Regularized Regression I

Setup

```r
#library(learnr)
library(mlbench)
library(glmnet)
library(caret)

options(scipen=999)
```

Data

In this notebook, we use the Boston Housing data set. "This dataset contains information collected by the U.S Census Service concerning housing in the area of Boston Mass. It was obtained from the StatLib archive (http://lib.stat.cmu.edu/datasets/boston), and has been used extensively throughout the literature to benchmark algorithms."

Source: https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html

```r
data(BostonHousing2)
head(BostonHousing2)
```

```
##         town tract      lon     lat medv cmedv    crim zn indus chas   nox
## 1     Nahant  2011 -70.9550 42.2550 24.0  24.0 0.00632 18  2.31    0 0.538
## 2 Swampscott  2021 -70.9500 42.2875 21.6  21.6 0.02731  0  7.07    0 0.469
## 3 Swampscott  2022 -70.9360 42.2830 34.7  34.7 0.02729  0  7.07    0 0.469
## 4 Marblehead  2031 -70.9280 42.2930 33.4  33.4 0.03237  0  2.18    0 0.458
## 5 Marblehead  2032 -70.9220 42.2980 36.2  36.2 0.06905  0  2.18    0 0.458
## 6 Marblehead  2033 -70.9165 42.3040 28.7  28.7 0.02985  0  2.18    0 0.458
##      rm  age    dis rad tax ptratio      b lstat
## 1 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
```

```r
names(BostonHousing2)
```

```
##  [1] "town"    "tract"   "lon"     "lat"     "medv"    "cmedv"   "crim"
##  [8] "zn"      "indus"   "chas"    "nox"     "rm"      "age"     "dis"
## [15] "rad"     "tax"     "ptratio" "b"       "lstat"
```

Since we want to compare the performance of some regularized models at the end of the modeling process, we first split the data into a training and a test part. This can be done by random sampling with `sample`.

```
set.seed(7345)
train <- sample(1:nrow(BostonHousing2), 0.8*nrow(BostonHousing2))
boston_train <- BostonHousing2[train,]
boston_test <- BostonHousing2[-train,]
```

A quick look on our outcome variable for the next sections, which is the Median value of owner-occupied homes in $1000's.

```
summary(boston_train$medv)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    5.00   16.68   21.15   22.39   25.00   50.00
```

```
summary(boston_test$medv)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    8.10   18.27   21.30   23.10   24.60   50.00
```

Regularized regression

Now we can prepare our training data for the regularized regression models. The `glmnet` package needs models to be fitted on an X matrix and an y vector, which we need to generate first. - function model.metrix is to get the features and no outcome.

```
X <- model.matrix(medv ~ . - town - tract - cmedv,
                  boston_train)[,-1]
y <- boston_train$medv
```

Ridge regression

To estimate a sequence of regularized models we pass our X and y objects to the `glmnet` function. Setting alpha to zero equals to fitting ridge regression models. By default, `glmnet` figures out an appropriate series of lambda values.

```
m1 <- glmnet(X, y, alpha = 0)
summary(m1)
```

```
##           Length Class     Mode
## a0         100   -none-    numeric
## beta      1500   dgCMatrix S4
## df         100   -none-    numeric
## dim          2   -none-    numeric
## lambda     100   -none-    numeric
## dev.ratio  100   -none-    numeric
## nulldev      1   -none-    numeric
## npasses      1   -none-    numeric
## jerr         1   -none-    numeric
```

```
## offset       1    -none-    logical
## call         4    -none-    call
## nobs         1    -none-    numeric
```

