Statistical Learning assignment 1

H.C. (Bram) Otten (s2330326)

24-11-2019

The purpose of this homework is to apply linear regression on a real data set. We will consider three ways of improving prediction accuracy: feature selection, ridge regression and the lasso. We will split the data set into two parts (training and test set), then fit the models on the training set (including choice of the best parameter using cross-validation) and test their performance on the separate test set.

Introductory analyses

1

Some properties of the data (link) are:

- Title: Boston Housing Data
- Sources:
  (a) Origin: This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.
  (b) Creator: Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978.
  (c) Date: July 7, 1993
- Relevant Information: concerns housing values in suburbs of Boston.
- Number of Instances: 506.
- Number of Attributes: 13 continuous attributes (including "class" attribute "MEDV"), 1 binary-valued attribute.
- Attribute Information:
  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 per $10,000
  11. PTRATIO pupil-teacher ratio by town
  12. B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
  13. LSTAT % lower status of the population
  14. MEDV Median value of owner-occupied homes in $1000's

2

```r
data <- read.table("housing-p.data", header = TRUE)
dim(data)
```

```
## [1] 506  14
```

```r
head(data)
```

```
##      CRIM ZN INDUS CHAS   NOX    RM  AGE    DIS RAD TAX PTRATIO      B
## 1 0.01301 35  1.52    0 0.442 7.241 49.3 7.0379   1 284    15.5 394.74
## 2 9.39063  0 18.10    0 0.740 5.627 93.9 1.8172  24 666    20.2 396.90
## 3 0.54011 20  3.97    0 0.647 7.203 81.8 2.1121   5 264    13.0 392.80
## 4 0.20746  0 27.74    0 0.609 5.093 98.0 1.8226   4 711    20.1 318.43
## 5 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90
## 6 1.20742  0 19.58    0 0.605 5.875 94.6 2.4259   5 403    14.7 292.29
##   LSTAT MEDV
## 1  5.49 32.7
## 2 22.88 12.8
## 3  9.59 33.8
## 4 29.68  8.1
## 5  9.14 21.6
## 6 14.43 17.4
```

```r
str(data)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ CRIM   : num  0.013 9.3906 0.5401 0.2075 0.0273 ...
##  $ ZN     : num  35 0 20 0 0 0 0 0 0 0 ...
##  $ INDUS  : num  1.52 18.1 3.97 27.74 7.07 ...
##  $ CHAS   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ NOX    : num  0.442 0.74 0.647 0.609 0.469 0.605 0.74 0.55 0.713 0.585 ...
##  $ RM     : num  7.24 5.63 7.2 5.09 6.42 ...
##  $ AGE    : num  49.3 93.9 81.8 98 78.9 94.6 92.4 85.1 84.4 72.9 ...
##  $ DIS    : num  7.04 1.82 2.11 1.82 4.97 ...
##  $ RAD    : int  1 24 5 4 2 5 24 5 24 6 ...
##  $ TAX    : int  284 666 264 711 242 403 666 276 666 391 ...
##  $ PTRATIO: num  15.5 20.2 13 20.1 17.8 14.7 20.2 16.4 20.2 19.2 ...
##  $ B      : num  395 397 393 318 397 ...
##  $ LSTAT  : num  5.49 22.88 9.59 29.68 9.14 ...
##  $ MEDV   : num  32.7 12.8 33.8 8.1 21.6 17.4 10.5 28.7 20 19.7 ...
```

```r
# data$CHAS <- as.factor(data$CHAS) # the models should already do this
train <- data[1:350,]
test <- data[351:nrow(data),]
all.equal(data, rbind(train, test))
```

```
## [1] TRUE
```

Note that the data has not been shuffled before splitting into a training and test set.

