Applied Genetic Evaluation - Solution 1

Peter von Rohr

2019-04-15

Problem 1: Model Selection

We assume that we have a dataset for the response variable `carcass weight` (CW) and for some predictor variables

- sex (`sex`)
- slaughterhouse (`slh`)
- herd (`hrd`)
- age at slaughter (`age`)
- day of month when animal was slaughtered (`day`) and
- humidity (`hum`)

Use a fixed linear effects model and determine which of the predictor variables are important for the response.

The data is available from `data_bp_w09.csv`.

Hint

- Use the function `lm` in R to fit the fixed linear effects model
- Use Mallow C_p statistic and the adjusted coefficient of determination R^2_{adj} as model selection criteria
- Use the backward model selection approach

Solution

On possibility to solve this problem is searching on Google for a R-package that does model selection. With more than 10^4 R-packages available on CRAN (Comprehensive R Archive Network), it is very likely that there is a package available that does exactly what we need.

For our model selection problem, there is a package called `olsrr` which uses different criteria for model selection one of which is Mallows C_p statistic. Then there is the function `MASS::stepAIC()` which does step-wise model selection using the Akaike Information Criterion. As a third possibility, we can come up with our own solution.

As preparatory step we have to first read the data from the file

```
s_data_file <- "https://charlotte-ngs.github.io/GELASMSS2019/ex/w09/data_bp_w09.csv"
tbl_modsel <- readr::read_csv2(s_data_file)
```

```
## Using ',' as decimal and '.' as grouping mark. Use read_delim() for more control.
```

```
## Parsed with column specification:
## cols(
##   Id = col_double(),
##   sex = col_double(),
##   slh = col_double(),
##   hrd = col_double(),
##   age = col_double(),
##   cw = col_double(),
##   day = col_double(),
```

```
##   hum = col_double()
## )
```

Before we can do any model fits, we have to convert all fixed effects into `factors`. Fixed effects will be

- `sex`
- `slh`
- `hrd`

These must be converted into factors. All other predictors are fit as covariables and can stay as numeric types.

```
tbl_modsel$sex <- as.factor(tbl_modsel$sex)
tbl_modsel$slh <- as.factor(tbl_modsel$slh)
tbl_modsel$hrd <- as.factor(tbl_modsel$hrd)
```

Package `olsrr`

According to https://cran.r-project.org/web/packages/olsrr/vignettes/variable_selection.html with `olsrr` we can select the best model according to

```
lm_full_model <- lm(cw ~ sex + slh + hrd + age + day + hum, data = tbl_modsel)
olsrr::ols_step_best_subset(lm_full_model)
```

```
##        Best Subsets Regression
## -------------------------------------
## Model Index    Predictors
## -------------------------------------
##      1         hrd
##      2         slh hrd
##      3         sex slh hrd
##      4         sex slh hrd age
##      5         sex slh hrd age hum
##      6         sex slh hrd age day hum
## -------------------------------------
##
##                                                    Subsets Regression Summary
## ---------------------------------------------------------------------------------------------------------------------------------------------------
##                        Adj.        Pred
## Model    R-Square    R-Square    R-Square       C(p)          AIC           SBIC          SBC          MSEP        FPE        HSP       APC
## ---------------------------------------------------------------------------------------------------------------------------------------------------
##   1        0.8412      0.8411      0.8409    14234.1586    43459.9264    28338.2611    43499.4074    204.9309    204.9309    0.0385    0.1589
##   2        0.9177      0.9176      0.9175     4812.2559    39962.1327    24839.1408    40014.7741    106.2318    106.2317    0.0200    0.0824
##   3        0.9464      0.9464      0.9463     1278.7853    37678.6724    22557.3614    37737.8939     69.1864     69.1864    0.0130    0.0536
##   4        0.9568      0.9568      0.9567        0.8401    36531.8580    21412.1798    36597.6597     55.7814     55.7814    0.0105    0.0432
##   5        0.9568      0.9568      0.9567        1.2941    36532.3090    21412.6378    36604.6909     55.7862     55.7861    0.0105    0.0432
##   6        0.9569      0.9568      0.9567        3.0000    36534.0143    21414.3479    36612.9764     55.8041     55.8040    0.0105    0.0433
## ---------------------------------------------------------------------------------------------------------------------------------------------------
## AIC: Akaike Information Criteria
##  SBIC: Sawa's Bayesian Information Criteria
##  SBC: Schwarz Bayesian Criteria
##  MSEP: Estimated error of prediction, assuming multivariate normality
##  FPE: Final Prediction Error
##  HSP: Hocking's Sp
##  APC: Amemiya Prediction Criteria
```

This shows that based on the C_p criterion model number four would be the best model for our data.