Let's see how we can access the results from glmnet…

```r
m1$lambda[1]
```

```
## [1] 6786.868
```

```r
m1$lambda[100]
```

```
## [1] 0.6786868
```

```r
m1$beta[,1]
```

```
##                                              lon
## -0.00000000000000000000000000000000004020939208
##                                              lat
##  0.00000000000000000000000000000000000540353909
##                                             crim
## -0.00000000000000000000000000000000000039664290
##                                               zn
##  0.00000000000000000000000000000000000012771298
##                                            indus
## -0.00000000000000000000000000000000000065028971
##                                            chas1
##  0.00000000000000000000000000000000000569227069
##                                              nox
## -0.00000000000000000000000000000000003460459480
##                                               rm
##  0.00000000000000000000000000000000000975845461
##                                              age
## -0.00000000000000000000000000000000000011391230
##                                              dis
##  0.00000000000000000000000000000000000104378926
##                                              rad
## -0.00000000000000000000000000000000000043945467
##                                              tax
## -0.00000000000000000000000000000000000002692880
##                                          ptratio
## -0.00000000000000000000000000000000000228772285
##                                                b
##  0.00000000000000000000000000000000000003387063
##                                            lstat
## -0.00000000000000000000000000000000000096793963
```

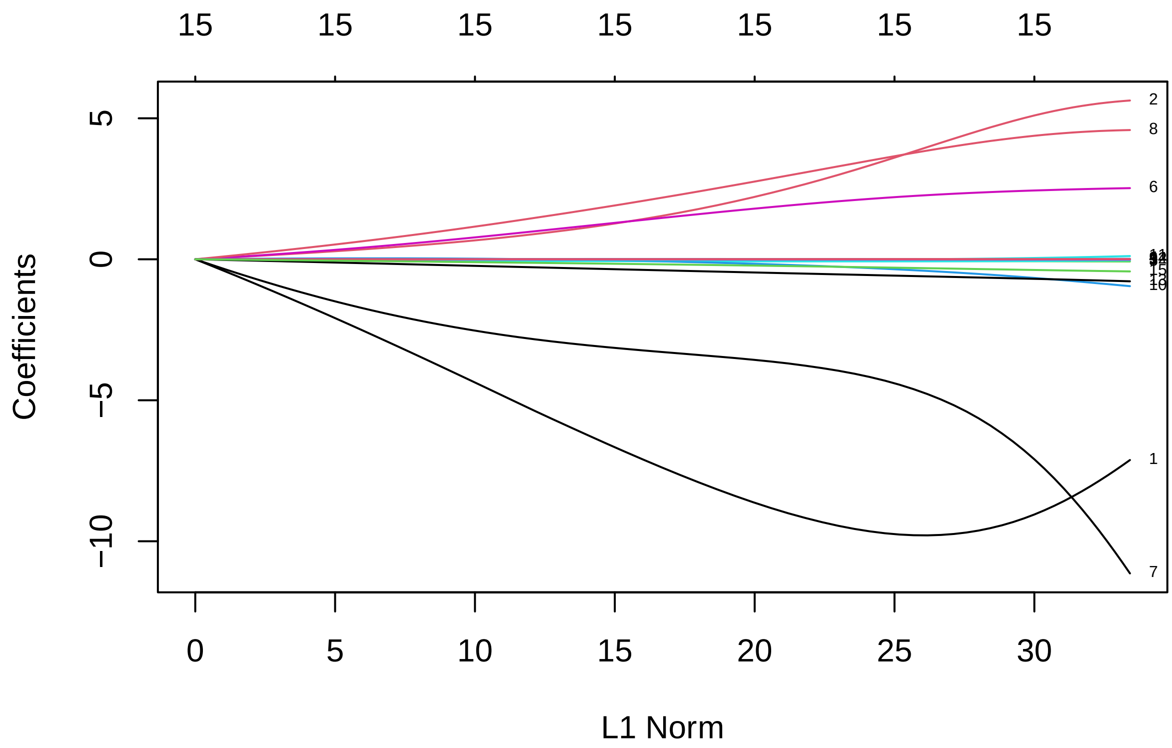
```r
m1$beta[,ncol(m1$beta)]
```

```
##           lon           lat          crim            zn         indus
## -7.1192300892  5.6306278624 -0.0780558629  0.0175753734 -0.0341012420
```