3

We begin by fitting a least squares model on the training set, regarding MEDV ("median value of owner-occupied homes in $1000's") as dependent / output variable.

```r
lmf <- lm(MEDV ~ ., train)
```

4

```r
summary(lmf)
```

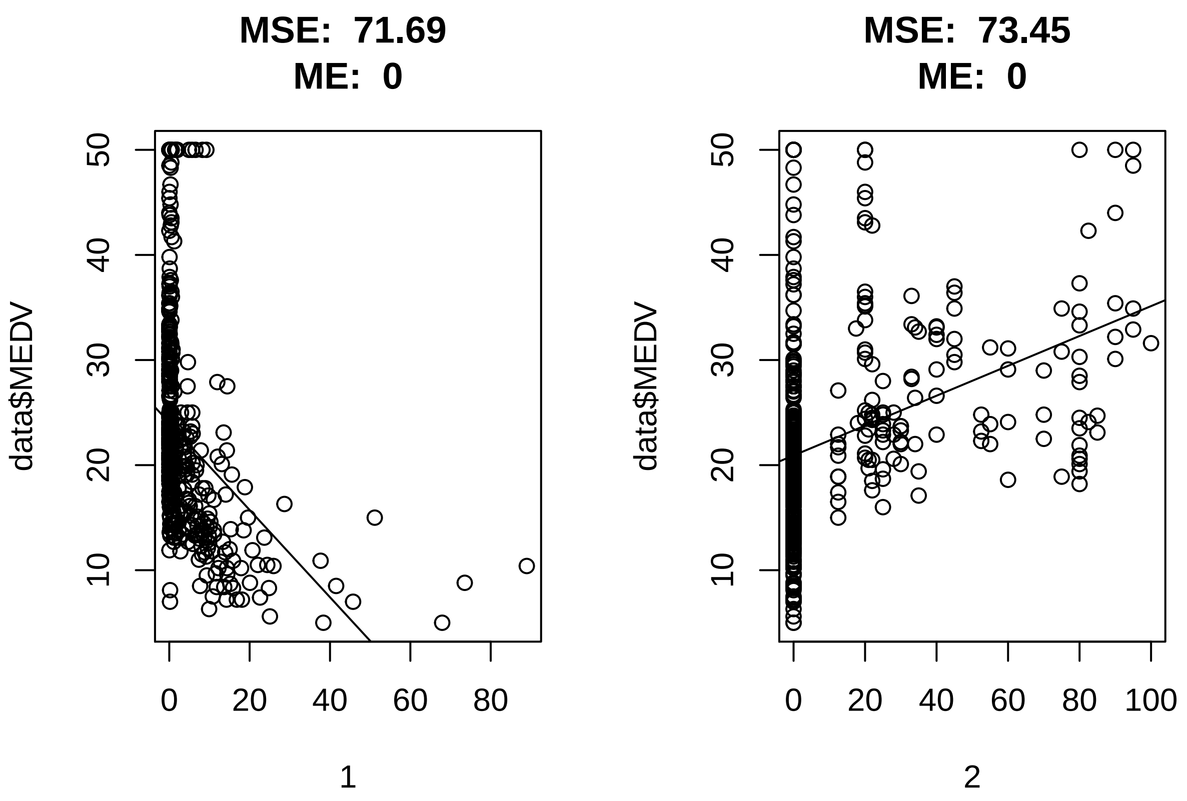
```
##
## Call:
```

```
## lm(formula = MEDV ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.1905  -2.6593  -0.7882   1.7322  25.8305
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  39.400328   6.174425   6.381 5.83e-10 ***
## CRIM         -0.113524   0.041778  -2.717 0.006923 **
## ZN            0.063716   0.016383   3.889 0.000121 ***
## INDUS         0.024835   0.073737   0.337 0.736475
## CHAS          1.486094   0.997993   1.489 0.137404
## NOX         -17.031976   4.700724  -3.623 0.000336 ***
## RM            3.358048   0.522390   6.428 4.43e-10 ***
## AGE          -0.005373   0.015617  -0.344 0.731013
## DIS          -1.642477   0.238963  -6.873 3.06e-11 ***
## RAD           0.299260   0.075406   3.969 8.84e-05 ***
## TAX          -0.013785   0.004319  -3.192 0.001548 **
## PTRATIO      -0.842203   0.153720  -5.479 8.41e-08 ***
## B             0.006313   0.003177   1.987 0.047704 *
## LSTAT        -0.521567   0.060674  -8.596 3.17e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.675 on 336 degrees of freedom
## Multiple R-squared:  0.7448, Adjusted R-squared:  0.7349
## F-statistic: 75.43 on 13 and 336 DF,  p-value: < 2.2e-16
```

The summary above shows coefficients and their estimated standards errors as found by a least squares model, as well as statistical significance of 'H0: this coefficient = 0' (in column `Pr(>|t|)`). Note that we use all variables but MEDV as regressors / features / independent variables / explanatory variables. (These are the first 13 columns of `data`.)

