Linear models - model fit, prediction and cross validation

Søren Højsgaard

Fri Sep 20 07:01:29 2024

Contents

1 Choosing among models

For simplicity consider linear regression models

$$y_i = b_1 x_{i1} + \cdots + b_p x_{ip} + e_i; i = 1, \ldots, N$$

Each subset of the predictors $x_1, x_2, \ldots, x_p$ defines a model.

Two tasks to consider:

1. Suppose we are given Q different candidate models $M_1, M_2, \ldots, M_Q$ corresponding to Q different subsets of the predictors. Which of these models should we choose as the best model?
2. How do we select Q different candidate models from data?

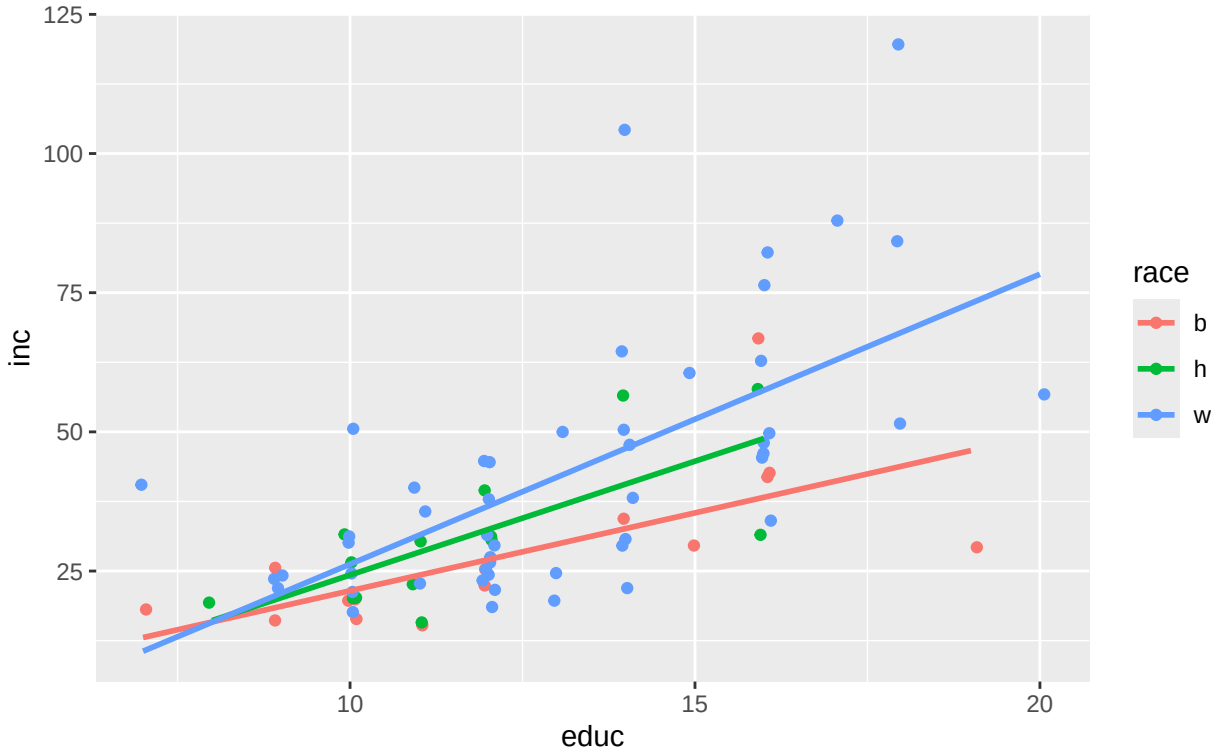
2 The `income` data

```
dat <- doBy::income
dat |> head()
```

```
##   inc educ race
## 1  16   10    b
## 2  18    7    b
## 3  26    9    b
## 4  16   11    b
## 5  34   14    b
## 6  22   12    b
```

```
pl0 <- dat |> ggplot(aes(x=educ, y=inc, color=race)) + geom_jitter(width=0.1)
pl0 + geom_smooth(method="lm", se=FALSE)
```

