Exercise 09 - Overfitting

Zoltan Kekecs

13 november 2020

Contents

1 Abstract **2**

2 Data management and descriptive statistics **2**
2.1 Loading packages 2
2.2 Load data about housing prices in King County, USA 2
2.3 Check the dataset 2

3 Overfitting **2**
3.1 First rule of model selection: 2
3.2 Comparing model performance on the training set and the test set 3
3.3 Result-based models selection 7
3.4 Testing performance on the test set 8
3.5 BOTTOM LINE 8

1 Abstract

This exercise will demonstrate how overfitting can lead to bad decisions about the importance of predictors and goodness of fit of models. The basic idea is that by providing too much flexibility for the model, we allow it to fit to the sample it is being trained on too well. However, this close fit to the sample will also fit some of the patterns that are only present in our sample (due to the sampling error), and are not present in the whole population.

2 Data management and descriptive statistics

2.1 Loading packages

You will need to load the following packages for this exercise:

```
library(tidyverse)  # for tidy format
```

2.2 Load data about housing prices in King County, USA

In this exercise we will predict the price of apartments and houses.

We use a dataset from Kaggle containing data about housing prices and variables that may be used to predict housing prices. This dataset contains house sale prices for King County, USA (Seattle and sorrounding area) which includes Seattle. It includes homes sold between May 2014 and May 2015. More info about the dataset here: https://www.kaggle.com/harlfoxem/housesalesprediction

We only use a portion of the full dataset now containing information about N = 200 accomodations.

You can load the data with the following code

```
data_house = read_csv("https://bit.ly/2DpwKOr")
```

2.3 Check the dataset

You should always get familiar with the dataset you are usung, and check for any inconsistencies that need to be corrected.

In the code below we convert the area metrics that are in square feet in the original dataset to square meters. We also specify that the variable has_basement is a factor.

```
data_house = data_house %>% mutate(price_thsnd_USD = round(price/1000,
    0), sqm_living = sqft_living * 0.09290304, sqm_lot = sqft_lot *
    0.09290304, sqm_above = sqft_above * 0.09290304, sqm_basement = sqft_basement *
    0.09290304, sqm_living15 = sqft_living15 * 0.09290304, sqm_lot15 = sqft_lot15 *
    0.09290304, has_basement = factor(has_basement))

data_house %>% summary()
```