There are a few assumptions we have to meet in order to justify using a linear model for interpreting the relationship between the variables in the data. Among these are (1) a linear relationship between independent and dependent variables (see following plots, MSE gives some indication of fit); and (2) that the mean of the residuals is close to 0 (ME in plot titles)).

```r
par(mfrow = c(1,2))
for (i in 1:2) { # change end to 13 for all (i.e. too many for a pdf) plots
   lmf_i <- lm(paste("MEDV ~ ", colnames(data)[i]), data)
   plot(data[, i], data$MEDV, xlab = i,
        main = paste("MSE: ", round(mean(lmf_i$residuals ^ 2), 2),
                     "\nME: ", round(mean(lmf_i$residuals), 2)))
   abline(lmf_i)
}
```



```r
par(mfrow = c(1,1))
```

There are a few more assumptions, as listed on this website. [It does not seem to be part of the assignment to check all of these manually and try to correct them.] Low multicollinearity (i.e. small correlations between 'independent' variables, resulting in low variance in finding the best coefficients) is of special interest because the models proposed later aim to deal with it.

```r
# install.packages('usdm')
library(usdm)
```

```
## Loading required package: sp

## Loading required package: raster
```

```r
vif(data[, -14])
```

```
##    Variables      VIF
## 1       CRIM 1.792192
## 2         ZN 2.298758
## 3      INDUS 3.991596
## 4       CHAS 1.073995
## 5        NOX 4.393720
## 6         RM 1.933744
## 7        AGE 3.100826
## 8        DIS 3.955945
## 9        RAD 7.484496
## 10       TAX 9.008554
## 11   PTRATIO 1.799084
## 12          B 1.348521
## 13      LSTAT 2.941491
```

A common rule of thumb for diagnosing a set of independent variables to be highly multicollinear is to check if the variance inflation factor $(1/(1 - R_i^2))$ of any of the variables is greater than some threshold, e.g. 5 or 10 (here is a paper with some critique on using a threshold and also links to the proposers of these thresholds). There seems to be some multicollinearity among our independent variables.

In any case, here follows a quick indication of our model's accuracy on the unseen test set.

```
lmf_predictions <- predict(lmf, test)
lmf_mse <- mean((test[, 14] - lmf_predictions) ^ 2)
cat("Train MSE:", mean(lmf$residuals ^ 2),
    "\nTest MSE:", lmf_mse, "\n")
```

```
## Train MSE: 20.97704
## Test MSE: 25.1503
```

```
total_ss <- sum((test[, 14] - mean(test[, 14])) ^ 2)
lmf_reg_ss <- lmf_mse * length(test[, 14])
(lmf_R2 <- 1 - lmf_reg_ss / total_ss)
```

```
## [1] 0.7185375
```

Cross-validation and regularization

5

In 5-fold cross-validation, we fit our model on four out of five parts of the training data, and evaluate on the fifth part. Each of the five parts is considered a test fold once, and the average error of these five iterations is taken to be the cv error. Compared to fitting on the whole training set, this results in only a bit more bias for each model, but a better estimate of prediction error (than `mean(lmf$residuals ^ 2` above).

```
set.seed(1)
s_train <- train[sample(nrow(train)),]
K <- 5
folds <- cut(seq(1, nrow(s_train)), breaks = K, labels = FALSE)
cv_mse <- numeric(K)
for (i in 1:K) {
   test_indices <- which(folds == i, arr.ind = TRUE)
   test_fold <- s_train[test_indices,]
   train_folds <- s_train[-test_indices,]
   lmf_i <- lm(MEDV ~ ., train_folds)
   cv_mse[i] <- mean((test_fold[, 14] - predict(lmf_i, test_fold)) ^ 2)
}
lmf_5cv_mse <- mean(cv_mse)
cat("5-fold cross-validation MSE:", lmf_5cv_mse, "\n")
```

```
## 5-fold cross-validation MSE: 23.17358
```