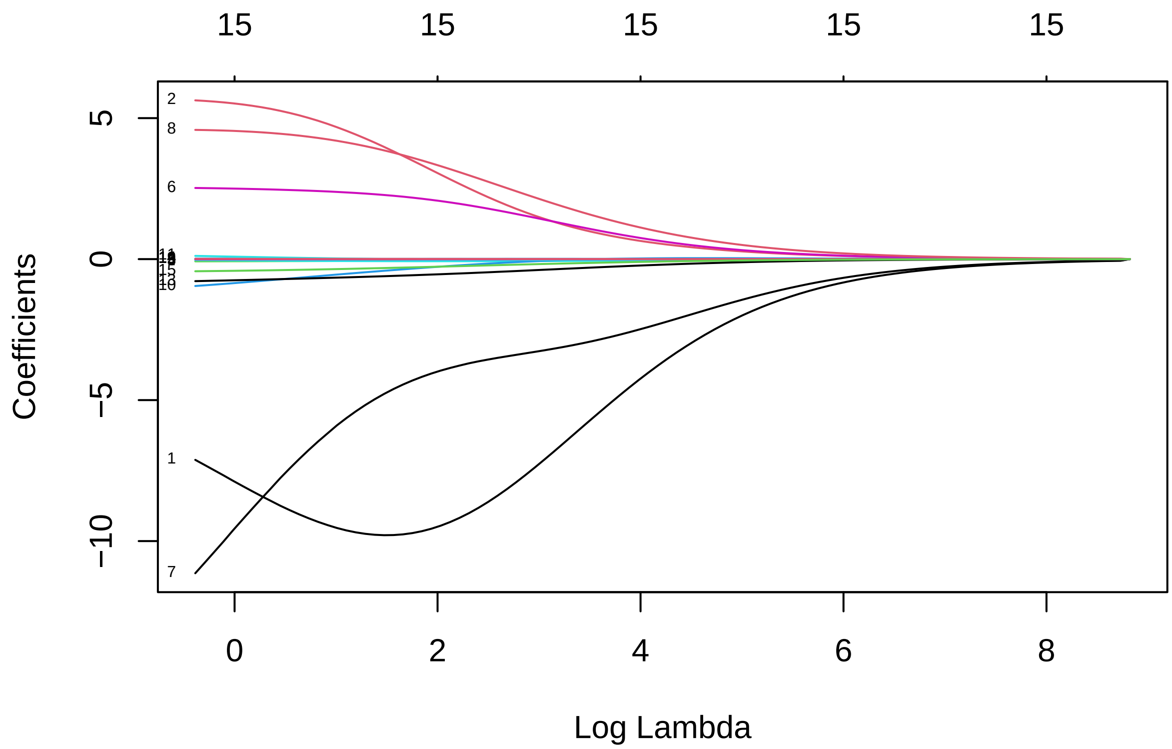
```
##           chas1             nox              rm             age             dis
##    2.5229417158 -11.1398273615    4.5829833396    0.0002521612   -0.9527754982
##             rad             tax         ptratio               b           lstat
##    0.1145021276   -0.0064694984   -0.7805409765    0.0094249700   -0.4298141064
```

A nice feature of `glmnet` is that we can easily plot the coefficient paths by simply calling `plot` in connection with our results object.

```
plot(m1, label=T)
```