3 Overfitting

3.1 First rule of model selection:

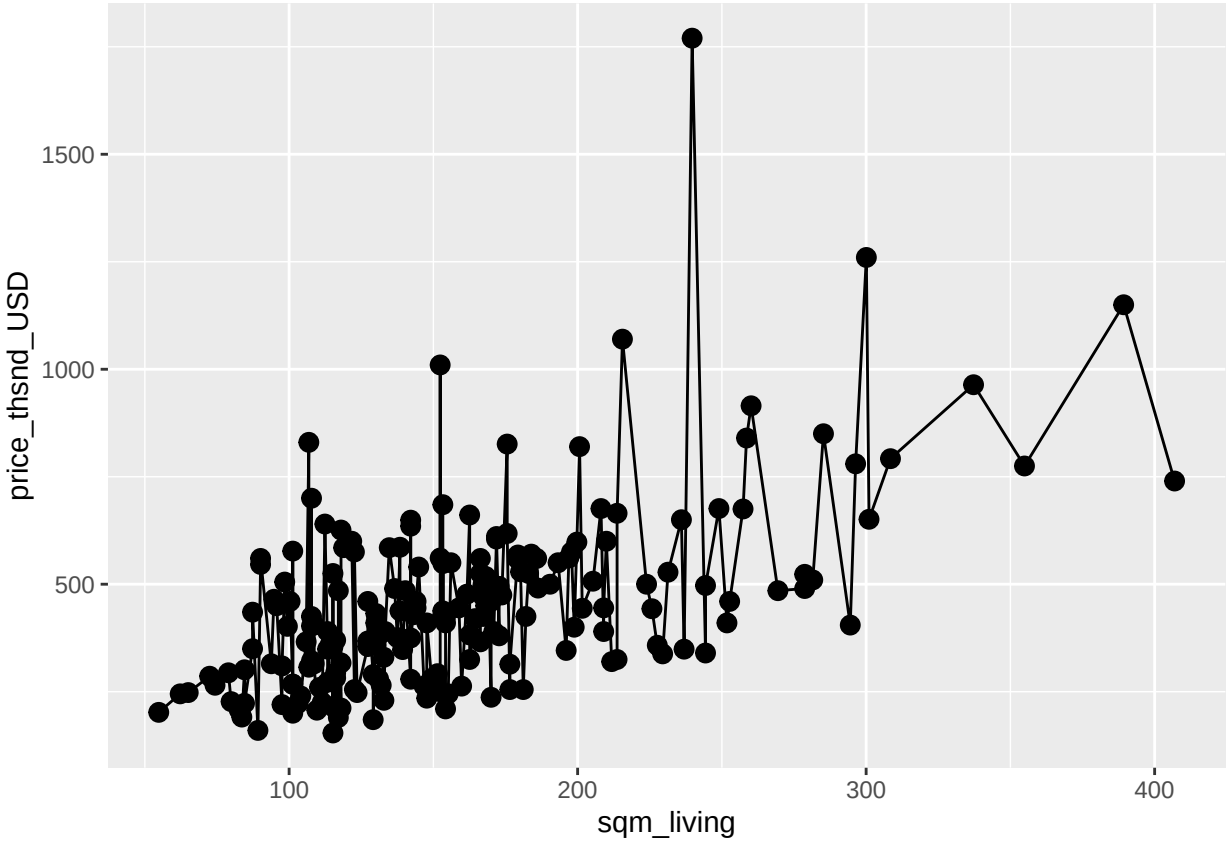
Always use the model that is grounded in theory and prior research, result-driven model selection can lead to bad predictions on new datasets due to overfitting!

"Predicting" variability of the outcome **in your original data** is easy If you fit a model that is flexible enough, you will get perfect fit on your intitial data.

For example you can fit a line that would cover your data perfectly, reaching 100% model fit... to a dataset where you already knew the outcome.

See the example "prediction line" below.

```
data_house %>% ggplot() + aes(y = price_thsnd_USD, x = sqm_living) +
    geom_point(size = 3) + geom_line()
```



However, when you try to apply the same model to new data, it will produce bad model fit. In most cases, worse, than a simple regression.

3.2 Comparing model performance on the training set and the test set

In this context, data on which the model was built is called the training set, and the new data where we test the true prediction efficiency of a model is called the test set. The test set can be truely newly collected data, or it can be a set aside portion of our old data which was not used to fit the model.

Linear regression is very inflexible, so it is less prone to overfitting. This is one of its advantages compared to more flexible prediction approaches.

In the next part of the exercise we will demonstrate that the more predictors you have, the higher your R^2 will be, even if the predictors have nothing to do with your outcome variable.

First, we will generate some random variables for demonstration purposes. These will be used as predictors in some of our models in this exercise. It is important to realize that these variables are randomly generated, and have no true relationship to the sales price_thsnd_USD of the apartments. Using these random numbers we can demonstrate well how people can be mislead by good prediction performance of models containing many predictors.

```
rand_vars = as.data.frame(matrix(rnorm(mean = 0, sd = 1, n = 50 *
    nrow(data_house)), ncol = 50))
data_house_withrandomvars = cbind(data_house, rand_vars)
```

We create a new data object from the first half of the data (N = 100). We will use this to fit our models on. This is our training set. We set aside the other half of the dataset so that we will be able to test prediction performance on it later. This is called the test set.

```
training_set = data_house_withrandomvars[1:100, ]  # training set, using half of the data
test_set = data_house_withrandomvars[101:200, ]  # test set, the other half of the dataset
```

Now we will perform a hierarchical regression where first we fit our usual model predicting price_thsnd_USD with sqm_living and grade on the training set. Next, we fit a model containing sqm_living and grade and the 50 randomly generated variables that we just created.

