Exercise 13 - Model comparison and model selection

Zoltan Kekecs

20 november 2018

Contents

1 Abstract

This exercise will show you how different models can be compared to each other. It will denonstrate hierarchical regression. We will also discuss some model selection strategies which as discouraged because they lead to unreliable predictions on new data due to overfitting.

The latest version of this document and the code the document refers to can be found in the GitHub repository of the class at: https://github.com/kekecsz/PSYP13_Data_analysis_class-2018

2 Data management and descriptive statistics

2.1 Loading packages

no specific package is needed for this exercise

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. The data is about accomodations in King County, USA (Seattle and sorrounding area).

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 from
# github/kekecsz/PSYP13_Data_analysis_class-2018/master/data_house_small_sub.csv.
data_house = read.csv("https://bit.ly/2DpwKOr")
```

2.3 Check the dataset

You should always check the dataset for coding errors or data that does not make sense, by eyeballing the data through the data view tool, checking descriptive statistics and through data visualization.

3 Hierarchical regression

Using hierarchical regression, you can quantify the amount of information gained by adding a new predictor or a set of predictors to a previous model. To do this, you will build two models, the predictors in one is the subset of the predictors in the other model.

3.1 Hierarchical regression with two predictor blocks

Here we first build a model to predict the price of the apartment by using only sqft_living and grade as predictors.

```
mod_house2 <- lm(price ~ sqft_living + grade, data = data_house)
```

Next, we want to see whether we can improve the effeffctiveness of our prediction by taking into account geographic location in our model, in addition to living space and grade

```
mod_house_geolocation = lm(price ~ sqft_living + grade + long +
    lat, data = data_house)
```

We can look at the adj. R squared statistic to see how much variance is explained by the new and the old model.

```r
summary(mod_house2)$adj.r.squared
```

```
## [1] 0.3515175
```

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

```
## [1] 0.4932359
```

It seems that the variance explained has increased substantially by adding information about geographic location to the model.

Now, we should compare residual error and model fit thought the anova() function and the AIC() function.

The anova() function can only be used for comparing models if the two models are "nested", that is, predictors in one of the models are a subset of predictors of the other model. If the anova F test is significant, it means that the models are significantly different in terms of their residual errors.

```r
anova(mod_house2, mod_house_geolocation)
```

```
## Analysis of Variance Table
##
## Model 1: price ~ sqft_living + grade
## Model 2: price ~ sqft_living + grade + long + lat
##   Res.Df        RSS Df  Sum of Sq      F    Pr(>F)
## 1    197 5.6981e+12
## 2    195 4.4076e+12  2 1.2905e+12 28.546 1.338e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

If the difference in AIC of the two models is larger than 2, the two models are significantly different in their model fit. Smaller AIC means less error and better model fit, so in this case we accept the model with the smaller AIC. However, if the difference in AIC does not reach 2, we can retain either of the two models. Ususally we stick with the less complicated model in this case, but theoretical considerations and previous results should also be considered when doing model selection.

```r
AIC(mod_house2)
```

```
## [1] 5390.142
```

```r
AIC(mod_house_geolocation)
```

```
## [1] 5342.783
```

The AIC is a more established model comparison tool, so if the anova and AIC methods return discrepant results, the AIC should be used for decision making.

3.2 Hierarchical regression with more than two blocks

The same procedure can be repeated if we have more than two steps/blocks in the hierarchical regression.

Here we build a third model, which adds even more predictors to the formula. This time, we add information about the condition of the apartment.

```r
mod_house_geolocation_cond = lm(price ~ sqft_living + grade +
    long + lat + condition, data = data_house)
```

We can compare the three models now.

```r
# R^2
summary(mod_house2)$adj.r.squared
```

```
## [1] 0.3515175
```

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

```
## [1] 0.4932359
```

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

```
## [1] 0.5065859
```

```
# anova
anova(mod_house2, mod_house_geolocation, mod_house_geolocation_cond)
```

```
## Analysis of Variance Table
##
## Model 1: price ~ sqft_living + grade
## Model 2: price ~ sqft_living + grade + long + lat
## Model 3: price ~ sqft_living + grade + long + lat + condition
##   Res.Df        RSS Df  Sum of Sq      F    Pr(>F)
## 1    197 5.6981e+12
## 2    195 4.4076e+12  2 1.2905e+12 29.318 7.493e-12 ***
## 3    194 4.2695e+12  1 1.3812e+11  6.276   0.01306 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# AIC
AIC(mod_house2)
```

```
## [1] 5390.142
```

```
AIC(mod_house_geolocation)
```

```
## [1] 5342.783
```

```
AIC(mod_house_geolocation_cond)
```

```
## [1] 5338.416
```

Did we gain substantial information about housing price by adding information about the condition of the apartment to the model?

4 Model selection

First rule of model selection:

Always go with the model that is grounded in theory and prior research, because automatic 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 too flexible, 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.

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.

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.

