Task 3

for Advanced Methods for Regression and Classification

Teodor Chakarov

15.11.2022

Contents

Exercise 1

Data Preprocess

Let's load our College data for this exercise.

```
library(MASS)
library(glmnet)
```

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

```
## Loaded glmnet 4.1-4
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:MASS':
##
##     select
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library("cvTools")
```

```
## Loading required package: lattice

## Loading required package: robustbase
```

```
compute_rmse <- function(y_true, y_pred) {

   return(sqrt(mean((y_true-y_pred)^2)))
}
```

```
data(College ,package="ISLR")
data_col <- College
str(data_col)
```

```
## 'data.frame':    777 obs. of  18 variables:
##  $ Private    : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Apps       : num  1660 2186 1428 417 193 ...
##  $ Accept     : num  1232 1924 1097 349 146 ...
##  $ Enroll     : num  721 512 336 137 55 158 103 489 227 172 ...
##  $ Top10perc  : num  23 16 22 60 16 38 17 37 30 21 ...
##  $ Top25perc  : num  52 29 50 89 44 62 45 68 63 44 ...
##  $ F.Undergrad: num  2885 2683 1036 510 249 ...
##  $ P.Undergrad: num  537 1227 99 63 869 ...
##  $ Outstate   : num  7440 12280 11250 12960 7560 ...
##  $ Room.Board : num  3300 6450 3750 5450 4120 ...
##  $ Books      : num  450 750 400 450 800 500 500 450 300 660 ...
##  $ Personal   : num  2200 1500 1165 875 1500 ...
##  $ PhD        : num  70 29 53 92 76 67 90 89 79 40 ...
##  $ Terminal   : num  78 30 66 97 72 73 93 100 84 41 ...
##  $ S.F.Ratio  : num  18.1 12.2 12.9 7.7 11.9 9.4 11.5 13.7 11.3 11.5 ...
##  $ perc.alumni: num  12 16 30 37 2 11 26 37 23 15 ...
##  $ Expend     : num  7041 10527 8735 19016 10922 ...
##  $ Grad.Rate  : num  60 56 54 59 15 55 63 73 80 52 ...
```

Train and Test split

We need to predict the attribute ***Apps*** (which will be our dependent variable). First we will get rid of the columns ***"Accept"*** and ***"Enroll"***. We have to convert our categorical variables to numeric one.

```
data_col <- data_col %>%
  mutate(Apps = log(Apps))
```

```
data_col2 <- data_col[ , -which(names(data_col) %in% c("Accept", "Enroll"))]
data_col2$Private <- as.numeric(data_col2$Private)
str(data_col2)
```

```
## 'data.frame':    777 obs. of  16 variables:
##  $ Private    : num  2 2 2 2 2 2 2 2 2 2 ...
##  $ Apps       : num  7.41 7.69 7.26 6.03 5.26 ...
##  $ Top10perc  : num  23 16 22 60 16 38 17 37 30 21 ...
##  $ Top25perc  : num  52 29 50 89 44 62 45 68 63 44 ...
##  $ F.Undergrad: num  2885 2683 1036 510 249 ...
##  $ P.Undergrad: num  537 1227 99 63 869 ...
##  $ Outstate   : num  7440 12280 11250 12960 7560 ...
##  $ Room.Board : num  3300 6450 3750 5450 4120 ...
##  $ Books      : num  450 750 400 450 800 500 500 450 300 660 ...
##  $ Personal   : num  2200 1500 1165 875 1500 ...
##  $ PhD        : num  70 29 53 92 76 67 90 89 79 40 ...
##  $ Terminal   : num  78 30 66 97 72 73 93 100 84 41 ...
##  $ S.F.Ratio  : num  18.1 12.2 12.9 7.7 11.9 9.4 11.5 13.7 11.3 11.5 ...
##  $ perc.alumni: num  12 16 30 37 2 11 26 37 23 15 ...
##  $ Expend     : num  7041 10527 8735 19016 10922 ...
##  $ Grad.Rate  : num  60 56 54 59 15 55 63 73 80 52 ...
```

```
data_col2$Private[data_col2$Private == 1] <- 0
data_col2$Private[data_col2$Private == 2] <- 1
```

I am picking 2/3rd random data indexes of the data for the training set and 1/3 for the testing.

```
set.seed(16)

## 2/3 of the sample size
smp_size <- floor(round(nrow(data_col2)*2/3))
train_ind <- sample(seq_len(nrow(data_col2)), size = smp_size)
smp_size
```

```
## [1] 518
```