(the names of the random variables are V1, V2, V3, ...)

```
mod_house_train <- lm(price_thsnd_USD ~ sqm_living + grade, data = training_set)
mod_house_rand_train <- lm(price_thsnd_USD ~ sqm_living + grade +
    V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 +
    V12 + V13 + V14 + V15 + V16 + V17 + V18 + V19 + V20 + V21 +
    V22 + V23 + V24 + V25 + V26 + V27 + V28 + V29 + V30 + V31 +
    V32 + V33 + V34 + V35 + V36 + V37 + V38 + V39 + V40 + V41 +
    V42 + V43 + V44 + V45 + V46 + V47 + V48 + V49 + V50, data = training_set)
```

Now we can compare the model performance. First, if we look at the normal R^2 indexes of the models or the RSS, we will find that the model using the random variables (mod_house_rand_train) was much better at predicting the training data. The error was smaller in this model, and the overall variance explained is bigger. You can even notice that some of the random predictors were identified as having significant added prediction value in this model, even though they are not supposed to be related to price_thsnd_USD at all, since we just created them randomly. This is because some of these variables are alligned with the outcome to some extend by random chance.

```
summary(mod_house_train)
```

```
##
## Call:
## lm(formula = price_thsnd_USD ~ sqm_living + grade, data = training_set)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -420.74  -84.22  -15.67   81.09  471.34
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -319.1069   123.3489  -2.587 0.011165 *
## sqm_living     1.2449     0.3534   3.523 0.000653 ***
## grade         77.8230    21.5044   3.619 0.000472 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 148.3 on 97 degrees of freedom
## Multiple R-squared:  0.4765, Adjusted R-squared:  0.4657
## F-statistic: 44.15 on 2 and 97 DF,  p-value: 2.322e-14
```

```
summary(mod_house_rand_train)
```

```
##
## Call:
## lm(formula = price_thsnd_USD ~ sqm_living + grade + V1 + V2 +
##     V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 + V12 + V13 +
##     V14 + V15 + V16 + V17 + V18 + V19 + V20 + V21 + V22 + V23 +
##     V24 + V25 + V26 + V27 + V28 + V29 + V30 + V31 + V32 + V33 +
##     V34 + V35 + V36 + V37 + V38 + V39 + V40 + V41 + V42 + V43 +
##     V44 + V45 + V46 + V47 + V48 + V49 + V50, data = training_set)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -228.544  -61.623   -9.312   65.703  219.479
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -269.3619   150.1422  -1.794 0.079240 .
## sqm_living     1.6878     0.4421   3.818 0.000393 ***
## grade         59.4658    27.2174   2.185 0.033921 *
## V1            11.1839    16.5483   0.676 0.502461
## V2           -39.0622    17.1845  -2.273 0.027633 *
## V3            13.9364    20.0882   0.694 0.491247
## V4            15.4573    19.6187   0.788 0.434720
## V5             1.4713    16.1061   0.091 0.927603
## V6           -43.0891    24.2184  -1.779 0.081676 .
## V7            18.3769    18.5766   0.989 0.327606
## V8             5.4535    19.4030   0.281 0.779895
## V9            16.8922    18.9619   0.891 0.377546
## V10            4.7760    18.0340   0.265 0.792297
## V11           42.1352    17.9683   2.345 0.023303 *
## V12           12.5766    21.6102   0.582 0.563363
## V13           30.9266    18.1821   1.701 0.095564 .
## V14          -26.6132    22.2485  -1.196 0.237627
## V15            1.8907    19.0288   0.099 0.921274
## V16          -28.2452    18.0379  -1.566 0.124085
## V17          -13.3120    16.1728  -0.823 0.414602
## V18           18.6853    14.9488   1.250 0.217505
## V19          -21.1914    22.3292  -0.949 0.347452
## V20            9.6385    16.4675   0.585 0.561142
## V21           -2.3215    17.7317  -0.131 0.896394
## V22           -1.9163    20.8848  -0.092 0.927282
## V23           10.1695    21.9981   0.462 0.646007
## V24          -31.8917    20.7022  -1.541 0.130144
## V25           16.0240    18.0752   0.887 0.379850
## V26           27.9319    15.6608   1.784 0.080954 .
## V27           43.2225    16.7506   2.580 0.013054 *
## V28           -3.7206    17.6758  -0.210 0.834195
## V29          -10.6476    18.6152  -0.572 0.570061
## V30          -31.7692    18.1000  -1.755 0.085741 .
## V31           28.4814    19.0433   1.496 0.141439
## V32           32.5897    16.2570   2.005 0.050780 .
## V33          -47.1342    20.5956  -2.289 0.026646 *
## V34          -21.8229    21.8922  -0.997 0.323950
## V35          -14.2073    18.7271  -0.759 0.451850
## V36           16.1612    16.8721   0.958 0.343031
```

```
## V37           12.9263    20.5396   0.629 0.532177
## V38           -4.0104    20.1569  -0.199 0.843151
## V39            7.9160    17.9027   0.442 0.660395
## V40           17.0964    18.8642   0.906 0.369407
## V41            5.5350    20.5785   0.269 0.789130
## V42          -41.1877    21.7988  -1.889 0.065010 .
## V43           -8.4619    18.0629  -0.468 0.641614
## V44           11.8760    18.8132   0.631 0.530930
## V45            1.8539    20.2894   0.091 0.927586
## V46           15.0402    20.4363   0.736 0.465415
## V47          -21.1707    18.3387  -1.154 0.254159
## V48          -17.0836    19.0706  -0.896 0.374921
## V49           40.3486    18.4911   2.182 0.034138 *
## V50            5.3430    17.7366   0.301 0.764561
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 127.7 on 47 degrees of freedom
## Multiple R-squared:  0.8118, Adjusted R-squared:  0.6037
## F-statistic:   3.9 on 52 and 47 DF,  p-value: 2.811e-06
```

```
pred_train <- predict(mod_house_train)
pred_train_rand <- predict(mod_house_rand_train)
RSS_train = sum((training_set[, "price_thsnd_USD"] - pred_train)^2)
RSS_train_rand = sum((training_set[, "price_thsnd_USD"] - pred_train_rand)^2)
RSS_train
```

```
## [1] 2133370
```

```
RSS_train_rand
```

```
## [1] 766817.4
```

