 436.4136 440.7900
## [25] 445.3434 450.0406 454.8529 459.7553 464.7264 469.7475 474.8024 479.8771
## [33] 484.9596 490.0394 495.1074 500.1558 505.1780 510.1682 515.1214 520.0335
## [41] 524.9009
```

```
plot(mpe~lvals)
```



## 2.2 Regularized regression with glmnet library

A widely used R library for regularized regression and classification is glmnet, developed by Friedman et al (<https://glmnet.stanford.edu/articles/glmnet.html>). To use the library it must first be installed in R with the command

`install.packages("glmnet")`, and then called using the library command:

```
library(glmnet)
```

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

```
## Loaded glmnet 4.1-8
```

The library has a great variety of functions to perform, analyze and plot regularized variations of Generalized Linear Models and not only regression. For full description of the package, the reader may refer to the guide or the webpage above. Here we use the `glmnet()` function which computes the regularized coefficients under a generalized elastic-net penalty which combines the ridge and lasso penalties:

$$\lambda \left( \frac{1-\alpha}{2} \|\beta\|_2^2 + \alpha \|\beta\|_1 \right)$$

Note that  $\alpha = 1$  corresponds to lasso and  $\alpha = 0$  to ridge regularization.

We call the `glmnet()` function with the standardized matrix  $X$  and the vector  $y$  from the training set, using  $\alpha = 0$  and the appropriate value of  $\lambda$ . First we verify the results for the no-regularization model with  $\lambda = 0$ :

```
bnet0=glmnet(Xtrainc, ytrain, alpha=0, lambda = 0)
bnet0$beta
```

```
## 3 x 1 sparse Matrix of class "dgCMatrix"
##              s0
## radiation    6.785835
## temperature 16.626863
## wind        -12.340399
```

```
betac
```

```
## (Intercept)  radiation temperature      wind
##  46.971831    6.786065   16.626547  -12.340529
```

We observe that the results obtained with `glmnet` are slightly different from the (correct) values obtained from the `lm` function. The reason is that `glmnet` computes the coefficients using an iterative optimization algorithm similar to gradient descent, which converges to the optimal values of the coefficients asymptotically and is not 100% precise when it stops after a number of iterations. The reason is that for values  $\alpha > 0$  there is no analytical expression for the optimal coefficients such as that for  $\alpha = 0$  (ridge regression), therefore they must be calculated numerically.