Getting the sample size. Lets now split the data into train and test while also separating the dependent variable with the independent once.

```
train <- data_col2[train_ind, ]
test <- data_col2[-train_ind, ]

# Setting the y to be "Apps"
y_train = train[ , which(names(train) %in% c("Apps"))]
y_test = test[ , which(names(test) %in% c("Apps"))]

# Removing the predictive variable from the training and testing sets.
x_train = train[ , -which(names(train) %in% c("Apps"))]
x_test = test[ , -which(names(test) %in% c("Apps"))]
```

Linear Regression

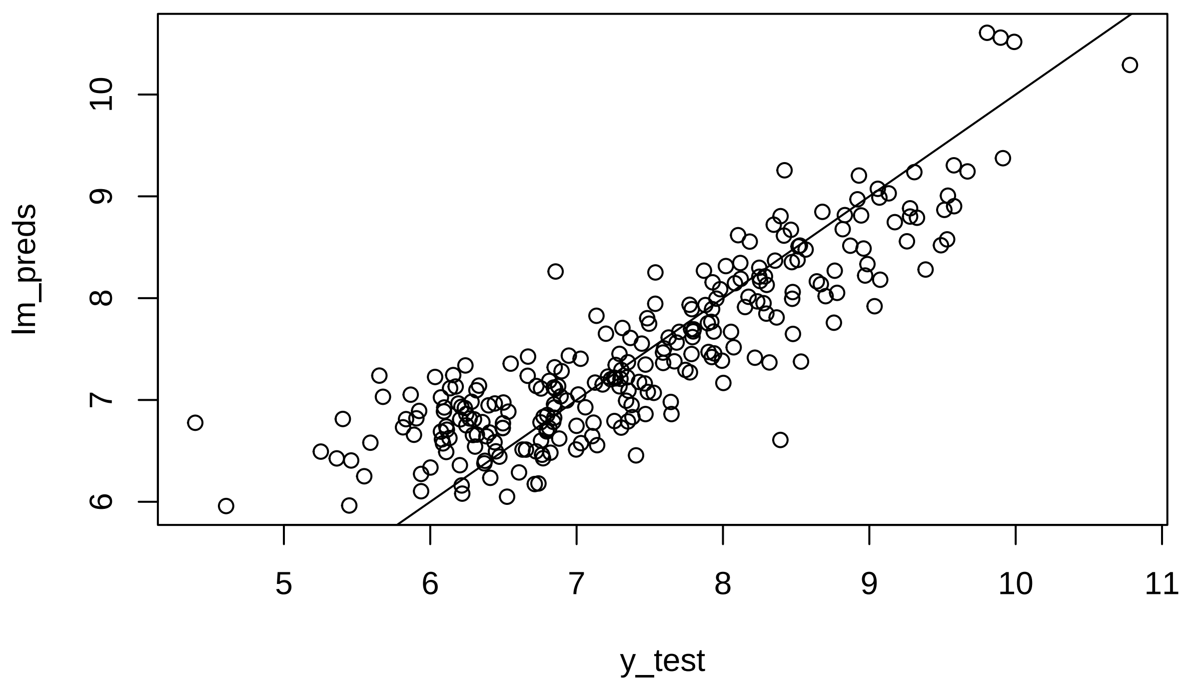
```
lin_reg <- lm(y_train ~ ., data = x_train)

cv_model <- cvFit(lm, formula=y_train ~ ., data=x_train, y=y_train, cost=rmspe, K=5, seed = 16)
cv_model
```

```
## 5-fold CV results:
##        CV
## 0.5918423
```

```
lm_preds <- predict(lin_reg, x_test)

plot(y_test,lm_preds)
abline(c(0,1))
```