6

```
# install.packages("leaps")
library(leaps)

to_lm_input <- function(regressors) {
   # Like everything else, this is not meant to be especially robust.
   if ("(Intercept)" %in% regressors) {
      regressors <- regressors[-which(regressors == "(Intercept)")]
   } else {
      regressors <- c(regressors, "-1")
   }
   regressorsp <- paste(regressors, collapse = " + ", sep = "")
   return(paste("MEDV ~ ", regressorsp, sep = " "))
}
```

Here, instead of fitting all regressors per test fold, we fit only the *best* regressors (correcting for the number of degrees of freedom with something like the AIC). These best regressors have to be found seperately for each test fold to avoid evaluating over data already used in determining which of the (or how many) regressors are best.

```
subset_5cv_mse <- matrix(nrow = K, ncol = 13)
for (k in 1:K) {
   test_indices <- which(folds == k, arr.ind = TRUE)
   test_fold <- s_train[test_indices,]
   train_folds <- s_train[-test_indices, ]
   rsti <- regsubsets(x = train_folds[, -14], y = train_folds[, 14],
                      nvmax = 13, method = "exhaustive")
   for (n in 1:13) {
      regressors <- names(which(summary(rsti)$which[n,]))
      lm_kn <- lm(to_lm_input(regressors), train_folds)
      subset_5cv_mse[k, n] <-
         mean((test_fold[, 14] - predict(lm_kn, test_fold)) ^ 2)
   }
}
(subset_5cv_mse_per_n <- apply(subset_5cv_mse, 2, mean))
```

```
##  [1] 36.09388 29.78185 26.85972 26.49689 24.66702 24.82970 24.62877
##  [8] 25.05117 23.41937 23.13482 22.92910 23.17054 23.17358
```

```
(best_5cv_n <- which.min(subset_5cv_mse_per_n))
```

```
## [1] 11
```

Now we find the `best_5cv_n` regressors that are best on the whole training set, and use those to fit a new model. Note that no cross-validation is involved here – we will compare full models on their actual test set prediction error later on in this report.

```
train_subset <- regsubsets(x = train[, -14], y = train[, 14],
                           nvmax = best_5cv_n, method = "exhaustive")
best_5cv_n_regressors <- names(which(summary(train_subset)$which[best_5cv_n,]))
lm_subset <- lm(to_lm_input(best_5cv_n_regressors), train)
subset_predictions <- predict(lm_subset, test)
subset_mse <- mean((test[, 14] - subset_predictions) ^ 2)
cat("Test MSE:", subset_mse, "\n")
```

```
## Test MSE: 25.11224
```

```
best_5cv_n_regressors
```

```
##  [1] "(Intercept)" "CRIM"        "ZN"          "CHAS"        "NOX"
##  [6] "RM"          "DIS"         "RAD"         "TAX"         "PTRATIO"
## [11] "B"           "LSTAT"
```

We see that AGE and INDUS are been left out.

7

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following objects are masked from 'package:raster':
##
##     area, select
```

Shrinkage methods like ridge regression are a continuous version of subset selection – they allow us to penalize coefficient magnitude. The best amount of penalty can be found using cross validation error (or its GCV approximation) again; this parameter is then used to fit a model on all training data. Rescaling (to prevent regressors that have only a small Δ(MEDV) per Δ(regressor) from being favoured) should happen automatically.