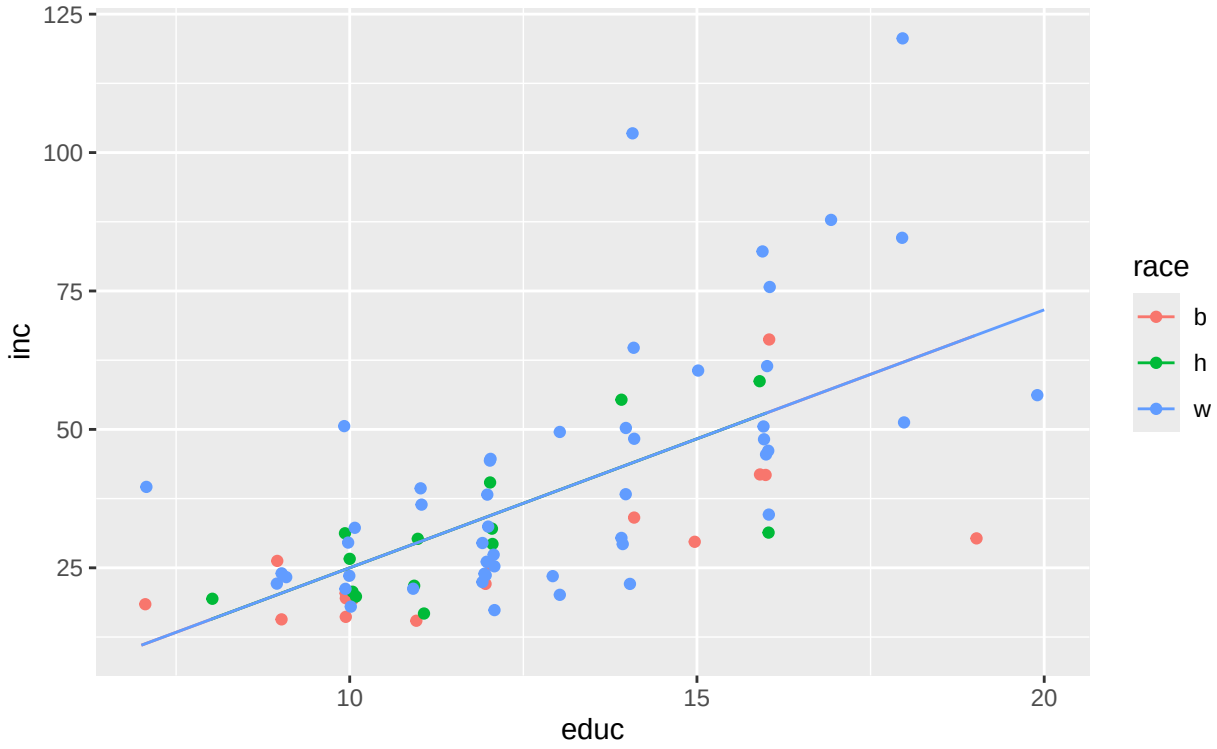
```
## `geom_smooth()` using formula = 'y ~ x'
```



2.1 Model 1 - Simple linear regression

Income grows linearly with years of education; no effect of ethnicity (Simple linear regression)

```
mm1 <- lm(inc ~ educ, data=dat)
pl0 + geom_line(aes(y=fitted(mm1)))
```



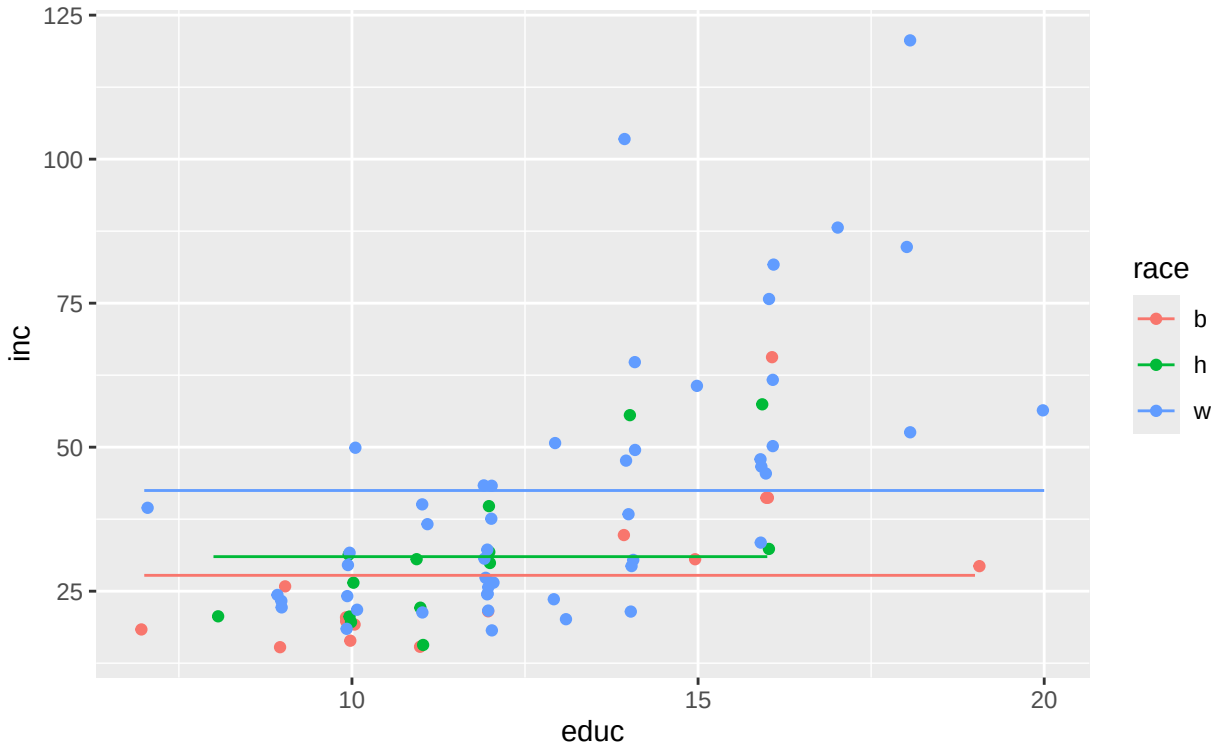

```
mm1 |> tidy()
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)   -21.6      8.08      -2.67 9.20e- 3
## 2 educ            4.66     0.622      7.50 8.85e-11
```

2.2 Model 2 - one-way ANOVA

Income is constant across all levels of education within ethnic groups

```
mm2 <- lm(inc ~ race, data=dat)
pl0 + geom_line(aes(y=fitted(mm2)))
```