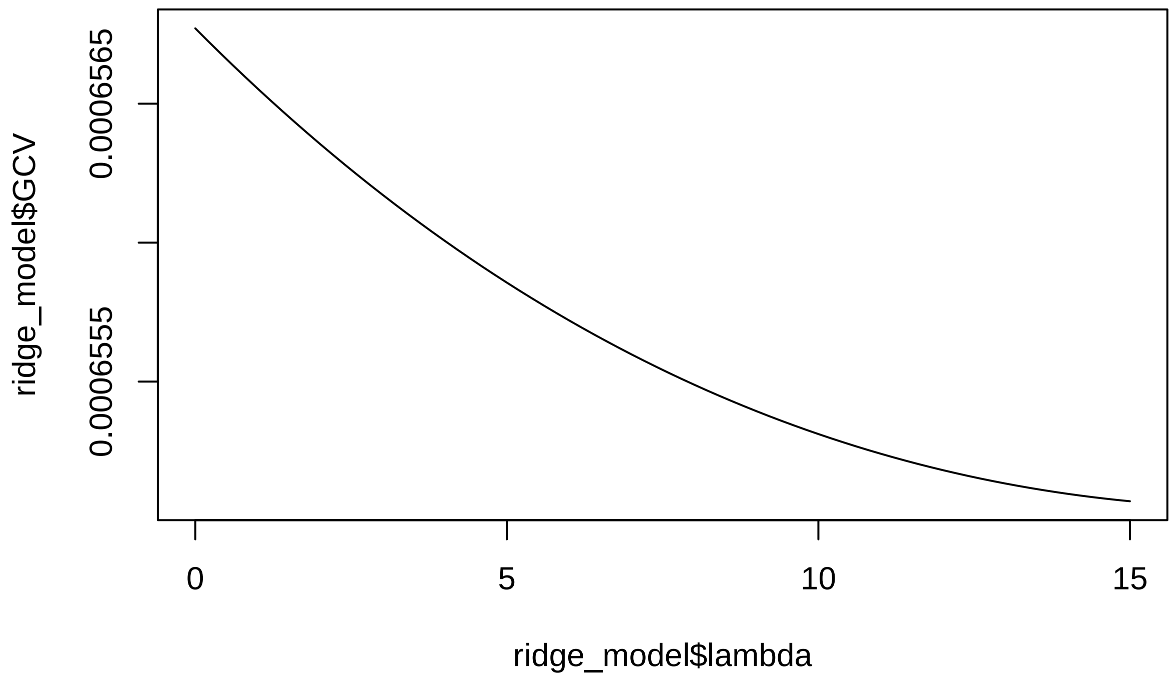

```
compute_rmse(y_test, lm_preds)
```

```
## [1] 0.5647091
```

Ridge Regression

Lets fit our data in Ridge Regression

```
ridge_model <- lm.ridge(y_train ~., data=x_train, lambda=seq(0,15, by=0.2))
plot(ridge_model$lambda,ridge_model$GCV,type="l")
```



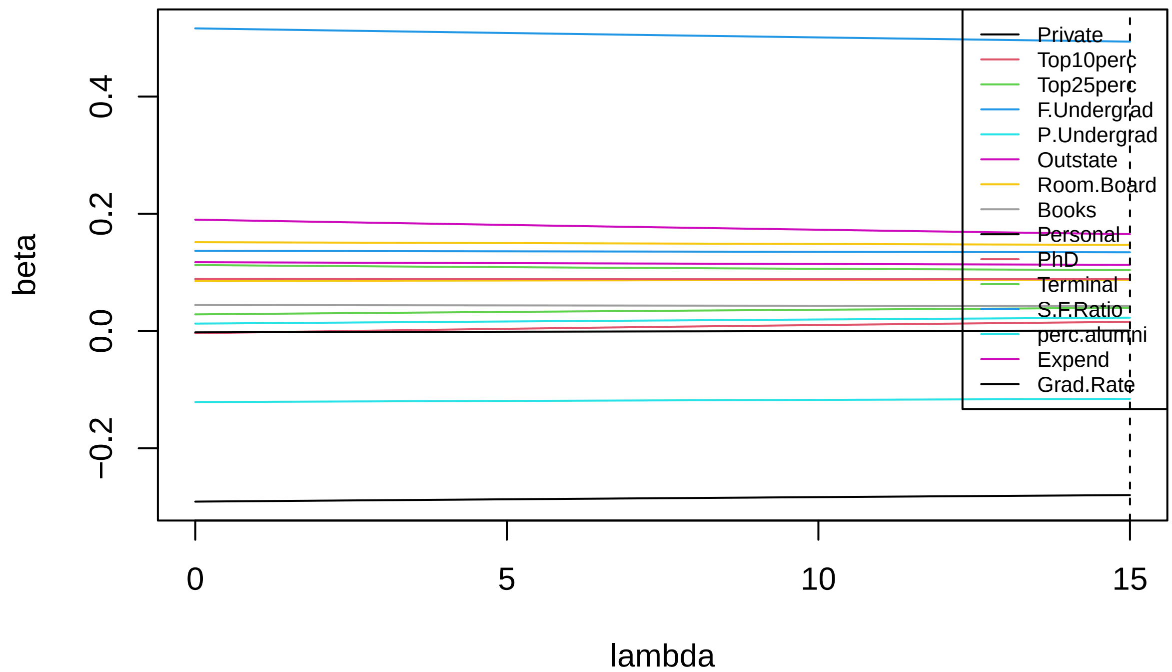

```
ridge_model$lambda[which.min(ridge_model$GCV)]
```

```
## [1] 15
```