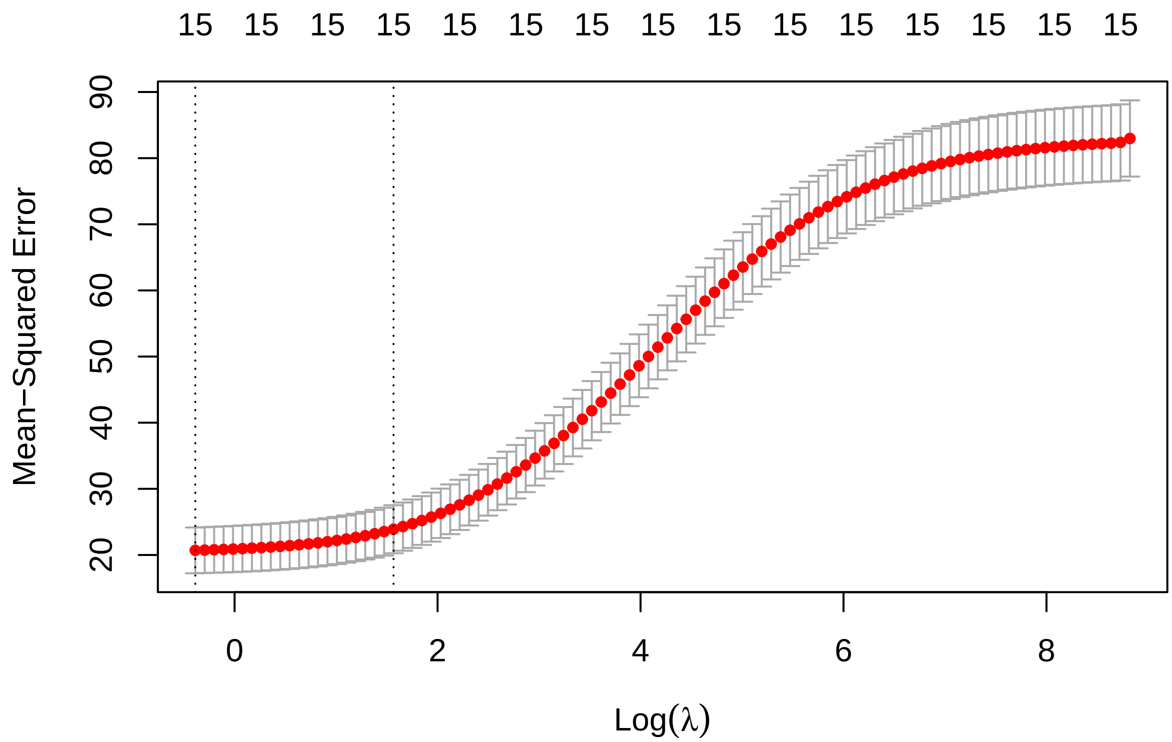

```
plot(m1, label=T, xvar = "lambda")
```



However, at this point we do not know which lambda leads to the best model. Defining "best" in terms of prediction performance for new data, Cross-Validation can be used for this task.

- For different lambdas we have a MSE. Very low values of log(lambda), close to 0 is OLS might be overfitting, close to infinity can be underfitting. We should expect to see a U shape. In this example, we do not have many features and those that are there have a strong relationship which in this case OLS is maybe not overfitting.
- Typically we prefer the simplest model within 1 sd of the lowest.

```
m1_cv <- cv.glmnet(X, y, alpha = 0)
plot(m1_cv)
```



On this basis, we can now have a look at the models that perform best in terms of the smallest CV error and with respect to the 1-SE rule. We also store the value of lambda that corresponds to the smallest CV error for later usage.

```r
coef(m1_cv, s = "lambda.min")
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                           s1
## (Intercept) -722.0381670534
## lon           -7.1192300892
## lat            5.6306278624
## crim          -0.0780558629
## zn             0.0175753734
## indus         -0.0341012420
## chas1          2.5229417158
## nox          -11.1398273615
## rm             4.5829833396
## age            0.0002521612
## dis           -0.9527754982
## rad            0.1145021276
## tax           -0.0064694984
## ptratio       -0.7805409765
## b              0.0094249700
## lstat         -0.4298141064
```

```r
coef(m1_cv, s = "lambda.1se")
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept) -839.5290652785
## lon           -9.7857920873
## lat            3.8222147957
## crim          -0.0630520846
## zn             0.0090393305
## indus         -0.0724589794
## chas1          2.2461586627
## nox           -4.6063552745
## rm             3.7734336376
## age           -0.0031405747
## dis           -0.3870609097
## rad           -0.0008737155
## tax           -0.0035717375
## ptratio       -0.5946401879
## b              0.0077017025
## lstat         -0.3083477088
```

```r
bestlam1 <- m1_cv$lambda.min
bestlam1
```

```
## [1] 0.6786868
```

