ML Basics

Setup

Load needed packages.

```
library(mlbench)
library(boot)
library(tidymodels)
```

```
## -- Attaching packages ------------------------------------- tidymodels 1.2.0 --

## v broom        1.0.7     v recipes      1.1.0
## v dials        1.3.0     v rsample      1.2.1
## v dplyr        1.1.4     v tibble       3.2.1
## v ggplot2      3.5.1     v tidyr        1.3.1
## v infer        1.0.7     v tune         1.2.1
## v modeldata    1.4.0     v workflows    1.1.4
## v parsnip      1.2.1     v workflowsets 1.1.0
## v purrr        1.0.2     v yardstick    1.3.1

## -- Conflicts ---------------------------------------- tidymodels_conflicts() --
## x purrr::discard() masks scales::discard()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x recipes::step()  masks stats::step()
## * Dig deeper into tidy modeling with R at https://www.tmwr.org
```

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

```
data(BostonHousing2)
boston <- BostonHousing2
head(boston)
```

```
##         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
```

Regression in R

In this section, we begin with estimating a fairly simple regression model using the median home value as the outcome and four variables as predictors.

```
m1 <- glm(medv ~ crim + chas + age + lstat, data = boston)
summary(m1)
```

```
##
## Call:
## glm(formula = medv ~ crim + chas + age + lstat, data = boston)
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 32.73813    0.73635  44.460  < 2e-16 ***
## crim        -0.07492    0.03543  -2.115   0.0350 *
## chas1        4.44525    1.07516   4.135 4.17e-05 ***
## age          0.02987    0.01220   2.448   0.0147 *
## lstat       -0.97132    0.05026 -19.326  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 36.61201)
##
##     Null deviance: 42716  on 505  degrees of freedom
## Residual deviance: 18343  on 501  degrees of freedom
## AIC: 3264.7
##
## Number of Fisher Scoring iterations: 2
```

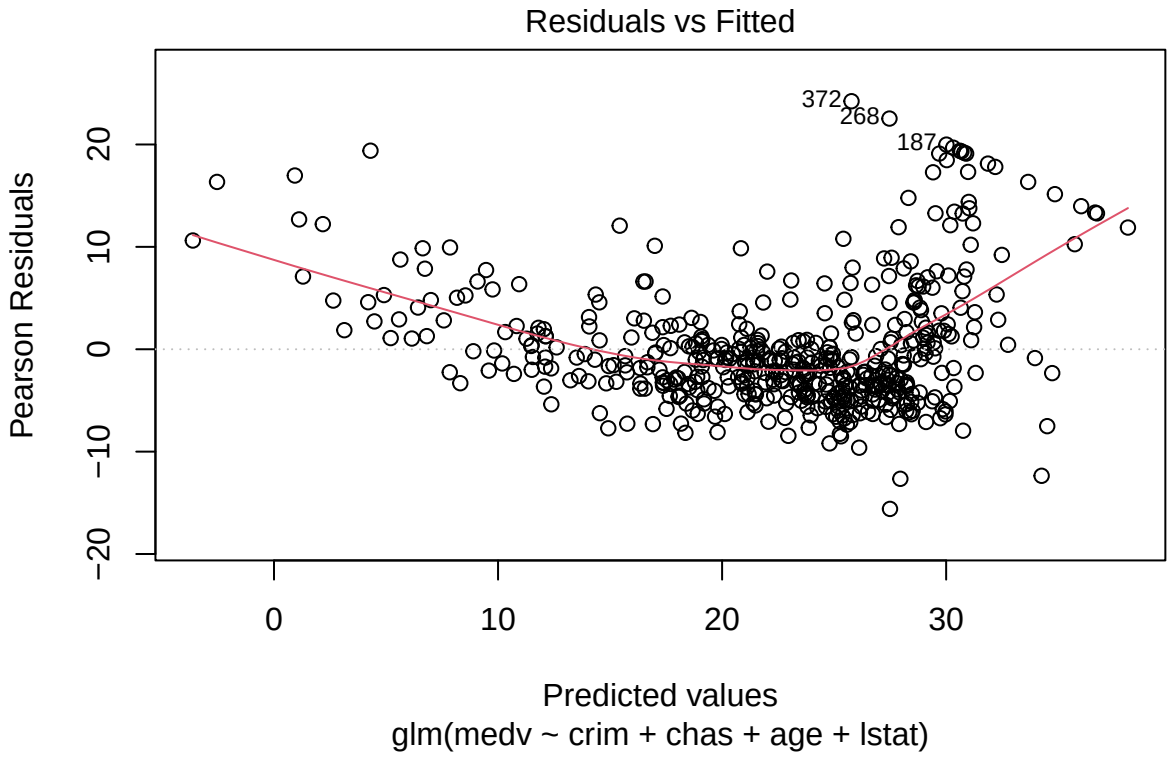
Some more information about our first model.