We can see now how our GCV score lowers around 2,2 and then it raises again. So I will take the lambda at 2,2

```
lambda_opt <- ridge_model$lambda[which.min(ridge_model$GCV)]
```

```
plot(0,0,xlim=range(ridge_model$lambda),ylim=range(ridge_model$coef),
     type="n",xlab="lambda",ylab="beta")
for(i in 1:nrow(ridge_model$coef))
{
   lines(ridge_model$lambda,ridge_model$coef[i,],col=i)
}
legend("topright", legend=rownames(ridge_model$coef), lty=rep(1,14),col=1:14, cex=0.65)
abline(v=lambda_opt, lty=2)
```



We can see based on the lines the beta coefficients for each attribute based on certain lambda value. And on the horizontal line sits our optimal lambda value and respectively the coefficients for the attributes.

```
ridge_model$coef[,12]
```

```
##       Private     Top10perc     Top25perc   F.Undergrad   P.Undergrad
## -0.2891536810 -0.0002507967  0.1108485117  0.5127294466  0.0143154311
##      Outstate    Room.Board         Books      Personal           PhD
##  0.1859480666  0.0857165330  0.0440693520 -0.0017381819  0.0885806287
##      Terminal     S.F.Ratio   perc.alumni        Expend     Grad.Rate
##  0.0303417982  0.1364038779 -0.1202256716  0.1166888888  0.1508709347
```

Those are the regression coefficients for our optimal model.

Lets make some predictions…

```
# Prediction with Ridge:
ridge_model2 <- lm.ridge(y_train ~., data=x_train, lambda = lambda_opt, )
ridge_model2$coef # coefficients for scaled x
```

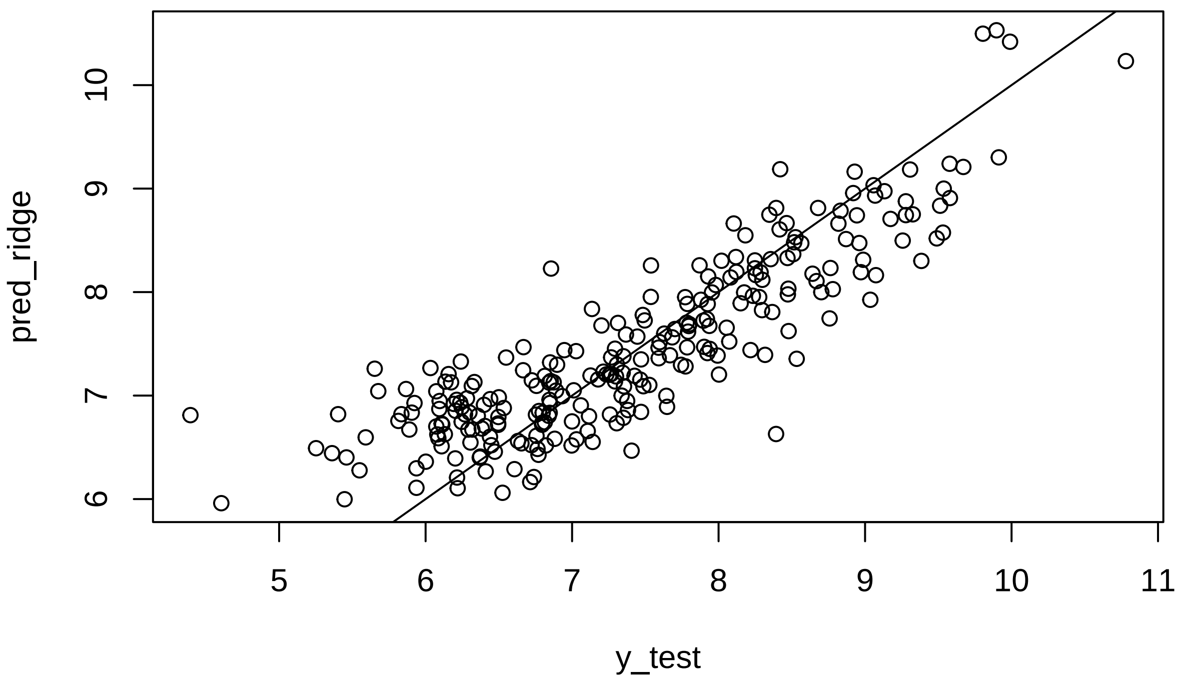
```
##       Private     Top10perc     Top25perc   F.Undergrad   P.Undergrad
## -0.2797740218  0.0156767633  0.1040576431  0.4936522841  0.0228819497
##      Outstate    Room.Board         Books      Personal           PhD
##  0.1652989392  0.0875836747  0.0426679135  0.0008956986  0.0884711007
##      Terminal     S.F.Ratio   perc.alumni        Expend     Grad.Rate
##  0.0396168570  0.1343167395 -0.1156500780  0.1129900233  0.1469606258
```

```r
ridge_coef <- coef(ridge_model2)
ridge_coef # coefficients in original scale + intercept
```

```
##                     Private     Top10perc     Top25perc   F.Undergrad 
##  4.643567e+00 -6.343287e-01  8.928825e-04  5.174666e-03  9.858868e-05 
##   P.Undergrad      Outstate    Room.Board         Books      Personal 
##  1.417093e-05  4.058391e-05  8.286644e-05  2.874775e-04  1.250199e-06 
##           PhD      Terminal     S.F.Ratio   perc.alumni        Expend 
##  5.488876e-03  2.632117e-03  3.331830e-02 -9.117464e-03  2.164376e-05 
##     Grad.Rate 
##  8.350890e-03
```

```r
pred_ridge <- as.matrix(cbind(rep(1,length(y_test)),x_test))%*%ridge_coef

plot(y_test,pred_ridge)
abline(c(0,1))
```




```r
compute_rmse(y_test, pred_ridge)
```

```
## [1] 0.5683723
```