Lasso

To estimate a Lasso sequence, we simply call `glmnet` again and set alpha to one.

```r
m2 <- glmnet(X, y, alpha = 1)
```

Here we want to display the first, last and one in-between model of our model series. We see that coefficients are eventually shrunken exactly to zero as the penalty on model complexity increases.

```r
m2$lambda[1]
```

```
## [1] 6.786868
```

```r
m2$lambda[(ncol(m2$beta)/2)]
```

```
## [1] 0.21713
```

```r
m2$lambda[ncol(m2$beta)]
```

```
## [1] 0.005767164
```

```r
m2$beta[,1]
```

```
##      lon      lat     crim       zn    indus    chas1      nox       rm      age      dis 
##        0        0        0        0        0        0        0        0        0        0 
##      rad      tax  ptratio        b    lstat 
##        0        0        0        0        0
```

```r
m2$beta[,(ncol(m2$beta)/2)]
```

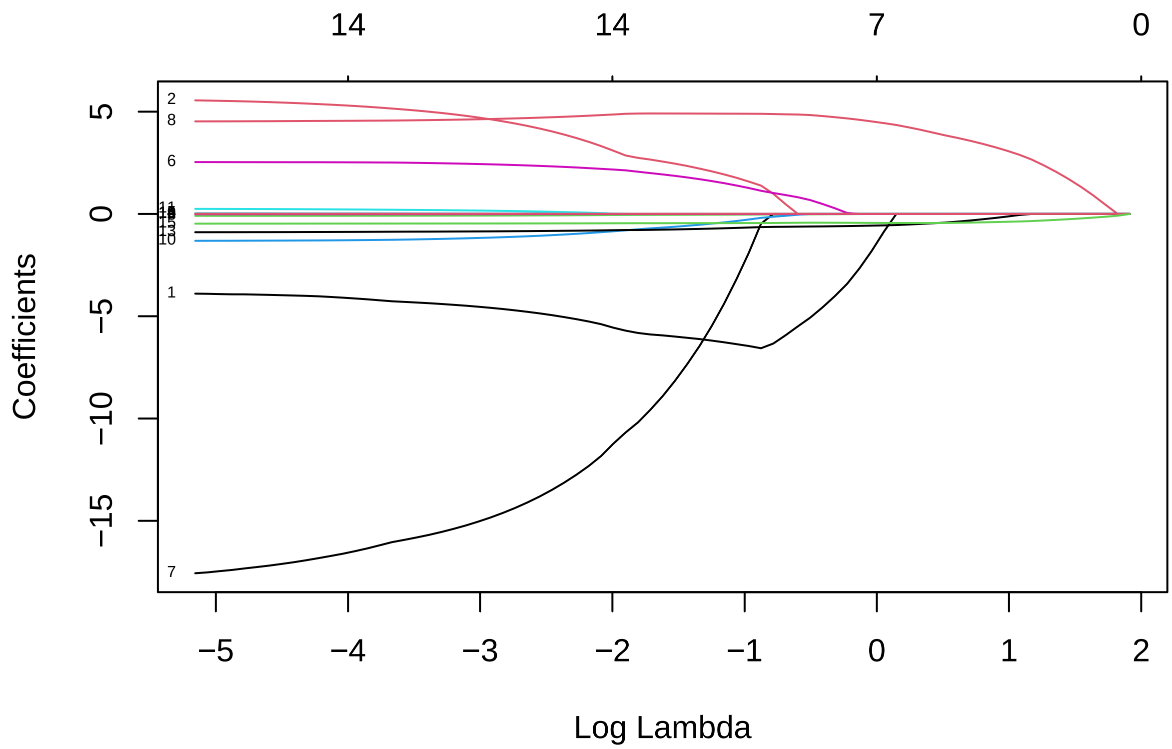
```
##          lon          lat         crim           zn        indus        chas1 
## -5.992391832  2.458763718 -0.042629046  0.000000000  0.000000000  1.862330986 
##          nox           rm          age          dis          rad          tax 
## -8.160857589  4.909346699  0.000000000 -0.625030144  0.000000000 -0.002154534 
##      ptratio            b        lstat 
## -0.756126834  0.008461913 -0.450541237
```

```r
m2$beta[,ncol(m2$beta)]
```

```
##           lon           lat          crim            zn         indus 
##  -3.895371176   5.553192899  -0.092616460   0.029792679   0.030094273 
##         chas1           nox            rm           age           dis 
##   2.535798274 -17.562578334   4.524820661   0.001411861  -1.313856328 
##           rad           tax       ptratio             b         lstat 
##   0.247956800  -0.012090046  -0.892933311   0.009390218  -0.473572510
```