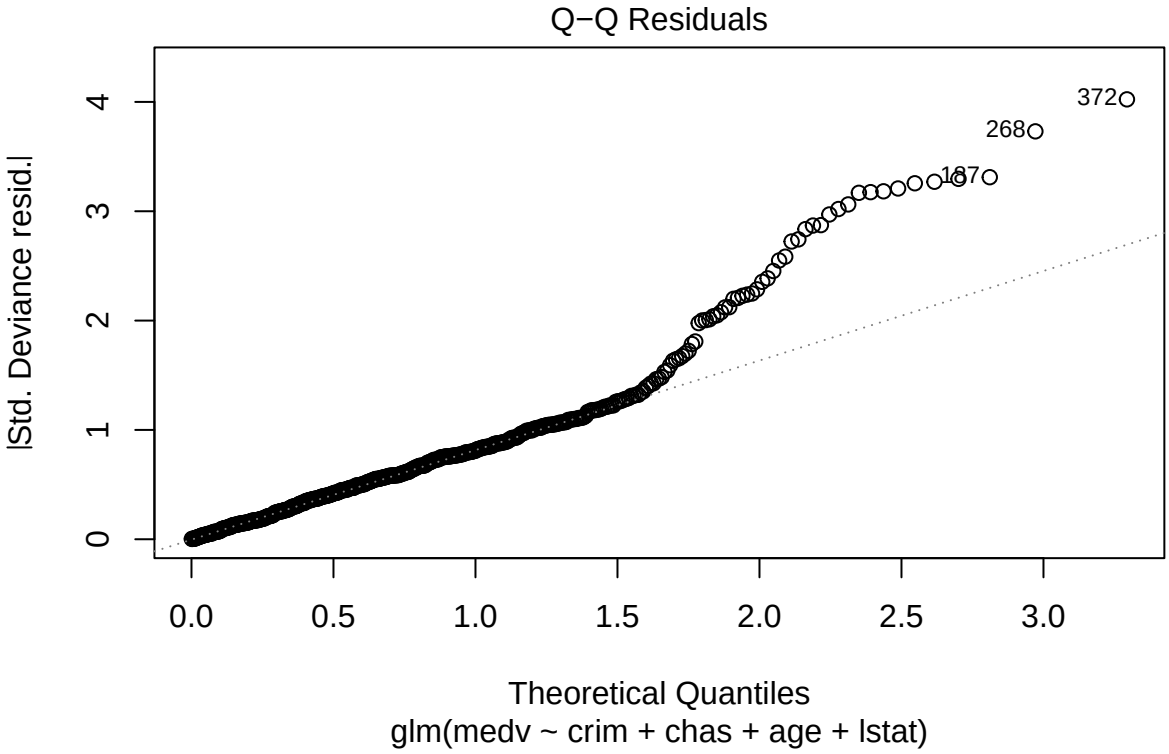
```
anova(m1)
```

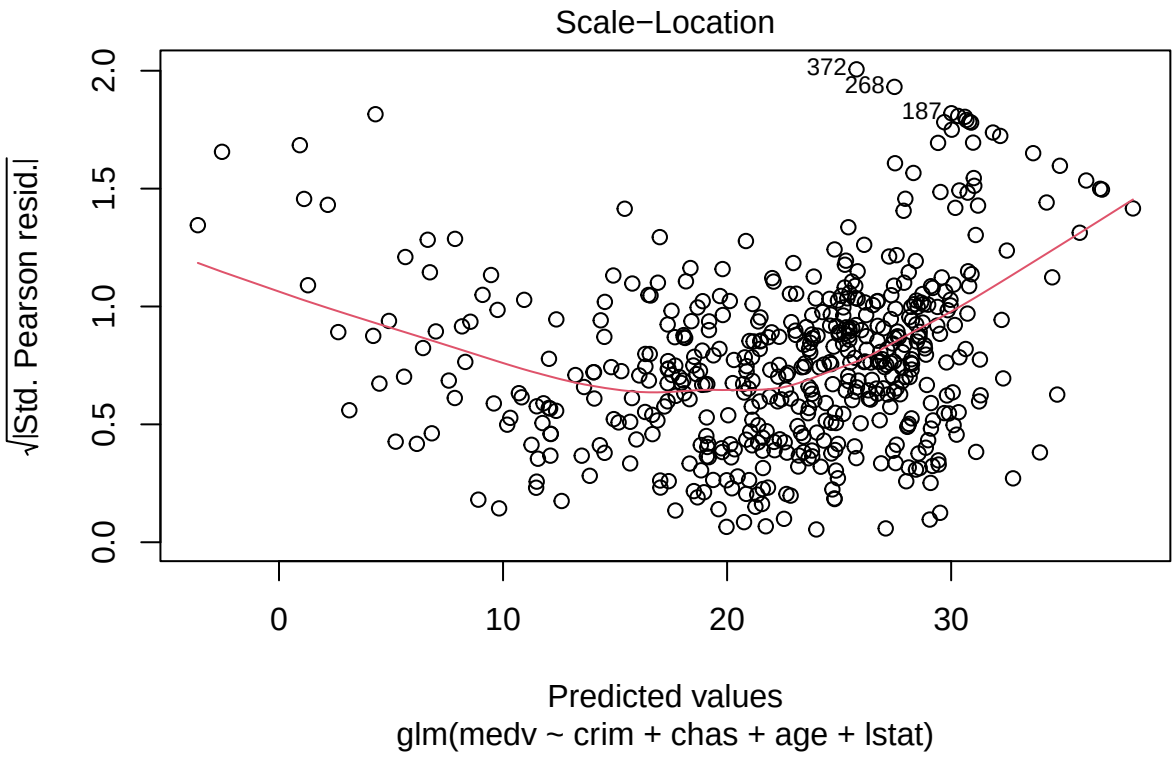
```
## Analysis of Deviance Table
```

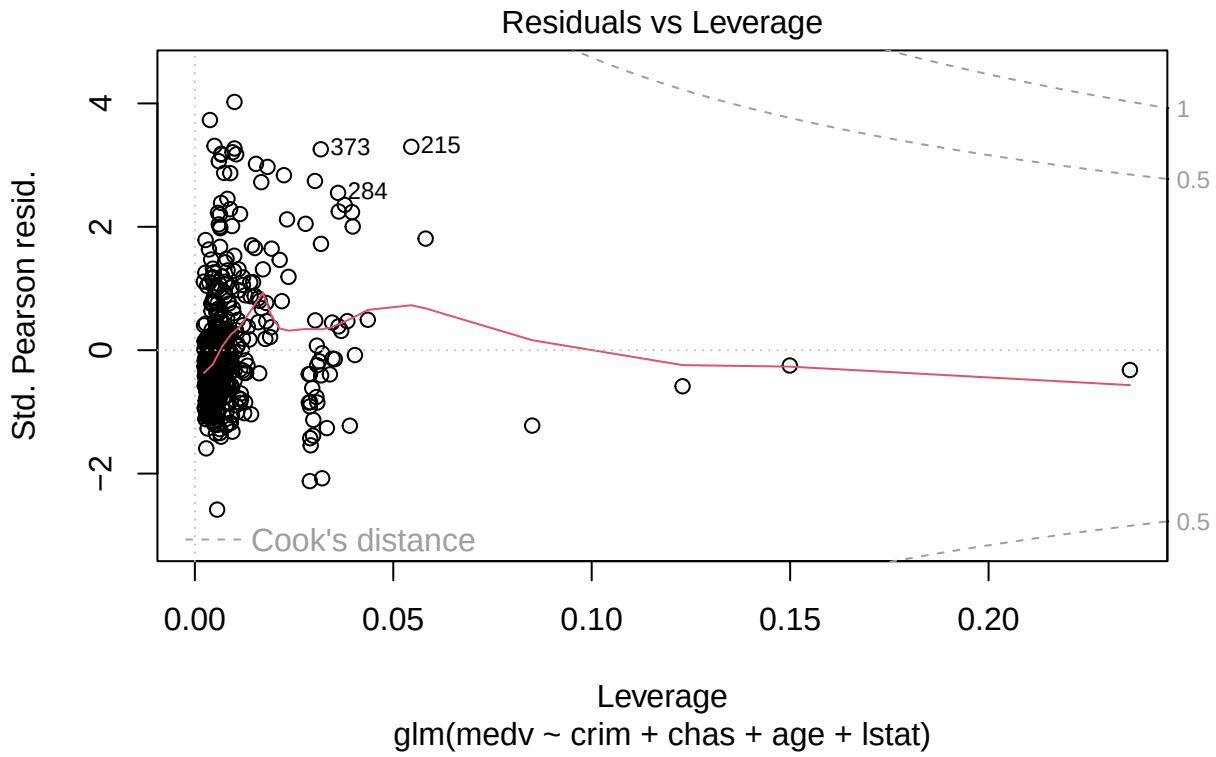
```
##
## Model: gaussian, link: identity
##
## Response: medv
##
## Terms added sequentially (first to last)
##
##
##       Df Deviance Resid. Df Resid. Dev       F    Pr(>F)
## NULL                    505      42716
## crim   1   6440.8       504      36276 175.920 < 2.2e-16 ***
## chas   1   1010.4       503      35265  27.598 2.213e-07 ***
## age    1   3248.0       502      32017  88.714 < 2.2e-16 ***
## lstat  1  13674.5       501      18343 373.497 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
plot(m1)
```