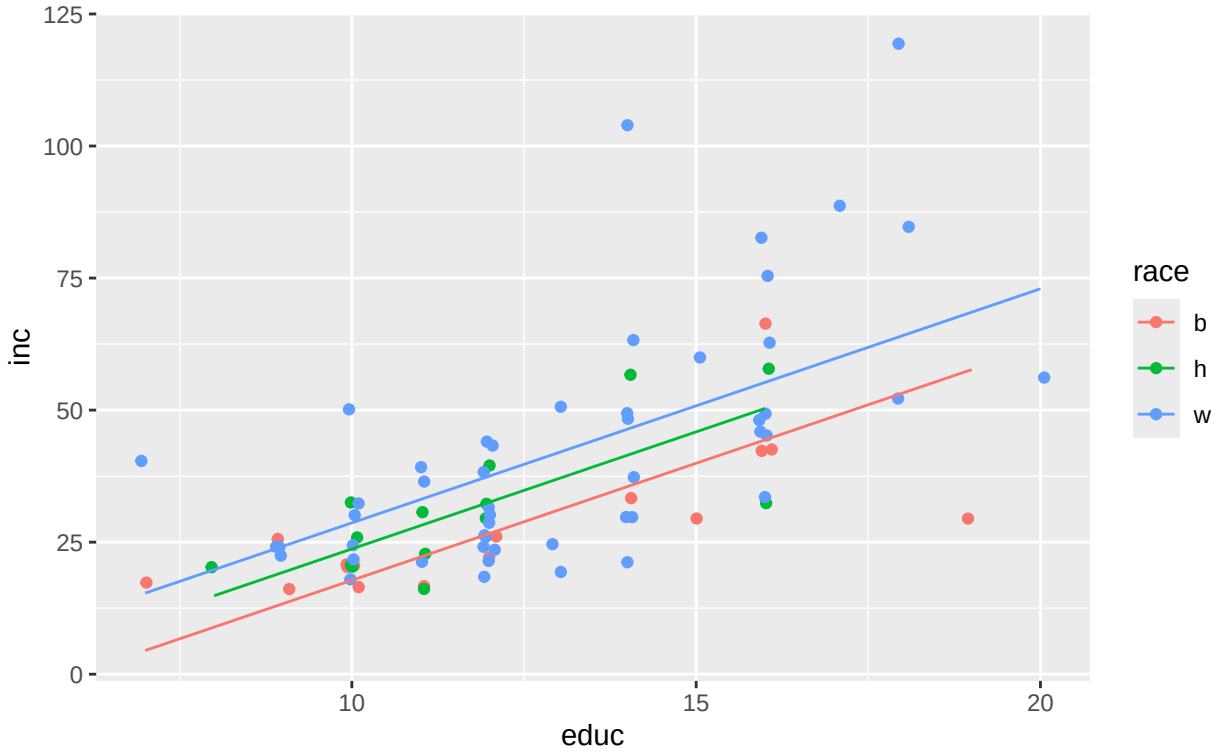

```
mm2 |> tidy()
```

```
## # A tibble: 3 x 5
##   term        estimate std.error statistic     p.value
##   <chr>          <dbl>     <dbl>     <dbl>       <dbl>
## 1 (Intercept)    27.8       4.97     5.59  0.000000337
## 2 raceh           3.25      7.27     0.447 0.656
## 3 racew          14.7       5.71     2.58  0.0118
```

2.3 Model 3 - ANCOVA

Income grows linearly with years of education but with offset depending on ethnicity

```
mm3 <- lm(inc ~ race + educ, data=dat)
pl0 + geom_line(aes(y=fitted(mm3)))
```



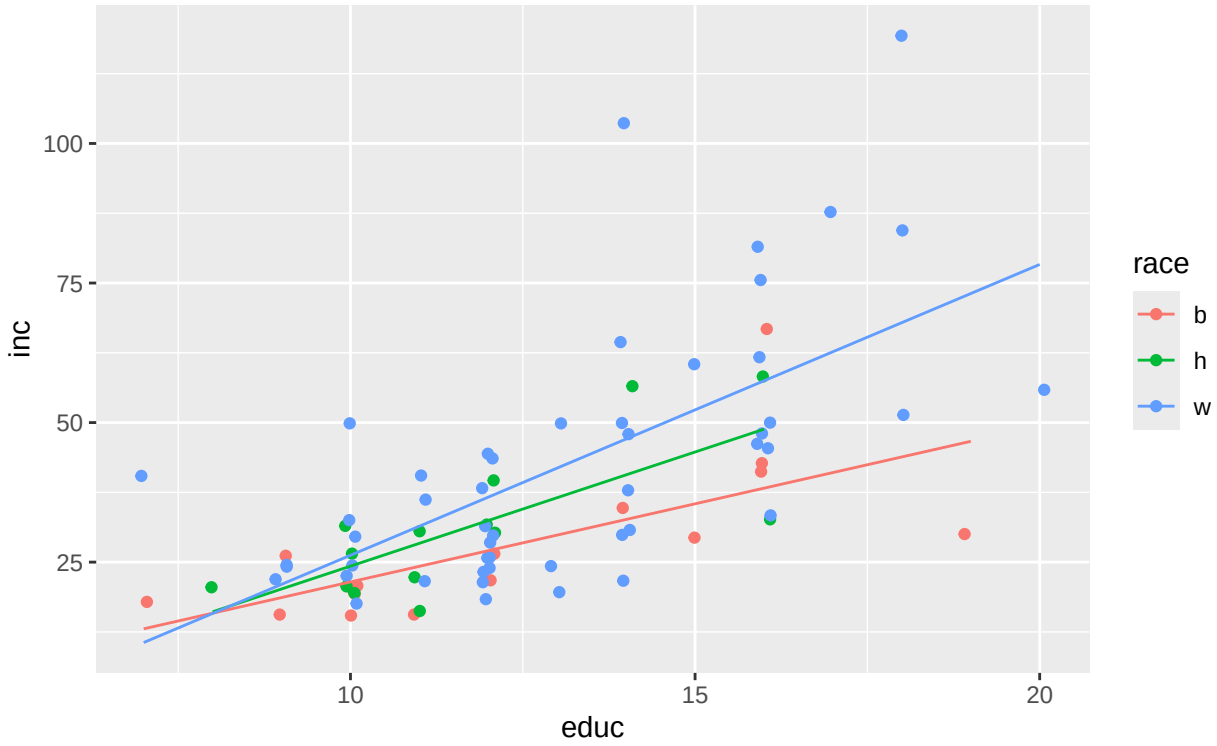

```
mm3 |> tidy()
```

```
## # A tibble: 4 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)   -26.5      8.51      -3.12 2.57e- 3
## 2 raceh           5.94     5.67       1.05 2.98e- 1
## 3 racew          10.9      4.47       2.43 1.74e- 2
## 4 educ            4.43     0.619      7.16 4.42e-10
```