```
select(lm.ridge(MEDV ~ ., train, lambda = seq(0, 10, .001)))
```

```
## modified HKB estimator is 4.363396
## modified L-W estimator is 3.926067
## smallest value of GCV  at 3.96
```

```
select(lm.ridge(MEDV ~ ., train, lambda = seq(3.5, 4.5, 0.0001)))
```

```
## modified HKB estimator is 4.363396
## modified L-W estimator is 3.926067
## smallest value of GCV  at 3.96
```

```
ridge_gcv_lambda <- 3.96

ridge <- lm.ridge(MEDV ~ ., train, lambda = ridge_gcv_lambda)
ridge_predictions <- as.matrix(cbind(const = 1, test[-14])) %*% coef(ridge)
ridge_mse <- mean((test[, 14] - ridge_predictions) ^ 2)
cat("Test MSE:", ridge_mse, "\n")
```

```
## Test MSE: 25.11726
```

8

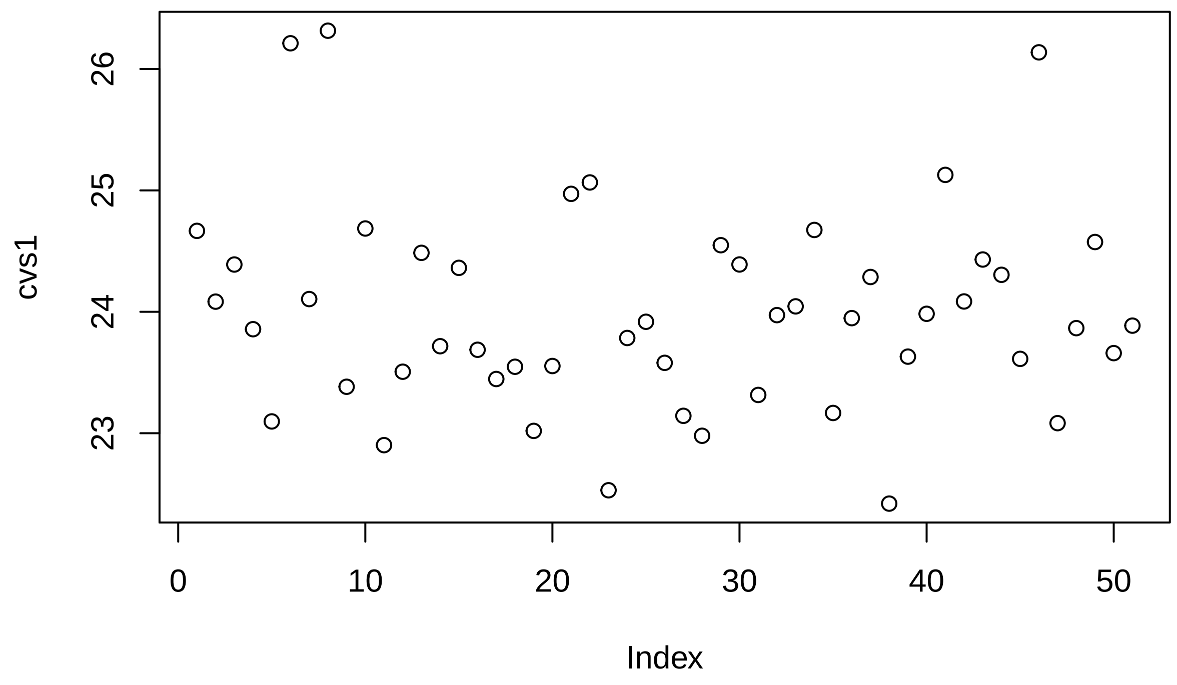
```
# install.packages("lars")
library(lars)
```