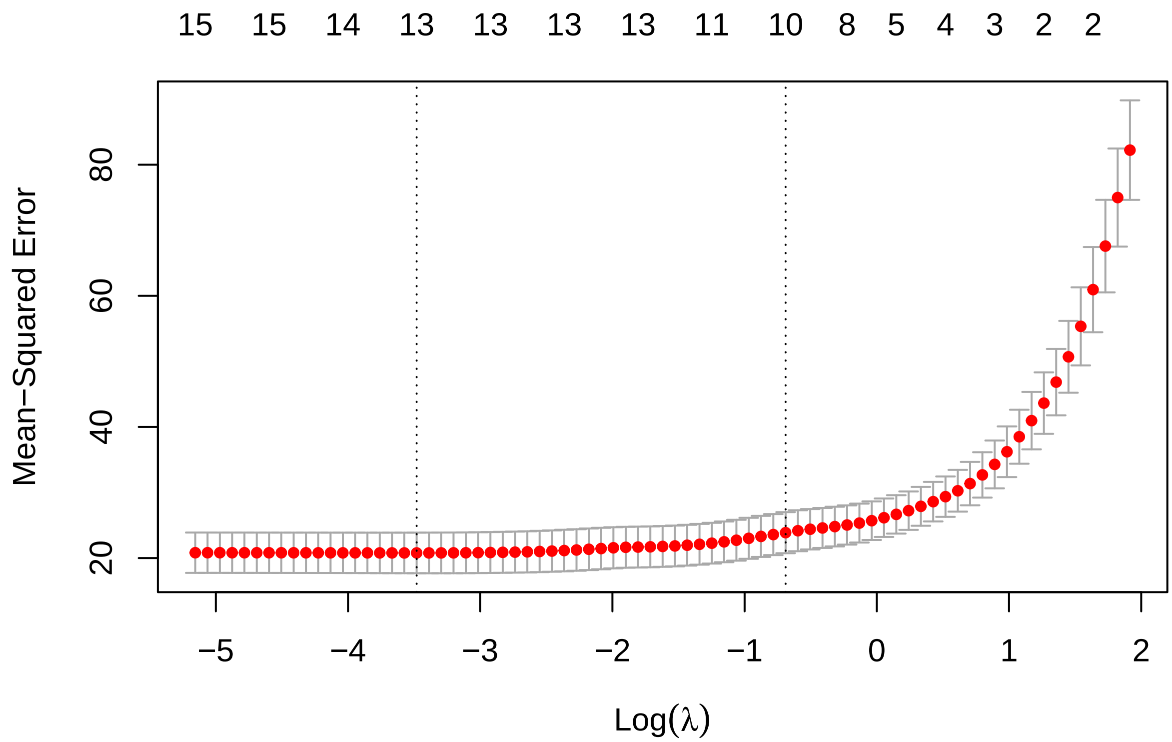
This also becomes clear when plotting the coefficient paths.

```r
plot(m2, label=T, xvar = "lambda")
```