In comparison from the previous exercise we have: - PCR: 0.5674299 - PLSR: 0.5672898 - Ridge Regression: 0.5648764

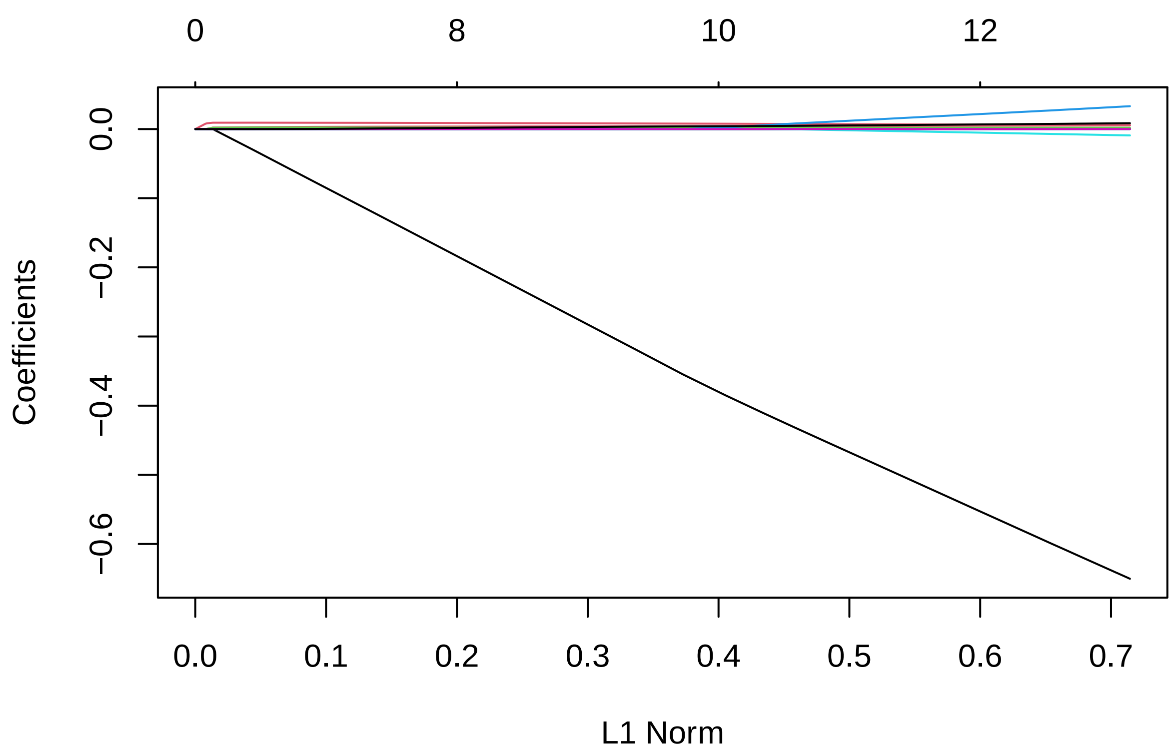
Lasso Regression

```
lasso <- glmnet(as.matrix(x_train),y_train)
print(lasso)
```

```
##
## Call:  glmnet(x = as.matrix(x_train), y = y_train)
##
##    Df  %Dev  Lambda
## 1   0  0.00 0.76590
## 2   1  9.08 0.69780
## 3   1 16.62 0.63580
## 4   1 22.88 0.57930
## 5   1 28.08 0.52790
## 6   1 32.40 0.48100
## 7   1 35.98 0.43830
## 8   2 39.45 0.39930
## 9   2 43.23 0.36380
## 10  2 46.36 0.33150
## 11  2 48.97 0.30210
## 12  2 51.13 0.27520
## 13  2 52.92 0.25080
## 14  4 54.51 0.22850
## 15  4 55.97 0.20820
## 16  4 57.19 0.18970
## 17  4 58.21 0.17290
## 18  4 59.05 0.15750
## 19  6 60.04 0.14350
## 20  7 61.23 0.13080
## 21  7 62.35 0.11910
## 22  8 63.35 0.10860
## 23  8 64.18 0.09891
## 24  8 64.87 0.09013
## 25  8 65.45 0.08212
## 26  8 65.93 0.07483
## 27  9 66.37 0.06818
## 28 10 66.89 0.06212
## 29 10 67.39 0.05660
## 30 12 67.85 0.05157
## 31 12 68.34 0.04699
## 32 12 68.75 0.04282
## 33 12 69.09 0.03901
## 34 12 69.37 0.03555
## 35 12 69.60 0.03239
## 36 12 69.79 0.02951
## 37 12 69.95 0.02689
## 38 12 70.08 0.02450
## 39 12 70.19 0.02233
## 40 12 70.29 0.02034
## 41 12 70.36 0.01853
## 42 12 70.42 0.01689
## 43 12 70.48 0.01539
## 44 13 70.52 0.01402
```

```
## 45 13 70.56 0.01278
## 46 13 70.59 0.01164
## 47 13 70.62 0.01061
## 48 13 70.64 0.00966
## 49 13 70.65 0.00881
## 50 13 70.67 0.00802
## 51 13 70.68 0.00731
## 52 13 70.69 0.00666
## 53 13 70.70 0.00607
## 54 13 70.71 0.00553
## 55 13 70.71 0.00504
## 56 13 70.72 0.00459
## 57 13 70.72 0.00418
## 58 13 70.73 0.00381
## 59 13 70.73 0.00347
## 60 13 70.73 0.00316
## 61 13 70.73 0.00288
## 62 13 70.73 0.00263
## 63 13 70.74 0.00239
## 64 13 70.74 0.00218
## 65 13 70.74 0.00199
## 66 13 70.74 0.00181
```

```
plot(lasso)
```