```
## Loaded lars 1.2
```

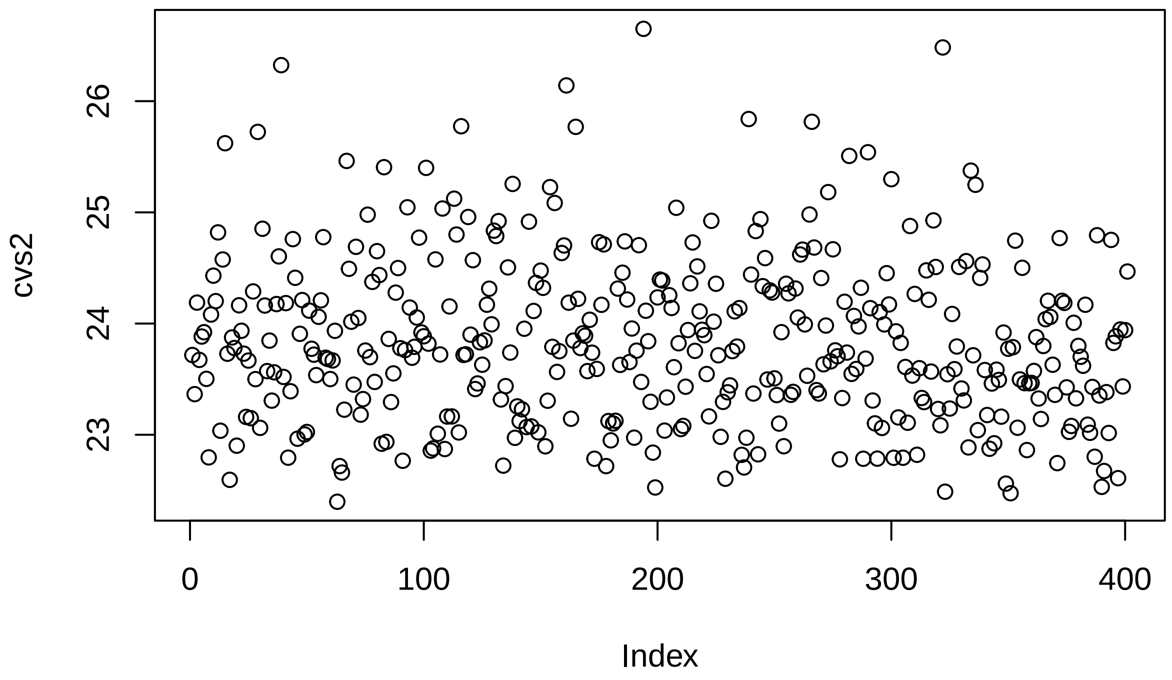
Lasso regression is another shrinkage method. The difference between it and ridge regression is that, in coming up with the to-be-minimized error, the shrinkage parameter is multiplied with the norm of the coefficient instead of with the square.

```
set.seed(1)
X = as.matrix(train[,-14])
y = as.matrix(train[,14])
lambda_range1 <- seq(0, 10, .2)
cvs1 <- numeric(length(lambda_range1))
for (i in 1:length(lambda_range1)) {
   cvs1[i] <- min(cv.lars(X, y, K = 5, type = "lasso",
                          s = lambda_range1[i], plot.it = FALSE)$cv)
}
```

```r
plot(cvs1)
```



```r
# Try good region again:
lambda_range2 <- seq(20 * .2, 40 * .2, .01)
cvs2 <- numeric(length(lambda_range2))
for (i in 1:length(lambda_range2)) {
   cvs2[i] <- min(cv.lars(X, y, K = 5, type = "lasso",
                          s = lambda_range2[i], plot.it = FALSE)$cv)
}
plot(cvs2)
```



```r
cbind(min(cvs1), min(cvs2))
```

```
##          [,1]     [,2]
## [1,] 22.42028 22.39771
```

```r
(lasso_5cv_lambda <- lambda_range2[which.min(cvs2)])
```

```
## [1] 4.62
```

```r
lasso <- lars(X, y, type = "lasso")
lasso_predictions <- predict.lars(lasso, as.matrix(test[,-14]),
                                 mode = "lambda", s = lasso_5cv_lambda)$fit

lasso_mse <- mean((test[, 14] - lasso_predictions) ^ 2)
cat("Test MSE:", lasso_mse, "\n")
```

```
## Test MSE: 26.94795
```

This test MSE seems high, so let us try another library as well. It is quicker and finds a parameter that ultimately works better on the test set but does not result in a lower 5-fold cv error on the train set, and will thus not be used.