2.4 Model 4 - ANCOVA with interaction

Income grows linearly with years of education but with offset and slope depending on ethnicity

```
mm4 <- lm(inc ~ race * educ, data=dat)
pl0 + geom_line(aes(y=fitted(mm4)))
```




```
mm4 |> tidy()
```

```
## # A tibble: 6 x 5
##   term        estimate std.error statistic p.value
##   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)    -6.54     15.0     -0.436  0.664
## 2 raceh         -10.1      26.5     -0.380  0.705
## 3 racew         -19.3      18.3     -1.06   0.294
## 4 educ            2.80      1.18     2.37   0.0205
## 5 raceh:educ      1.29      2.19     0.588  0.558
## 6 racew:educ      2.41      1.42     1.70   0.0933
```

```
model_list <- list(mm1=mm1, mm2=mm2, mm3=mm3, mm4=mm4)
```

---

3 Measures of fit

3.1 Residual sums-of-squares as measure of fit

One measure of how well a linear model fits to data is the residual sum of squares and derived quantities:

$$RSS = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \quad MSE = \frac{1}{N}RSS, \quad RMSE = \sqrt{MSE}$$

In R there is no built-in function for computing RSS, so we create such a function

```
get_rss <- function(model){
    sum(resid(model)^2) / length(resid(model))
}
```

```
rss_vec <- model_list |> sapply(get_rss)
rss_vec
```

```
##   mm1   mm2   mm3   mm4
## 245.3 380.1 227.1 218.4
```

3.2 Coefficient of determination (R^2) as measure of fit

3.2.1 Text book version of R^2

The total sum of squares is defined as

$$TSS = \sum_i (y_i - \bar{y})^2$$

For any model, $0 \leq RSS \leq TSS$, which leads to the *coefficient of determination* R^2 which always ranges between 0 and 1:

$$R^2 = 1 - RSS/TSS$$

We just write R2. A well fitting model has small RSS so for such models R2 is close to 1.

R has no built-in function to get R2. We create such a function on the fly:

```
get_R2 <- function(model){
    summary(model)$r.squared
}
```

Larger models gives larger R2, but the models can be meaningless.

```
r2_vec <- model_list |> sapply(get_R2)
r2_vec
```

```
##    mm1    mm2    mm3    mm4
## 0.4187 0.0993 0.4620 0.4825
```

3.2.2 Correlation between observed and fitted values

Another idea: Calculate squared correlation between fitted and observed values for a model

```
observed <- function(model){
    fitted(model) + resid(model)
}

cor_fit_obs_sq <- function(model){
  cor(fitted(model), observed(model))^2
}
```

The squared correlation between observed and fitted values are the same as R2:

```
model_list |> sapply(cor_fit_obs_sq)
```

```
##    mm1    mm2    mm3    mm4
## 0.4187 0.0993 0.4620 0.4825
```

3.2.3 Regres fitted values on observed values

Another idea: Make regression model where the fitted values are response and observed values are explanatory. If the model is good the slope should be close to one.

```
regres_fit_obs <- function(model){
  m <- lm(fitted(model) ~ observed(model))
  coef(m)[2] |> unname()
}
```

```
model_list |> sapply(regres_fit_obs)
```

```
##    mm1    mm2    mm3    mm4
## 0.4187 0.0993 0.4620 0.4825
```

These numbers are exactly R2.

3.2.4 To summarize

We have seen three different (and informative) interpretations of R2

1. The proportion of variation in data explained by the model.
2. The squared correlation between observed and fitted values
3. The slope when regressing fitted values (response variable) on observed values (explanatory variable).

```
rbind(model_list |> sapply(get_R2),
      model_list |> sapply(cor_fit_obs_sq),
      model_list |> sapply(regres_fit_obs))
```

```
##         mm1    mm2   mm3    mm4
## [1,] 0.4187 0.0993 0.462 0.4825
## [2,] 0.4187 0.0993 0.462 0.4825
## [3,] 0.4187 0.0993 0.462 0.4825
```

We have seen that R2 always increases when models become more complex (have more parameters).

Hence we can not use R2 for choosing among models; we need a quantity that accounts for model complexity.