That is why we need to use model fit indexes that are more sensitive to the number of variables we included as redictors, to account for the likelyhood that some variables will show a correlation by chance. Such as adjusted R^2, or the AIC. The anova() test is also sensitive to the number of predictors in the models, so it is not easy to fool by adding a bunch of random data as predictors. Better yet, the AIC indicates that smaller model is significantly better than the more complicated model.

```
summary(mod_house_train)$adj.r.squared
```

```
## [1] 0.4657482
```

```
summary(mod_house_rand_train)$adj.r.squared
```

```
## [1] 0.6036804
```

```
AIC(mod_house_train)
```

```
## [1] 1288.592
```

```
AIC(mod_house_rand_train)
```

```
## [1] 1286.271
```

```
anova(mod_house_train, mod_house_rand_train)
```

```
## Analysis of Variance Table
##
## Model 1: price_thsnd_USD ~ sqm_living + grade
```

```
## Model 2: price_thsnd_USD ~ sqm_living + grade + V1 + V2 + V3 + V4 + V5 +
##     V6 + V7 + V8 + V9 + V10 + V11 + V12 + V13 + V14 + V15 + V16 +
##     V17 + V18 + V19 + V20 + V21 + V22 + V23 + V24 + V25 + V26 +
##     V27 + V28 + V29 + V30 + V31 + V32 + V33 + V34 + V35 + V36 +
##     V37 + V38 + V39 + V40 + V41 + V42 + V43 + V44 + V45 + V46 +
##     V47 + V48 + V49 + V50
##   Res.Df     RSS Df Sum of Sq      F  Pr(>F)
## 1     97 2133370
## 2     47  766817 50   1366553 1.6752 0.03844 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

3.3 Result-based models selection

(Result-based models selection is only shown here with demonstration purposes, to show how it can mislead researchers. Whenever possible, stay away from using such approaches, and rely on theoretical considerations and previous data when building models.)

After seeing the performance of mod_house_rand_train, and not knowing that it contains random variables, one might be tempted to build a model with only the predictors that were identified as having a significant added predictive value, to improve the model fit indices (e.g. adjusted R^2 or AIC). And that would achieve exactly that: it would result in the virtual improvement of the indexes, but not the actual prediction efficiency, so the better indexes would be just an illusion resulting from the fact that we "concealed" from the statistical tests that we have tried to use a lot of predictors in a previous model.

Excluding variables that seem "useless" based on the results will blind the otherwise sensitive measures of model fit. This is what happens when using automated model selection procedures, such as backward regression.

In the example below we use backward regression. This method first fits a complete model with all of the specified predictors, and then determins which predictor has the smallest amount of unique added explanatory value to the model, and excludes it from the list of predictors, refitting the model without this predictor. This procedure is iterated until until there is no more predictor that can be excluded without significantly reducing model fit, at which point the process stops.