Function `MASS::stepAIC()`

The model selection can also be done using `stepAIC()` from the `MASS` package.

```
MASS::stepAIC(lm_full_model)
```

```
## Start:  AIC=21420.32
## cw ~ sex + slh + hrd + age + day + hum
##
##        Df Sum of Sq     RSS   AIC
```

```
## - day   1        16  296169 21419
## - hum   1        87  296240 21420
## <none>               296153 21420
## - age   1     71349  367502 22568
## - sex   1    191454  487607 24074
## - slh   2    511671  807824 26760
## - hrd   4   5835105 6131258 37549
##
## Step:  AIC=21418.61
## cw ~ sex + slh + hrd + age + hum
##
##        Df Sum of Sq     RSS   AIC
## - hum   1        86  296256 21418
## <none>               296169 21419
## - age   1     71363  367532 22566
## - sex   1    191473  487643 24072
## - slh   2    511678  807847 26758
## - hrd   4   5835440 6131609 37547
##
## Step:  AIC=21418.16
## cw ~ sex + slh + hrd + age
##
##        Df Sum of Sq     RSS   AIC
## <none>               296256 21418
## - age   1     71332  367588 22565
## - sex   1    191461  487716 24071
## - slh   2    511719  807974 26757
## - hrd   4   5835356 6131612 37545
```

```
##
## Call:
## lm(formula = cw ~ sex + slh + hrd + age, data = tbl_modsel)
##
## Coefficients:
## (Intercept)         sex2         slh2         slh3         hrd2
##     11.6987     -74.2607      22.2570       3.6342      88.0069
##        hrd3         hrd4         hrd5          age
##      8.7056      58.7044      19.8066       0.6469
```

The final result of `MASS::stepAIC()` is the same as with `olsrr::ols_step_best_subset()` which gives good confidence that the model

```
cw ~ sex + slh + hrd + age
```

is the best model for our data. Although the computed values for `AIC` are not the same. We would need to analyse the computations done in more detail.

Backward Selection Using Our Own Functions

Step 1:

The backward model selection approach starts with the full model as the current model.

```
fo_full_model <- cw ~ sex + slh + hrd + age + day + hum
lm_full_model <- lm(fo_full_model, data = tbl_modsel)
summary(lm_full_model)
```

```
##
## Call:
## lm(formula = fo_full_model, data = tbl_modsel)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -27.1543  -5.1451  -0.0484   4.9538  26.3746
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  11.777602   7.382489   1.595    0.111
## sex2        -74.279088   1.267308 -58.612   <2e-16 ***
## slh2         22.258398   0.250964  88.691   <2e-16 ***
## slh3          3.635946   0.253017  14.370   <2e-16 ***
## hrd2         88.007266   0.323588 271.973   <2e-16 ***
## hrd3          8.700083   0.323737  26.874   <2e-16 ***
## hrd4         58.701466   0.321387 182.651   <2e-16 ***
## hrd5         19.810008   0.320882  61.736   <2e-16 ***
## age           0.647028   0.018083  35.781   <2e-16 ***
## day          -0.006396   0.011794  -0.542    0.588
## hum           0.126979   0.101433   1.252    0.211
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.465 on 5314 degrees of freedom
## Multiple R-squared:  0.9569, Adjusted R-squared:  0.9568
## F-statistic: 1.178e+04 on 10 and 5314 DF,  p-value: < 2.2e-16
```

From the full model we have to extract the estimate of the error variance $\hat{\sigma}^2$.

```
vec_res_full_model <- residuals(lm_full_model)
n_sse_full_model <- crossprod(vec_res_full_model)
n_hatsigma2_full_model <- n_sse_full_model / lm_full_model$df.residual
```

Step 2:

In step 2 we eliminate the predictor variable that increases the residual sum of squares the least. This is done using the function `get_reduced_model()`. In this function the model formula is converted into a vector of predictors (`vec_formula_label`). From this vector each predictor is excluded once and from the resulting reduced models the residual sums of squares are computed. As a result the model increasing the residual sum of squares the least is returned.

```
#' Given A Model Formula, Find Model Reduced By One Predictor
#'
#' From the given model formula pfo_model, the predictor that
#' increases the residual sum of squares the least is eliminated
#' resulting in the reduced model.
#'
#' @param pfo_model current model formula
#' @param ptbl_data tibble containing data
get_reduced_model <- function(pfo_model, ptbl_data){
  # convert pfo_model formula into a character vector of three elements
  # (1) "~", (2) response, (3) righthand-side of formula
  vec_char_formula_current <- as.character(pfo_model)
  # find vector of labels corresponding to predictors in pfo_model
  vec_formula_label <- labels(terms(pfo_model))
```

```
  n_nr_term <- length(vec_formula_label)
  # model that does not have predictors cannot be reduced anymore
  if (n_nr_term == 0){
    stop("[ERROR -- get_reduced_model] Cannot reduce model, because number of terms is: ",
         n_nr_term, "\n")
  }
  # Initialise tibble for reduced model
  tbl_backstep_result <- NULL
  # for a model with just one predictor, we construct an intercept model
  if (n_nr_term == 1){
    s_formula_red <- convert_formula_to_string(ps_response = vec_char_formula_current[2],
                                               pvec_predictor = c("1"))
    lm_red <- lm(as.formula(s_formula_red), data = ptbl_data)
    vec_res <- residuals(lm_red)
    n_sse_red <- crossprod(vec_res)
    tbl_backstep_result <- tibble::tibble( s_formula_min = as.vector(s_formula_red),
                                           n_sse_min     = as.vector(n_sse_red) )
  } else {
    # loop over all predictors and find the one that increases sse the least
    for (lidx in seq_along(vec_formula_label)){
      s_formula_red <- convert_formula_to_string(ps_response = vec_char_formula_current[2],
                                                 pvec_predictor = vec_formula_label[-lidx])
      lm_red <- lm(as.formula(s_formula_red), data = ptbl_data)
      vec_res <- residuals(lm_red)
      n_sse_red <- crossprod(vec_res)
      if (is.null(tbl_backstep_result) || n_sse_red < tbl_backstep_result$n_sse_min){
        tbl_backstep_result <- tibble::tibble( s_formula_min = as.vector(s_formula_red),
                                               n_sse_min     = as.vector(n_sse_red) )
      }
    }
  }
  # return result
  return(tbl_backstep_result)
}
```

The function `get_reduced_model()` can be used to eliminate the one predictor that increases the residual sum of squares the least compared to the current model.

```
get_reduced_model(pfo_model = fo_full_model, ptbl_data = tbl_modsel)
```

```
## # A tibble: 1 x 2
##   s_formula_min                   n_sse_min
##   <chr>                               <dbl>
## 1 cw ~ sex + slh + hrd + age + hum   296169.
```

The result of calling `get_reduced_model()` with the full model and the dataset, gives a tibble with one row and two columns. The first column contains the formula of the reduced model and the second model contains the residual sum of squares for the reduced model.

Inside of the function `get_reduced_model()`, the helper function `convert_formula_to_string()` is used to convert a formula into a string. This helper function is shown below.

```
convert_formula_to_string <- function(pfo_model = NULL,
                                      ps_response = NULL,
                                      pvec_predictor = NULL){
  # if a formula is specified in pfo_model, then paste together the components
```

```
  if (!is.null(pfo_model)){
    vec_formula_char <- as.character(pfo_model)
    return(paste0(vec_formula_char[c(2,1,3)], collapse = " "))
  }
  # if response and predictors are given separately, paste those together
  #  start with pvec_predictor
  n_nr_predictor <- length(pvec_predictor)
  if (n_nr_predictor > 1){
    s_pred <- paste0(pvec_predictor, collapse = " + ")
  } else if (n_nr_predictor == 1) {
      s_pred <- pvec_predictor[1]
  } else {
      s_pred <- "1"
  }
  # add response and return result
  return(paste0(ps_response, " ~ ", s_pred))
}
```

Step 3:

The step 2 is repeated until all predictors are eliminated. The result of this is a sequence of models. This sequence is produced by the function `construct_model_sequences()`.

```
construct_model_sequences <- function(pfo_full_model, ptbl_data){
  # initialise a tibble that will hold the sequence of models
  tbl_model_sequence_result <- NULL
  # start with a fit of the full model and compute estimate of error variance
  fo_current_model <- pfo_full_model
  lm_current_model <- lm(fo_current_model, data = ptbl_data)
  n_sigmahat2_full_model <- crossprod(residuals(lm_current_model)) / lm_current_model$df.residual
  # extract lables of model terms for full model corresponding
  #  to all predictors in a character vector
  vec_label_current_model <- labels(terms(lm_current_model))
  n_nr_pred_current_model <- length(vec_label_current_model)
  # loop until there are any predictors in the model
  while (n_nr_pred_current_model > 0){
    # compute C_p for current model and add it to result tibble
    n_cp_current_model <- compute_cp(pfo_model               = fo_current_model,
                                     pn_sigmahat2_full_model = n_sigmahat2_full_model,
                                     ptbl_data               = ptbl_data)

    # convert model into a character vector
    vec_fo_current_model <- as.character(fo_current_model)
    # put information about current model into a tibble
    s_cur_fo <- convert_formula_to_string(ps_response    = vec_fo_current_model[2],
                                          pvec_predictor = vec_label_current_model)
    tbl_cur_model <- tibble::tibble(model = as.vector(s_cur_fo),
                                    cp    = as.vector(n_cp_current_model))
    # add current model to result structure
    if (is.null(tbl_model_sequence_result)){
      tbl_model_sequence_result <- tbl_cur_model
    } else {
      tbl_model_sequence_result <- dplyr::bind_rows(tbl_model_sequence_result, tbl_cur_model)
    }
    # reduce current model by one predictor
```

```
    tbl_red_model <- get_reduced_model(pfo_model    = fo_current_model,
                                       ptbl_data    = ptbl_data)
    # update
    fo_current_model <- as.formula(tbl_red_model$s_formula_min)
    lm_current_model <- lm(fo_current_model, data = ptbl_data)
    vec_label_current_model <- labels(terms(lm_current_model))
    n_nr_pred_current_model <- length(vec_label_current_model)
  }
  # add the intercept model
  # compute C_p for current model and add it to result tibble
  n_cp_current_model <- compute_cp(pfo_model               = fo_current_model,
                                   pn_sigmahat2_full_model = n_sigmahat2_full_model,
                                   ptbl_data               = ptbl_data)

  # convert model into a character vector
  vec_fo_current_model <- as.character(fo_current_model)
  # put information about current model into a tibble
  s_cur_fo <- convert_formula_to_string(ps_response    = vec_fo_current_model[2],
                                        pvec_predictor = vec_label_current_model)
  tbl_cur_model <- tibble::tibble(model = as.vector(s_cur_fo),
                                  cp    = as.vector(n_cp_current_model))
  # add current model to result structure
  if (is.null(tbl_model_sequence_result)){
    tbl_model_sequence_result <- tbl_cur_model
  } else {
    tbl_model_sequence_result <- dplyr::bind_rows(tbl_model_sequence_result, tbl_cur_model)
  }

  return(tbl_model_sequence_result)
}
```

The model sequence is constructed with the following call.

```
(tbl_model_seq <- construct_model_sequences(pfo_full_model = fo_full_model,
                                            ptbl_data = tbl_modsel))
```

```
## # A tibble: 7 x 2
##   model                                      cp
##   <chr>                                   <dbl>
## 1 cw ~ sex + slh + hrd + age + day + hum     11
## 2 cw ~ sex + slh + hrd + age + hum            9.29
## 3 cw ~ sex + slh + hrd + age                  8.84
## 4 cw ~ sex + slh + hrd                     1287.
## 5 cw ~ slh + hrd                           4820.
## 6 cw ~ hrd                                14240.
## 7 cw ~ 1                                 117834.
```

The helper function to compute the Mallow C_p statistic used in `construct_model_sequences()` is shown below.

```
#' Compute Mallows C_p Statistic for model pfo_model
compute_cp <- function(pfo_model, pn_sigmahat2_full_model, ptbl_data){
  # fit the model given by pfo_model to data in ptbl_data
  lm_model <- lm(pfo_model, data = ptbl_data)
  # get residuals
```

```
  vec_res <- residuals(lm_model)
  # get number of observations
  n_nr_obs <- length(vec_res)
  # get residual sum of squares
  n_sse <- crossprod(vec_res)
  # get number of fixed effects
  n_cardm <- length(coefficients(lm_model))
  # compute cp
  n_cp_result <- n_sse / pn_sigmahat2_full_model - n_nr_obs + 2 * n_cardm
  # return result
  return(n_cp_result)
}
```

Final Remark

Also with the third solution, the same model is found to be the best model when looking at the minimal C_p value. Because this data is generated, we know the truth and indeed the predictors `day` and `hum` are just random numbers and were not used in the generation of the values of the response variable.