3.3 Information criteria as measure of fit (and complexity)

3.3.1 AIC (Akaike's Information Criterion)

Consider a (linear) regression models with p explanatory variables and let N be the number of observations.

We introduce a quantity called the *log-likelihood*:

$$l = -\frac{N}{2}\log(RSS/N) = -\frac{N}{2}\log(MSE)$$

```
model_list  |> lapply(logLik)
```

```
## $mm1
## 'log Lik.' -333.6 (df=3)
##
## $mm2
## 'log Lik.' -351.1 (df=4)
##
## $mm3
## 'log Lik.' -330.5 (df=5)
##
## $mm4
## 'log Lik.' -329 (df=7)
```

We will not dig into where this function comes from, but just note the following:

1. RSS (the residual sum of squares) is **small** for a well fitting model and **large** for a poorly fitting model.
2. The negative sign implies that the log-likelihood for a well fitting model is *larger* than for a poorly fitting model.

The AIC-value for a model is generally defined as

$$AIC = -2l + kp, \text{where k=2 gives genuine AIC}$$

That is for a linear model

$$AIC = N \log(RSS/N)) + kp$$

There is a non-trivial argument why AIC looks as it does. We will not repeat the argument here. Instead we motivate as follows:

The rule is: Select the model for which the AIC value is smallest.

Hence AIC represents a trade-off between fit ($-2l$) and complexity (p).

```
ll_vec <- model_list |> sapply(logLik)
ll_vec
```

```
##    mm1    mm2    mm3    mm4
## -333.6 -351.1 -330.5 -329.0
```

```
model_dim <- function(model){
    length(coef(model)) + 1
}
dim_vec <- model_list |> sapply(model_dim)
```

```
- 2 * ll_vec + 2 * dim_vec
```

```
##   mm1   mm2   mm3   mm4
## 673.2 710.3 671.0 671.9
```

```
aic_vec <- model_list |> sapply(AIC)
aic_vec
```

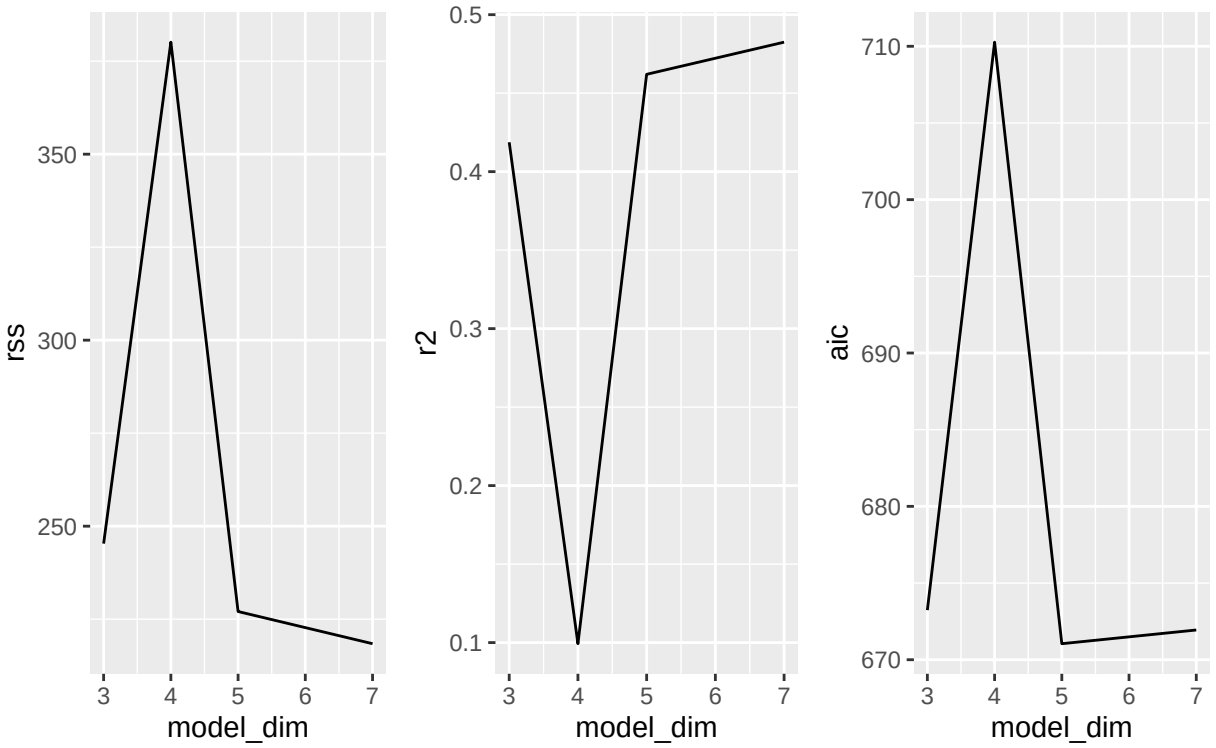
```
##   mm1   mm2   mm3   mm4
## 673.2 710.3 671.0 671.9
```

3.4 Gathering measures of fit

```
ic_df <- data.frame(rss = rss_vec,
                    r2  = r2_vec,
                    aic = aic_vec,
                    model_dim = dim_vec)
ic_df
```

```
##       rss     r2   aic model_dim
## mm1 245.3 0.4187 673.2         3
## mm2 380.1 0.0993 710.3         4
## mm3 227.1 0.4620 671.0         5
## mm4 218.4 0.4825 671.9         7
```

```
p1 <- ic_df |> ggplot(aes(model_dim, rss)) + geom_line()
p2 <- ic_df |> ggplot(aes(model_dim, r2)) + geom_line()
p3 <- ic_df |> ggplot(aes(model_dim, aic)) + geom_line()
cowplot::plot_grid(p1, p2, p3, nrow=1)
```