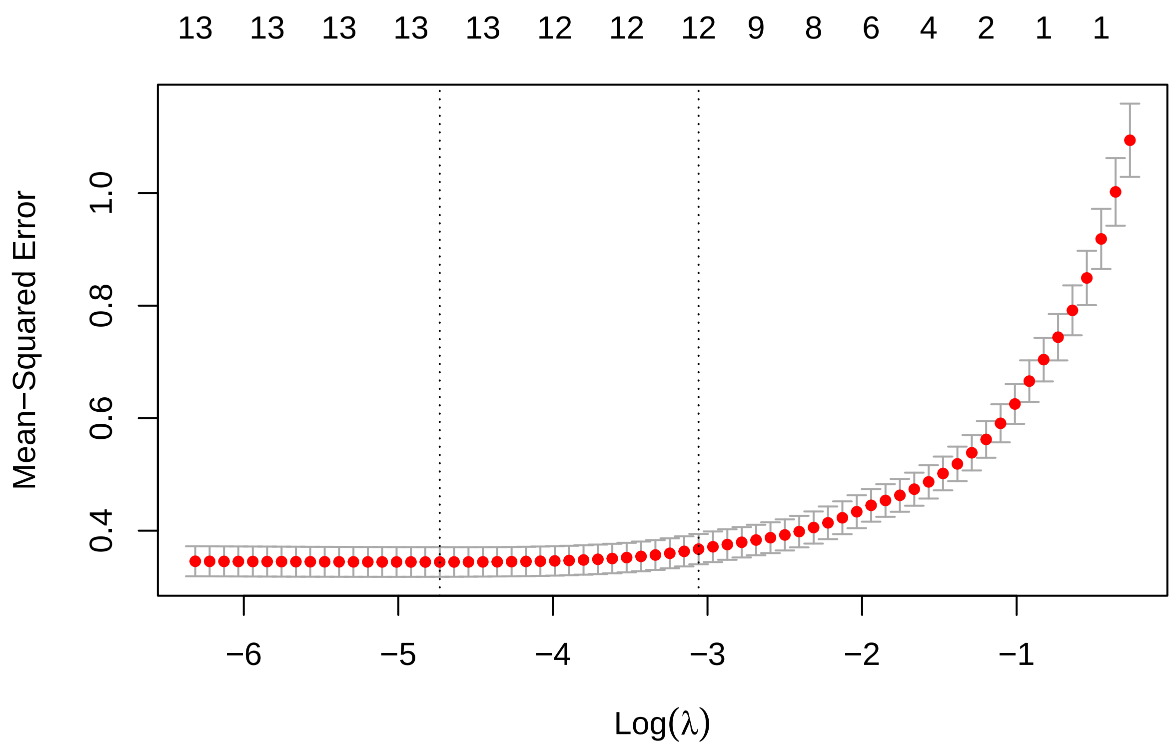
We can see how one of our attributes is changing rapidly with changing the lambda coefficient. The bigger the L1 is the less regularized is. The default parameter for alpha is 1.