When using Cross-Validation with Lasso, we see that a full model with all features may not lead to the best model in terms of prediction performance.

```
m2_cv <- cv.glmnet(X, y, alpha = 1)
plot(m2_cv)
```



Again, we may have a look at the model with the smallest CV error and store the corresponding lambda.

```
coef(m2_cv, s = "lambda.min")
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                          s1
## (Intercept) -493.631361809
## lon           -4.329729178
## lat            5.055107907
## crim          -0.085105419
## zn             0.024276367
## indus          .
## chas1          2.498279504
## nox          -15.814095103
## rm             4.579454886
## age            .
## dis           -1.242942353
## rad            0.195897499
## tax           -0.009592031
## ptratio       -0.864970242
## b              0.009246831
## lstat         -0.467602520
```

```
bestlam2 <- m2_cv$lambda.min
```

Prediction in test set

Finally, we investigate the performance of our models in the test set. For this task, we construct an X matrix from the test set.

```
Xt <- model.matrix(medv ~ . - town - tract - cmedv,
                   boston_test)[,-1]
```

This matrix can be used in the `predict` function, along with the respective model that should be used for prediction. We try out our best ridge, lasso and elastic net model. One can also add a "null model" with a huge penalty for comparison purposes.

```
p_ridge <- predict(m1, s = bestlam1, newx = Xt)
p_lasso <- predict(m2, s = bestlam2, newx = Xt)

p_null <- predict(m2, s = 1e10, newx = Xt) # lamba approximates infinity
```

As a last step, let's look at the test MSE of our models.

```
mean((p_null - boston_test$medv)^2)
```

```
## [1] 93.03412
```

```
mean((p_ridge - boston_test$medv)^2)
```

```
## [1] 37.62915
```

```
mean((p_lasso - boston_test$medv)^2)
```

```
## [1] 37.0057
```