```r
set.seed(1)
library(glmnet)
```

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

```
## Loaded glmnet 3.0-1
```

```r
lasso2 <- cv.glmnet(X, y,
                    alpha = 1, nfolds = 5, lambda = seq(0, 10, .001))
lasso2_5cv_lambda <- lasso2$lambda.min
(cv_lambdas <- c(lasso_5cv_lambda, lasso2_5cv_lambda)) # hmm...
```

```
## [1] 4.620 0.023
```

```r
lasso2_predictions <- predict(lasso2, s = lasso2_5cv_lambda,
                      newx = as.matrix(test[,-14]))
cat("Test MSE:", mean((test[, 14] - lasso2_predictions) ^ 2),
    "(compare to", lasso_mse, "above)\n")
```

```
## Test MSE: 25.16253 (compare to 26.94795 above)
```

```r
# Mixing lambda's:
lasso_predictions_object2 <- predict.lars(lasso, as.matrix(test[,-14]),
                        mode = "lambda", s = lasso2_5cv_lambda)
lasso_mse2 <- mean((test[, 14] - lasso_predictions_object2$fit) ^ 2)
cat("Test MSE:", lasso_mse2, "\n")
```

```
## Test MSE: 25.14952
```

```r
# That's lower, ofc possible since my earlier lambda search isn't perfect.

# Then let's try this:
min(cv.lars(X, y, K = 5, type = "lasso",
                         s = lasso_5cv_lambda, plot.it = FALSE)$cv)
```

```
## [1] 24.08372
```

```r
min(cv.lars(X, y, K = 5, type = "lasso",
                         s = lasso2_5cv_lambda, plot.it = FALSE)$cv)
```

```
## [1] 24.38932
```

```r
# CV error using the provided function _is_ lower for the s we found earlier --
# we'll keep using it (so as not to use our test set in training).
```

Model comparison

```
##  Least squares on full train set: R2 = 0.7185375
##  Best subset least squares (subset found with 5-fold cv): R2 = 0.7189634
##  Ridge (shrinkage found with 5-fold cv): R2 = 0.7189073
##  Lasso (shrinkage found with 5-fold cv): R2 = 0.6984197
##  Mean of MEDV:  22.53281
##  Std dev of MEDV:  9.197104
```

A. The magnitude of difference in prediction error is small between the four models (which have all been fitted to the same training data).

B. The first least squares model is regressed on all independent variables that are available. All other models have some method for reducing the variance of found coefficients. These other models may seem a bit more biased during training, but should predict better with unseen input data and be more interpretable (especially in case of multicollinearity). The test set is unseen data, and the full least squares model nevertheless performs quite well on it. It is not much harder to interpret either – the subset of variables used in 6 is not much smaller (`best_5cv_n` vs 13, only AGE and INDUS are left out), and the coefficients of the ridge and least squares regressions are very similar even with our large `ridge_gcv_lambda`.