```
mod_back_train = step(mod_house_rand_train, direction = "backward")
```

The final model with the reduced number of predictors will have much better model fit indexes than the original compex model, because the less useful variables were excluded, and only the most influential ones were retained, resulting in a small and powerful model. Or at least this is what the numbers would suggest us on the training set.

Lets compare the prediction performance of the final model returned by backward regression (mod_back_train) with the model only containing our good old predictors, sqm_living and grade (mod_house_train) on the training set.

```
anova(mod_house_train, mod_back_train)
```

```
## Analysis of Variance Table
##
## Model 1: price_thsnd_USD ~ sqm_living + grade
## Model 2: price_thsnd_USD ~ sqm_living + grade + V2 + V4 + V6 + V7 + V9 +
##     V11 + V14 + V16 + V18 + V24 + V25 + V26 + V27 + V30 + V31 +
##     V32 + V33 + V34 + V42 + V46 + V47 + V49
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1     97 2133370
## 2     75  912618 22   1220752 4.5601 4.005e-07 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(mod_house_train)$adj.r.squared
```

```
## [1] 0.4657482
```

```
summary(mod_back_train)$adj.r.squared
```

```
## [1] 0.7044171
```

```
AIC(mod_house_train)
```

```
## [1] 1288.592
```

```
AIC(mod_back_train)
```

```
## [1] 1247.678
```

All of the above model comparison methods indicate that the backward regression model (mod_back_train) performs better. We know that this model can't be too much better than the smaller model, since it only contains a number of randomly generated variables in addition to the two predictors in the smaller model. So if we would only rely on these numbers, we would be fooled to think that the backward regression model is better.

3.4 Testing performance on the test set

A surefire way of determining actual model performance is to test it on new data, data that was not used in the "training" of the model. Here, we use the set aside test set to do this.

Note that we did not re-fit the models on the test set, we use the models fitted on the training set to make our predictions using the predict() function on the test_set!!!

```
# calculate predicted values
pred_test <- predict(mod_house_train, test_set)
pred_test_back <- predict(mod_back_train, test_set)

# now we calculate the sum of squared residuals
RSS_test = sum((test_set[, "price_thsnd_USD"] - pred_test)^2)
RSS_test_back = sum((test_set[, "price_thsnd_USD"] - pred_test_back)^2)
RSS_test
```

```
## [1] 3639724
```

```
RSS_test_back
```

```
## [1] 5771743
```

This test reveals that the backward regression model has more error than the model only using sqm_living and grade.

3.5 BOTTOM LINE

1. Model selection should be done pre-analysis, based on theory, previous results from the literature, or conventions on the field. Post-hoc result-driven predictor selection can lead to overfitting.
2. The only good test of a model's true prediction performance is to test the accuracy of its predictions on new data (or a set-asid test set using the above described training-set test-set approach)