4 Predictions

4.1 RMSE

For any model and dataset we can compute *root mean square error* as a measure of the predictive ability of a model.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

BUT need to do so on dataset different from what was used for fitting model:

Naive idea:

```
## Non-deterministic
i <- sample(nrow(dat), size=nrow(dat) / 2)
head(i)
```

```
## [1] 70 68 31 27 59 39
```

```
dat_train <- dat[i,]
dat_test  <- dat[-i,]
```

```
mm2_train <- update(mm2, data=dat_train)
mm2 |> coef()
```

```
## (Intercept)       raceh       racew
##       27.75        3.25       14.73
```

```
mm2_train |> coef()
```

```
## (Intercept)       raceh       racew
##      24.500       5.167      14.100
```

```
rmse(mm2, dat)             ## No
```

```
## [1] 19.5
```

```
rmse(mm2_train, dat_train)  ## No
```

```
## [1] 16.03
```

```
rmse(mm2_train, dat_test)   ## Yes
```

```
## [1] 22.95
```

Redo the above on split dataset:

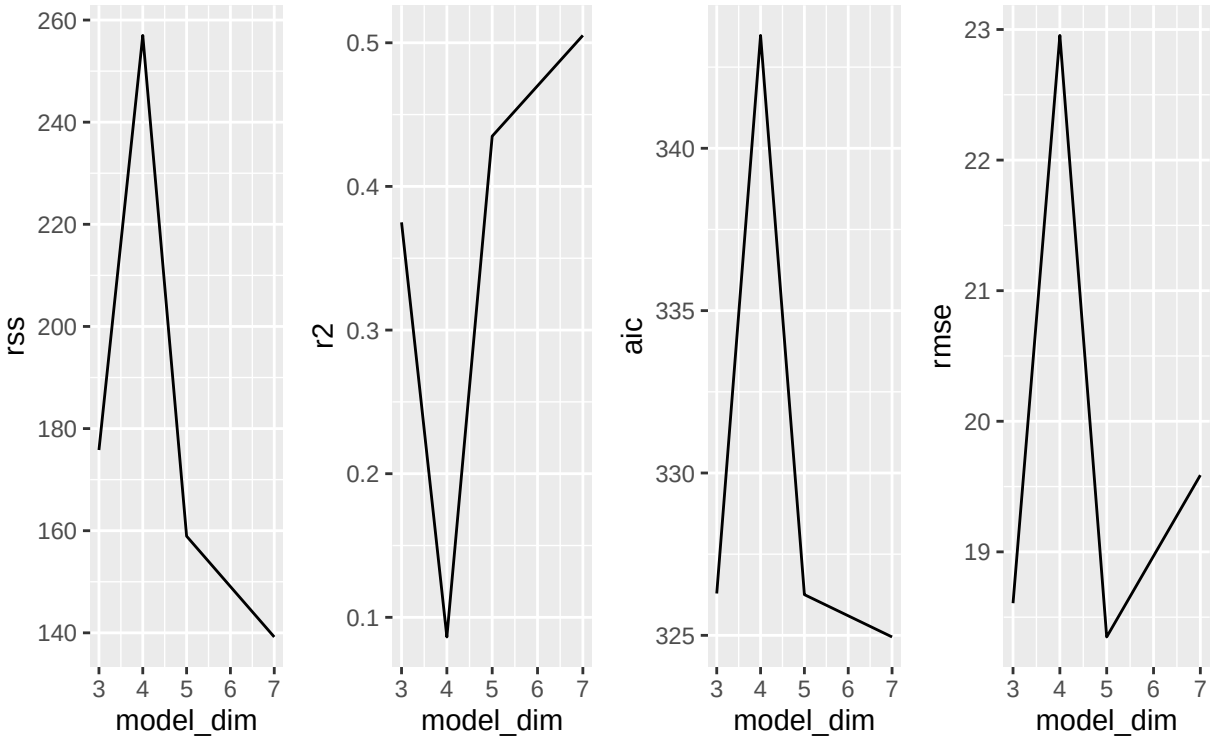
```
ml2 <- model_list |> lapply(function(x){
    update(x, data=dat_train) ## refit models to dat_train
})

zz <- ml2 |> lapply(function(x){
    c(rss=get_rss(x), r2=get_R2(x), aic=AIC(x), model_dim=model_dim(x),
      rmse=rmse(x, dat_test)) ## compute rmse on test_data
})
zz
```

```
## $mm1
##       rss        r2       aic model_dim      rmse
##   175.801     0.375   326.289     3.000    18.607
##
## $mm2
##       rss        r2       aic model_dim      rmse
## 256.98833   0.08635 343.47631   4.00000  22.95460
##
## $mm3
##       rss        r2       aic model_dim      rmse
##   158.931     0.435   326.254     5.000    18.348
##
## $mm4
##       rss        r2       aic model_dim      rmse
##  139.1968    0.5051  324.9506    7.0000   19.5874
```

```
zz <- simplify2array(zz)
zz <- t(zz) ## Transpose
zz <- zz |> as.data.frame()
zz
```

```
##       rss      r2   aic model_dim  rmse
## mm1 175.8 0.37499 326.3         3 18.61
## mm2 257.0 0.08635 343.5         4 22.95
## mm3 158.9 0.43497 326.3         5 18.35
## mm4 139.2 0.50513 325.0         7 19.59
```



---

5 Cross validation

Evaluating predictive performance on the same dataset as the model was fitted to can be misleading (give too optimistic results, e.g. in terms to too small rmse value).