```
best_5cv_n
```

```
## [1] 11
```

```
ridge_gcv_lambda
```

```
## [1] 3.96
```

```
round(cbind("Full least squares" = coef(lmf), "Ridge" = coef(ridge),
      ".2 tr mean" = c(NA, apply(data[, -14], 2, mean, trim = .2)),
      "Std dev" = c(NA, apply(data[, -14], 2, sd)),
      "Ridge * mean.2" =  coef(ridge) * c(NA, apply(data[, -14], 2,
                                                     mean, trim = .2))), 2)
```

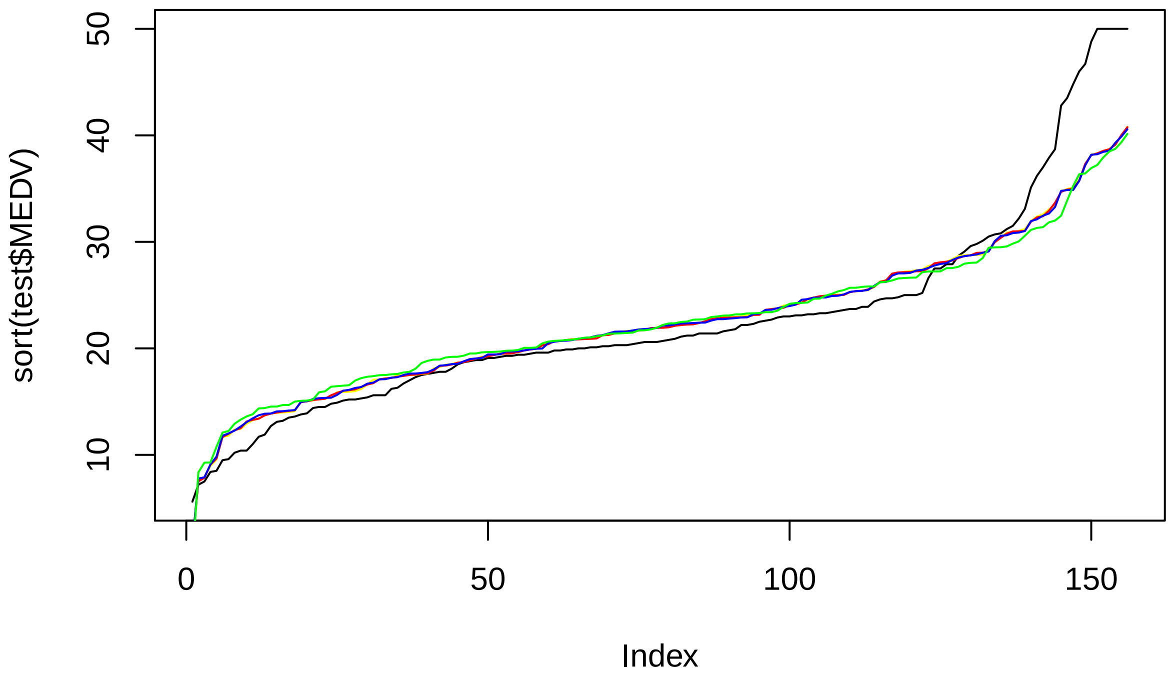
```
##              Full least squares  Ridge .2 tr mean Std dev Ridge * mean.2
## (Intercept)               39.40  37.17         NA      NA             NA
## CRIM                      -0.11  -0.11       0.90    8.60          -0.10
## ZN                         0.06   0.06       1.91   23.32           0.11
## INDUS                      0.02   0.01      10.92    6.86           0.07
## CHAS                       1.49   1.58       0.00    0.25           0.00
## NOX                      -17.03 -15.71       0.54    0.12          -8.46
## RM                         3.36   3.45       6.23    0.70          21.47
## AGE                       -0.01  -0.01      73.36   28.15          -0.43
## DIS                       -1.64  -1.56       3.38    2.11          -5.26
## RAD                        0.30   0.26       7.07    8.71           1.83
## TAX                       -0.01  -0.01     379.73  168.54          -4.50
## PTRATIO                   -0.84  -0.83      18.90    2.16         -15.59
## B                          0.01   0.01     388.67   91.29           2.48
## LSTAT                     -0.52  -0.51      11.62    7.14          -5.93
```

Since it is not done elsewhere, a brief interpretation of these coefficients seems appropriate. Remarking on statistical significance of linear relations between variables may require meeting all linear model assumptions, but we can see what influences the predictions of our models. Note that the coefficients are not normalized, and therefore that the last three columns may be useful.

We see that many of the provided variables can have a negative influence on house price, most notably DIS, NOX, PTRATIO and LSTAT (the latter four have relatively high variance). RM, RAD, B, and CHAS are influential in predicting higher house prices. B (i.e. $1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town) has, subjectively, the most surprising influence on prediction.

C. The unadjusted R^2 values when predicting "median value of owner-occupied homes in $1000's" on (unseen) test data shown above are all around .7. This is equivalent to saying the models explain around 70% of the variance of MEDV.

```
plot(sort(test$MEDV), pch = 20, type = "l")
lines(sort(lmf_predictions), col = "yellow")
lines(sort(subset_predictions), col = "red")
lines(sort(ridge_predictions), col = "blue")
lines(sort(lasso_predictions), col = "green")
```



It is again subjective, but I think these models are useful – especially in non-extreme cases.

D. The 5-fold cross-validation error is a bit lower than the test error. This was *not* inevitable. CV error is only a much better estimate of test error than training error (which was around 20) – we no longer evaluate our fit on data used to come up with the fit.

```
lmf_mse
```

```
## [1] 25.1503
```

```
lmf_5cv_mse
```

```
## [1] 23.17358
```