Lets try the cross validation:

```
lasso_cv <- cv.glmnet(as.matrix(x_train),y_train)
print(lasso_cv)
```

```
##
## Call:  cv.glmnet(x = as.matrix(x_train), y = y_train)
##
## Measure: Mean-Squared Error
##
##      Lambda Index Measure      SE Nonzero
## min 0.00881    49  0.3443 0.02630      13
## 1se 0.04699    31  0.3672 0.02696      12
```

```
plot(lasso_cv)
```



On the left side we have our full model but as we go towards log(lambda) = 0 the more regularization we apply and the smaller model we have.

```
coef(lasso_cv,s="lambda.1se")
```

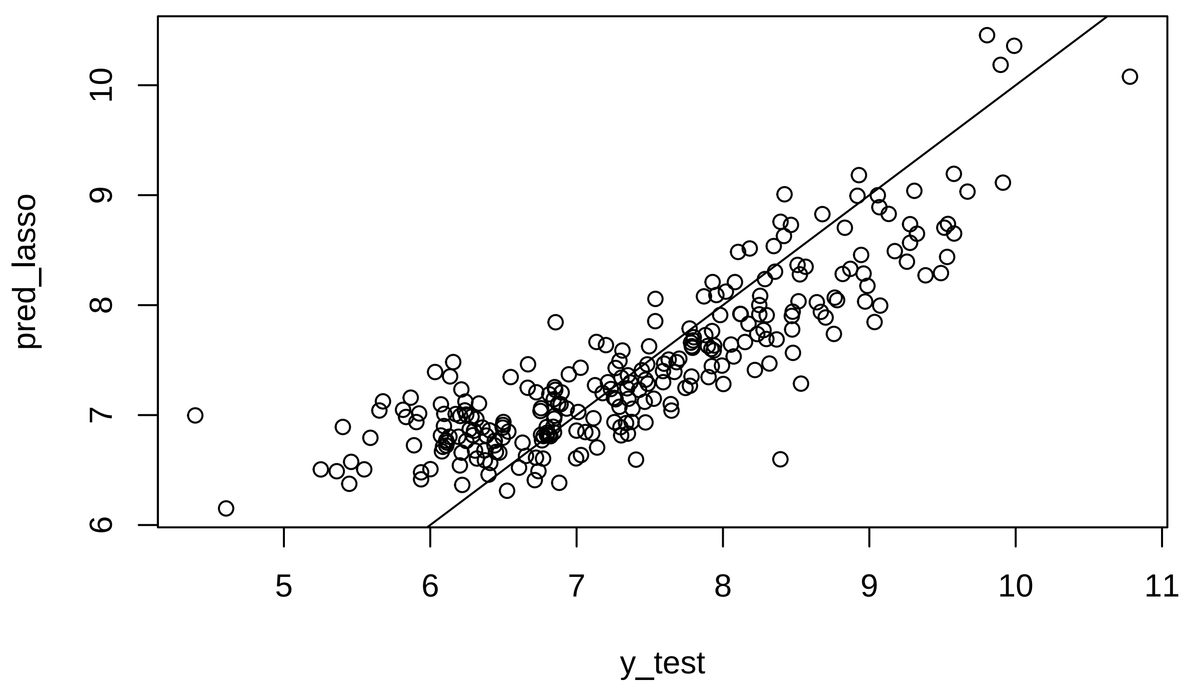
```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
```

```
## (Intercept)  5.382274e+00
## Private     -4.532462e-01
## Top10perc    .
## Top25perc    4.304019e-03
## F.Undergrad  1.091721e-04
## P.Undergrad  .
## Outstate     1.348555e-05
## Room.Board   8.349828e-05
## Books        2.957226e-05
## Personal     .
## PhD          7.165100e-03
## Terminal     1.831840e-03
## S.F.Ratio    1.042483e-02
## perc.alumni -1.186541e-03
## Expend       1.114200e-05
## Grad.Rate    5.012353e-03
```