This suggests to split data (randomly) into two groups: A training dataset for fitting models and a validation dataset for evaluating predictive performance.

A systematic approach to this is called cross validation:

1. Partition data randomly into, say 5 groups (called folds) numbered 1,2,3,4,5. Form 5 new training datasets from these folds. These consists of (1) fold 1,2,3,4; (2) fold 1,2,3,5; (3) fold 1,2,4,5; (4) fold 1,3,4,5 and (4) fold 2,3,4,5.
2. Fit a model to each new training set and evaluate predictive performance on the remaining data.

5.1 K-fold cross validation

Partition dataset K times into two groups: A test dataset with K rows and a training dataset with N-K rows. Notice: Non-deterministic partitioning, repeat and a different partitioning is obtained.

```
dat0 <- dat[1:9,]
K <- 3
cv1 <- modelr::crossv_kfold(dat0, k = K, id = ".id")
cv1
```

```
## # A tibble: 3 x 3
##   train               test                .id
##   <named list>        <named list>        <chr>
## 1 <resample [6 x 3]> <resample [3 x 3]> 1
## 2 <resample [6 x 3]> <resample [3 x 3]> 2
```

```
## 3 <resample [6 x 3]> <resample [3 x 3]> 3
```

```
dat0
```

```
##   inc educ race
## 1  16   10    b
## 2  18    7    b
## 3  26    9    b
## 4  16   11    b
## 5  34   14    b
## 6  22   12    b
## 7  42   16    b
## 8  42   16    b
## 9  16    9    b
```

```
cv1$train[[1]] |> as.data.frame()
```

```
##   inc educ race
## 1  16   10    b
## 2  18    7    b
## 3  26    9    b
## 4  16   11    b
## 5  34   14    b
## 9  16    9    b
```

```
cv1$test[[1]] |> as.data.frame()
```

```
##   inc educ race
## 6  22   12    b
## 7  42   16    b
## 8  42   16    b
```

To make partitioning deterministic (across different R sessions), use

```
set.seed(2024) ## Some number
cv2 <- crossv_kfold(dat0, k = K, id = ".id")
cv2
```

```
## # A tibble: 3 x 3
##   train              test               .id
##   <named list>       <named list>       <chr>
## 1 <resample [6 x 3]> <resample [3 x 3]> 1
## 2 <resample [6 x 3]> <resample [3 x 3]> 2
## 3 <resample [6 x 3]> <resample [3 x 3]> 3
```

5.2 Leave-one-out (LOO) cross validation

Notice: Leave-one-out is deterministic:

```
cv3 <- crossv_loo(dat0, id = ".id")
cv3
```

```
## # A tibble: 9 x 3
##   train              test                .id
##   <named list>       <named list>       <int>
## 1 <resample [8 x 3]> <resample [1 x 3]>     1
## 2 <resample [8 x 3]> <resample [1 x 3]>     2
## 3 <resample [8 x 3]> <resample [1 x 3]>     3
## 4 <resample [8 x 3]> <resample [1 x 3]>     4
## 5 <resample [8 x 3]> <resample [1 x 3]>     5
## 6 <resample [8 x 3]> <resample [1 x 3]>     6
## 7 <resample [8 x 3]> <resample [1 x 3]>     7
## 8 <resample [8 x 3]> <resample [1 x 3]>     8
## 9 <resample [8 x 3]> <resample [1 x 3]>     9
```

6 Cross validation in practice

6.1 Cross validation using `cv.glm`

There are several functions/packages for doing cross validation in R.

One function is `cv.glm` from the `boot` package.

The model must be a `glm` model rather than an `lm` model. Either we invoke the `glm` function the same way as we invoked `lm` or we can coerce an `lm` model to a `glm` model using `glm(an_lm_model)`.

```
rmse_fun <- function(y, yhat) sqrt(mean((y - yhat)^2))
set.seed(2024)
cv.glm(dat, glm(mm3), cost=rmse_fun, K=10)$delta[1]
```

```
## [1] 14.79
```

```
set.seed(2024)
cv.glm(dat, glm(mm1), cost=rmse_fun, K=10)$delta[1]
```

```
## [1] 14.97
```