Q−Q Residuals
|Std. Deviance resid.|
0
1
2
3
4
0.0
0.5
1.0
1.5
2.0
2.5
3.0
372
268
187
Theoretical Quantiles
glm(medv ~ crim + chas + age + lstat)


Scale−Location
372
268
187
√|Std. Pearson resid.|
0.0
0.5
1.0
1.5
2.0
0
10
20
30
Predicted values
glm(medv ~ crim + chas + age + lstat)



We can use `predict` to compute predicted home values based on our regression model.

```
boston$pred1 <- predict(m1)
head(boston[,c(5,20)])
```

```
##   medv    pred1
## 1 24.0 29.84777
## 2 21.6 26.21468
## 3 34.7 30.64650
## 4 33.4 31.24791
## 5 36.2 29.17458
## 6 28.7 29.42848
```

Next, we fit an extended model that includes `lstat` squared as an additional predictor variable.

```
m2 <- glm(medv ~ crim + chas + age + lstat + I(lstat^2), data = boston)
summary(m2)
```

```
##
## Call:
## glm(formula = medv ~ crim + chas + age + lstat + I(lstat^2),
##     data = boston)
```

```
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 40.752917   0.849614  47.966  < 2e-16 ***
## crim        -0.128786   0.030343  -4.244 2.61e-05 ***
## chas1        3.662587   0.915025   4.003 7.21e-05 ***
## age          0.069733   0.010753   6.485 2.13e-10 ***
## lstat       -2.645215   0.127447 -20.755  < 2e-16 ***
## I(lstat^2)   0.050618   0.003631  13.939  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 26.41852)
##
##     Null deviance: 42716  on 505  degrees of freedom
## Residual deviance: 13209  on 500  degrees of freedom
## AIC: 3100.6
##
## Number of Fisher Scoring iterations: 2
```

Both models were fitted using the full data set. Evaluating the prediction performance of these models on the same data gives us their training error. Here, we compute the training MSE.

```
mean((predict(m1) - boston$medv)^2)
```

```
## [1] 36.25024
```

```
mean((predict(m2) - boston$medv)^2)
```

```
## [1] 26.10525
```

Train and test set

However, to get an estimate of the test error we have to proceed differently. A simple option is to split the data into a train and test set by random. Here we use `sample` to prepare and 80 to 20 percent split.

```
set.seed(7345)
Boston_split <- initial_split(BostonHousing2, prop = .80, strata = NULL, breaks = 2, pool = 0.1)
```