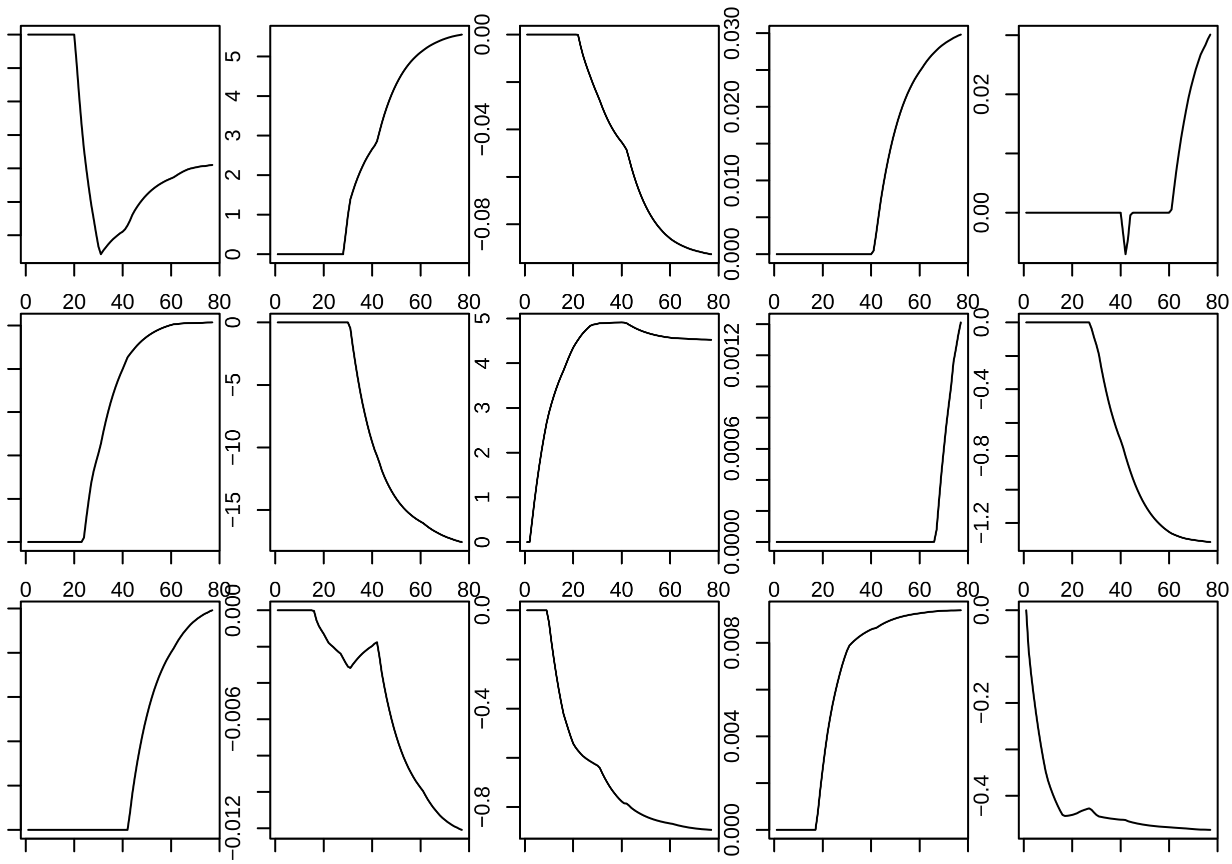
More plots

We can also plot the coefficient paths separately to get a more detailed picture. Here we use the Lasso results and extract the coefficients using `as.matrix`.

```
coefs <- as.matrix(m2$beta)
```

Now we can plot the coefficient path over the range of lambdas for each variable by looping over the rows of the `coefs` object.

```
par(mfrow=c(3,5), mar=c(1,1,1,1))
for (i in 1:nrow(coefs))
  plot(coefs[i,], type = "l")
```



Feature selection by filter

Another approach for selecting features is (simple) feature screening via filtering. In this context it is important to be cautious when estimating predicting performance and correctly combine filtering and CV. The `caret` can be used to take care of that. First, we have to set up our inner (trainControl) and outer (sbfControl) evaluation techniques.

```
sbf_ctrl <- sbfControl(functions = caretSBF,
                       method = "cv",
                       number = 10)

ctrl <- trainControl(method = "none")
```

Now we can run CV with feature selection by filter via `sbf`.

```
m3 <- sbf(medv ~ . - town - tract - cmedv,
          data = boston_train,
          method = 'lm',
          sbfControl = sbf_ctrl,
          trControl = ctrl)
```

The corresponding results object gives us an estimate of prediction performance and information on the selected features.

```r
m3
```

```
##
## Selection By Filter
##
## Outer resampling method: Cross-Validated (10 fold)
##
## Resampling performance:
##
##   RMSE Rsquared   MAE RMSESD RsquaredSD  MAESD
##  4.436   0.7739 3.167  1.088    0.06507 0.5724
##
## Using the training set, 14 variables were selected:
##    lon, crim, zn, indus, chas1...
##
## During resampling, the top 5 selected variables (out of a possible 14):
##    age (100%), b (100%), chas1 (100%), crim (100%), dis (100%)
##
## On average, 14 variables were selected (min = 14, max = 14)
```

We can also use this object to apply the final model (fitted on the full training set) to the test set and calculate the test MSE.

```r
p_filter <- predict(m3, newdata = boston_test)
mean((p_filter - boston_test$medv)^2)
```

```
## [1] 36.76964
```

References

- https://web.stanford.edu/~hastie/glmnet/glmnet_alpha.html