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

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 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 with sqft_living and grade on the training set. Next, we fit a model containing sqft_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 ~ sqft_living + grade, data = training_set)
mod_house_rand_train <- lm(price ~ sqft_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 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 ~ sqft_living + grade, data = training_set)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -420718  -84545  -15651   81155  471385
##
## Coefficients:
```

```
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -318985.28  123358.64  -2.586 0.011201 *
## sqft_living     115.71      32.83   3.524 0.000650 ***
## grade         77790.96   21506.05   3.617 0.000475 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 148300 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: 2.327e-14
```

```
summary(mod_house_rand_train)
```

```
##
## Call:
## lm(formula = price ~ sqft_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
## -267978  -82005     399   69004  288500
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -433964.54  190798.57  -2.274  0.02755 *
## sqft_living      55.88      53.82   1.038  0.30440
## grade        105636.69   34082.71   3.099  0.00327 **
## V1            40946.30   20464.86   2.001  0.05121 .
## V2           -32340.13   27417.34  -1.180  0.24412
## V3           -19985.44   20309.01  -0.984  0.33012
## V4            -3243.43   24148.43  -0.134  0.89373
## V5            -8439.53   22373.00  -0.377  0.70771
## V6           -25455.56   21266.66  -1.197  0.23732
## V7            -1845.50   19630.16  -0.094  0.92550
## V8            -7892.80   23697.67  -0.333  0.74057
## V9             -864.32   20669.74  -0.042  0.96682
## V10           28486.69   26533.13   1.074  0.28847
## V11          -13630.39   21864.38  -0.623  0.53603
## V12            9852.88   18750.62   0.525  0.60173
## V13            6560.07   19621.20   0.334  0.73961
## V14          -39928.45   22046.47  -1.811  0.07652 .
## V15           24039.64   21596.66   1.113  0.27132
## V16          -20413.31   26168.20  -0.780  0.43925
## V17            3027.18   20375.10   0.149  0.88253
## V18           17516.53   22980.39   0.762  0.44973
## V19          -16983.74   24287.51  -0.699  0.48782
## V20          -19586.96   24316.12  -0.806  0.42458
## V21           -3537.09   20367.71  -0.174  0.86288
## V22           -6735.28   22858.12  -0.295  0.76955
## V23            9263.31   22178.36   0.418  0.67809
## V24          -17901.09   22208.14  -0.806  0.42427
## V25           12040.84   21914.84   0.549  0.58531
## V26          -19134.04   27003.80  -0.709  0.48209
## V27          -12567.11   20957.13  -0.600  0.55161
## V28           12059.61   22116.85   0.545  0.58815
## V29           12718.44   22465.16   0.566  0.57399
## V30           -9503.13   21996.46  -0.432  0.66770
## V31          -13824.87   24411.15  -0.566  0.57386
## V32           11938.78   23419.79   0.510  0.61260
## V33           -5841.97   20098.72  -0.291  0.77259
## V34           -6355.61   23099.57  -0.275  0.78441
## V35           24925.10   21338.71   1.168  0.24867
## V36            9275.11   22822.09   0.406  0.68629
## V37           45258.41   24941.16   1.815  0.07597 .
## V38           45595.03   23922.44   1.906  0.06278 .
## V39           -9023.81   23556.08  -0.383  0.70339
## V40          -24101.47   24069.27  -1.001  0.32179
## V41           15791.80   20858.69   0.757  0.45278
## V42            9053.14   22991.53   0.394  0.69554
## V43           20289.55   23519.76   0.863  0.39271
## V44          -37623.31   23754.54  -1.584  0.11994
## V45          -14218.25   28430.99  -0.500  0.61934
## V46          -21329.26   28303.81  -0.754  0.45486
## V47          -20685.38   20934.95  -0.988  0.32817
## V48           26135.41   21117.49   1.238  0.22200
## V49           23647.89   25014.72   0.945  0.34931
## V50           25515.90   23915.67   1.067  0.29146
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 157500 on 47 degrees of freedom
## Multiple R-squared:  0.7138, Adjusted R-squared:  0.3971
## F-statistic: 2.254 on 52 and 47 DF,  p-value: 0.002714
```

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

```
## [1] 2.133707e+12
```

```r
RSS_train_rand
```

```
## [1] 1.16661e+12
```

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.

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

```
## [1] 0.4657206
```

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

```
## [1] 0.3971168
```

```
AIC(mod_house_train)
```

```
## [1] 2670.159
```

```
AIC(mod_house_rand_train)
```

```
## [1] 2709.783
```

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

```
## Analysis of Variance Table
##
## Model 1: price ~ sqft_living + grade
## Model 2: price ~ sqft_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 2.1337e+12
## 2     47 1.1666e+12 50 9.671e+11 0.7792 0.8069
```

4.2 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 acieve exactly that: it would result in the indcrease of the indexes, but not the actual prediction efficiency, so the better indexes would be just an illusion resulting from the fact that we have "hidden" 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, sqft_living and grade (mod_house_train) on the training set.

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

```
## Analysis of Variance Table
##
## Model 1: price ~ sqft_living + grade
## Model 2: price ~ sqft_living + grade + V1 + V6 + V10 + V11 + V14 + V20 +
##     V24 + V35 + V37 + V38 + V40 + V44 + V50
##   Res.Df        RSS Df  Sum of Sq      F   Pr(>F)
## 1     97 2.1337e+12
## 2     84 1.4735e+12 13 6.6025e+11 2.8954 0.001677 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

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

```
## [1] 0.4657206
```

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

```
## [1] 0.5739469
```

```r
AIC(mod_house_train)
```

```
## [1] 2670.159
```

```r
AIC(mod_back_train)
```

```
## [1] 2659.134
```

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.

4.2.1 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!!!

```r
# 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"] - pred_test)^2)
RSS_test_back = sum((test_set[, "price"] - pred_test_back)^2)
RSS_test
```

```
## [1] 3.639381e+12
```

```r
RSS_test_back
```

```
## [1] 4.968421e+12
```

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

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)