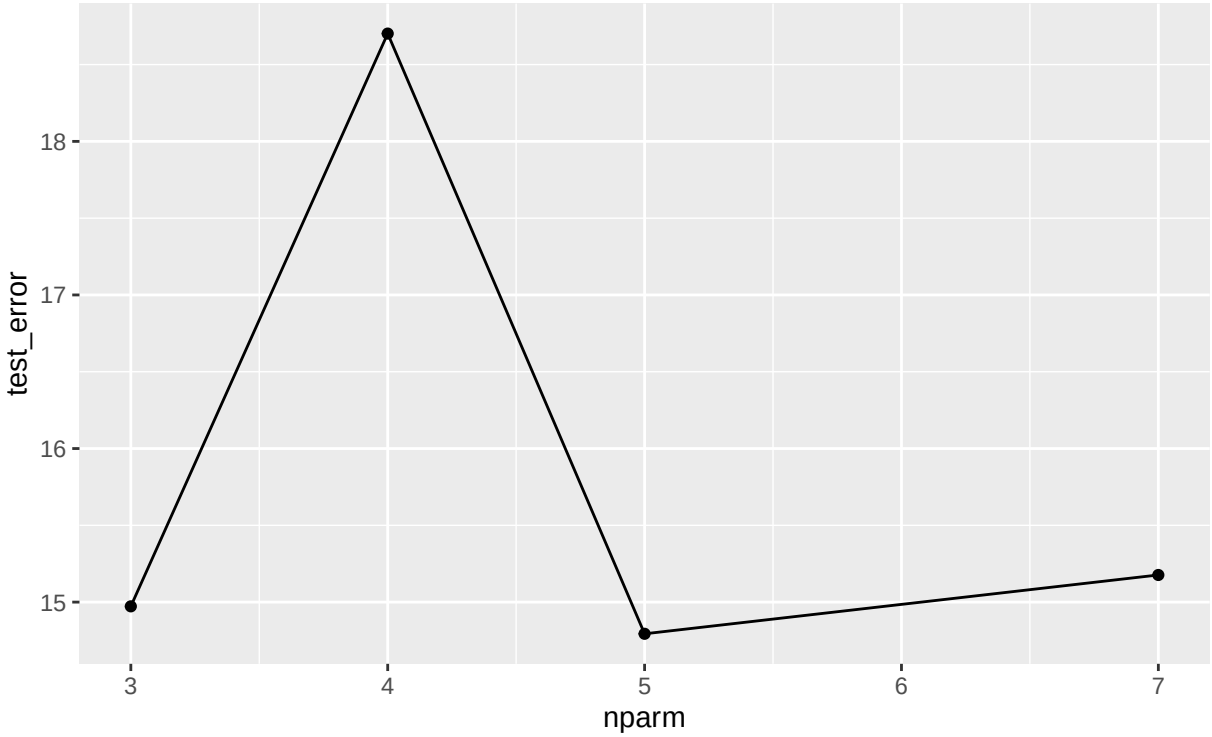
We can do so for all models using this code chunk:

```
rmse_vec <- model_list |>
    sapply(function(x){
        set.seed(2024)
        cv.glm(dat, glm(x), cost=rmse_fun, K=10)$delta[1]
    })
rmse_vec
```

```
##   mm1   mm2   mm3   mm4
## 14.97 18.70 14.79 15.18
```

```
rmse_df <- data.frame(
    test_error = rmse_vec,
    nparm = dim_vec
)

rmse_df |> ggplot(aes(nparm, test_error)) + geom_point() + geom_line()
```



6.2 Cross validation in practice - details - optional*

We show how to do cross validation in a form closer to code. Could come in handy one day.

Strategy:

1. Pick a model and fit this model to each training dataset.
2. Predict each test dataset and compute rmse.
3. Compute the average rmse.

```
set.seed(2024) ## Some number
cv1 <- crossv_kfold(dat, k = K, id = ".id")
cv1
```

```
## # A tibble: 3 x 3
##   train               test               .id
##   <named list>        <named list>       <chr>
## 1 <resample [53 x 3]> <resample [27 x 3]> 1
## 2 <resample [53 x 3]> <resample [27 x 3]> 2
## 3 <resample [54 x 3]> <resample [26 x 3]> 3
```

```
train_data <- cv1[["train"]]
test_data <- cv1[["test"]]
this_model <- mm1
```

Refit model to each training dataset:

```
model_fits <-
    train_data |>
    lapply(function(dat){
        update(this_model, data=dat)
    })

model_fits |> map(coef)
```

```
## $`1`
## (Intercept)        educ
##     -24.547       4.775
##
## $`2`
## (Intercept)        educ
##      -26.10        5.05
##
## $`3`
## (Intercept)        educ
##     -15.641       4.269
```

Predict each test dataset and compute rmse:

```
model_fits |> length()
```

```
## [1] 3
```

```
test_data |> length()
```

```
## [1] 3
```

```
rmse_vec <- mapply(function(model, data){
    rmse(model, data)
}, model_fits, test_data
)
rmse_vec
```

```
##     1     2     3
## 15.00 15.56 17.32
```

```
rmse_vec |> mean()
```

```
## [1] 15.96
```

We write a function that will do so for all models:

```
refit_to_train <- function(model, data_train_list){
  data_train_list |> lapply(function(dat){
      update(model, data=dat)
})
}
```

```
compute_rmse <- function(model_fits, cv_data){
  mapply(function(model, data){
      rmse(model, data)
  }, model_fits, cv_data
  )
}

model_fits <- refit_to_train(this_model, train_data)
compute_rmse(model_fits, test_data) |> mean()
```

```
## [1] 15.96
```

Almost done; just have to loop over all models:

```
get_rmse <- function(model_list, train_data, test_data){
    model_list |>
        sapply(function(this_model){
            model_fits <- refit_to_train(this_model, train_data)
            compute_rmse(model_fits, test_data) |> mean()
        })
}

get_rmse(model_list, train_data, test_data)
```

```
##   mm1   mm2   mm3   mm4
## 15.96 20.24 15.66 16.90
```

```
get_rmse <- function(model_list, cv_data){
    model_list |>
        sapply(function(this_model){
            model_fits <- refit_to_train(this_model, cv_data[["train"]])
            compute_rmse(model_fits, cv_data[["test"]]) |> mean()
        })
}

get_rmse(model_list, cv1)
```

```
##   mm1   mm2   mm3   mm4
## 15.96 20.24 15.66 16.90
```