We can see because of the regularization from Lasso, how some of our attributes are not used. We are obtaining the moder from the second vertical line.

```
pred_lasso <- predict(lasso_cv, newx=as.matrix(x_test),s="lambda.1se")

plot(y_test, pred_lasso)
abline(c(0,1))
```



```r
compute_rmse(y_test, pred_lasso)
```

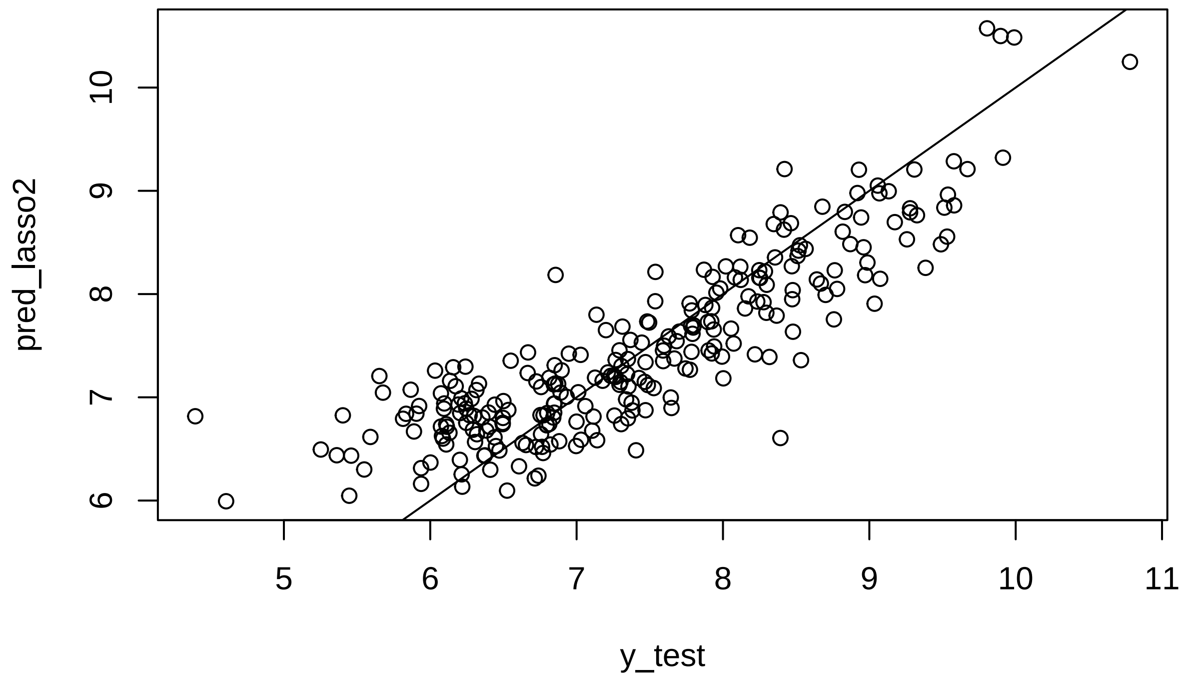
```
## [1] 0.6093105
```

```r
coef(lasso_cv,s="lambda.min")
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept)  4.777322e+00
## Private     -6.215008e-01
## Top10perc    .
## Top25perc    5.210804e-03
## F.Undergrad  1.046137e-04
## P.Undergrad  3.107481e-06
## Outstate     4.023807e-05
## Room.Board   8.186685e-05
## Books        2.466269e-04
## Personal     .
## PhD          5.809344e-03
## Terminal     1.926305e-03
## S.F.Ratio    2.960012e-02
## perc.alumni -7.977509e-03
## Expend       2.032567e-05
## Grad.Rate    7.898905e-03
```

```r
pred_lasso2 <- predict(lasso_cv, newx=as.matrix(x_test),s="lambda.min")

plot(y_test, pred_lasso2)
abline(c(0,1))
```




```
compute_rmse(y_test, pred_lasso2)
```

```
## [1] 0.5692547
```

In comparison from the previous exercise we have: - PCR: 0.5674299 - PLSR: 0.5672898 - Ridge Regression min + 1 std error: 0.5648764 - Lasso Regression min: 0.6061025