The resulting object gives us the row positions of the sampled elements. We use these positions to split the data into two pieces.

```
boston_train <- training(Boston_split)
boston_test <- testing(Boston_split)
```

Now, refit the previous regression model using the training set only.

```
m3 <- glm(medv ~ crim + chas + age + lstat, data = boston_train)
m4 <- glm(medv ~ crim + chas + age + lstat + I(lstat^2), data = boston_train)
```

On this basis, we use these models to predict home values in the hold-out test set.

```
pred3 <- predict(m3, newdata = boston_test)
pred4 <- predict(m4, newdata = boston_test)
```

And evaluate the prediction performance in the test set.

```
mean((pred3 - boston_test$medv)^2)
```

```
## [1] 46.83435
```

```
mean((pred4 - boston_test$medv)^2)
```

```
## [1] 36.11249
```

Regression and CV

Another (better) evaluation approach is to use cross-validation. To demonstrate how cross-validation works, we will build our own CV loop by hand. We start by shuffling the data with `sample()` and then create 10 random `folds` (groups).

```
set.seed(7346)
boston <- boston[sample(nrow(boston)),]
folds <- cut(seq(1, nrow(boston)), breaks = 10, labels = FALSE)
table(folds)
```

```
## folds
##  1  2  3  4  5  6  7  8  9 10
## 51 51 50 51 50 51 50 51 50 51
```

```
set.seed(7346)
folds_2 <- vfold_cv(boston, v = 10)
```

In the following loop, each group is used as a hold-out fold once per iteration (`test_data`). The other groups (`train_data`) are used to fit the regression model, which is then evaluated on the hold-out fold. This results in 10 test MSEs, one for each iteration.

```r
pred <- rep(NA, nrow(boston))

for(i in 1:10){
    holdout <- which(folds==i)
    test_data <- boston[holdout, ]
    train_data <- boston[-holdout, ]

    m <- glm(medv ~ crim + chas + age + lstat, data = train_data)
    pred[holdout] <- predict(m, newdata = test_data)
    print(mean((pred[holdout] - boston$medv[holdout])^2))
}
```

```
## [1] 39.86029
## [1] 49.81658
## [1] 35.04324
## [1] 47.37227
## [1] 17.06396
## [1] 34.60562
## [1] 37.26992
## [1] 41.01059
## [1] 41.69462
## [1] 29.93597
```

```r
# experimental for tidymodels
rf_mod <-
  linear_reg() %>%
  set_engine("glm", family = stats::gaussian()) %>%
  translate()

rf_wf <-
  workflow() %>%
  add_model(rf_mod) %>%
  add_formula(medv ~ crim + chas + age + lstat)

rf_fit_rs <-
  rf_wf %>%
  fit_resamples(folds_2)



# Define the recipe
rec <- recipe(qsec ~ hp, data = mtcars)

# Specify the linear regression model
lm_spec <- linear_reg()

# Combine recipe and model specification into a workflow
workflow <- workflow() %>%
  add_recipe(rec) %>%
  add_model(lm_spec)
```

Computing the MSE over all hold-out observations gives us the cross-validated MSE.

```
mean((pred - boston$medv)^2)
```

```
## [1] 37.40367
```

Cross-validation is implemented in many R packages, which typically allow more flexibility. For regression, we could e.g. use `cv.glm()` from the `boot` package. The default setting is to run leave-one-out cross-validation. For more information see `?cv.glm`.

```
cv.err <- cv.glm(boston, m1)
cv.err$delta
```

```
## [1] 133.46155  36.49592
```

We could also do 5-fold...

```
cv.err5 <- cv.glm(boston, m1, K = 5)
cv.err5$delta
```

```
## [1] 133.65917  36.57513
```

...or 10-fold CV.

```
cv.err10 <- cv.glm(boston, m1, K = 10)
cv.err10$delta
```

```
## [1] 133.93646  36.90992
```

On this basis, we can now check whether the extended model does not only yield a lower training error, but also performs better when using hold-out sets for model evaluation.

```
cv.err10.2 <- cv.glm(boston, m2, K = 10)
cv.err10.2$delta
```

```
## [1] 141.75589  26.47675
```