# Lab_06T

36-290 – Statistical Research Methodology

Week 6 Tuesday – Fall 2021

# Preliminaries

## Goal

The goal of this lab is to code and interpret a best subset selection analysis.

## Data

We'll begin by importing galaxy data:

```
file.path = "https://raw.githubusercontent.com/pefreeman/36-290/master/EXAMPLE_DATASETS/PHOTO_MORPH/photo_morph.Rda
ta"
load(url(file.path))
df = data.frame(predictors,"y"=response)
rm(file.path,predictors,response)
```

If everything loaded correctly, you should see the variable `df` in your global environment, with 17 measurements each for 3,419 galaxies. The column `y` is a response vector with 3,419 spectroscopically determined redshifts (spectroscopic = "effectively no error in the redshift determination"). I combine the predictors and response into one data frame because this is what the `bestglm()` function, the one you will use below, expects as an input.

To see a full description of the dataset, click here (https://github.com/pefreeman/36-290/tree/master/EXAMPLE_DATASETS/PHOTO_MORPH). The short version of the description is that of the 16 predictor variables, four represent brightness (one magnitude, three colors), and 12 are morphological statistics, i.e., statistics that encode the galaxy's appearance. The question at hand is: which, if any, of the morphological statistics are informative in predicting redshift?

# Questions

To answer the questions below, it will help you to refer to Sections 6.1 of ISLR; it might also help you to refer to your previous lab work (and, as always, to Google).

*Note, however, that you are not going to use the leops package as suggested by ISLR, but rather the bestglm package, which is applicable in both the linear regression and logistic regression regimes.*

# Question 1

Split your data into training and test datasets (keeping in mind that cross-validation is "better" but not necessary in a lab setting).

```
names(df)
```

```
##  [1] "mag.i"  "col.Vi" "col.iJ" "col.JH" "V.G"    "V.M20"  "V.C"    "V.size"
##  [9] "J.G"    "J.M20"  "J.C"    "J.size" "H.G"    "H.M20"  "H.C"    "H.size"
## [17] "y"
```

```
#3,419

set.seed(100)
s = sample(nrow(df), 0.7*nrow(df))
df.train = df[s, ]
df.test = df[-s, ]
```

# Question 2

Apply linear regression to your training data, and then compute the mean-squared error using your test data.

```
linreg = lm(y~., data=df.train)
summary(linreg)
```

```
##
## Call:
## lm(formula = y ~ ., data = df.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.5315 -0.2768 -0.0287  0.2408  4.0162
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.3939941  0.1146567  12.158  < 2e-16 ***
## mag.i        0.4530335  0.0120462  37.608  < 2e-16 ***
## col.Vi       0.0069624  0.0312710   0.223   0.8238
## col.iJ      -0.7453164  0.0371310 -20.073  < 2e-16 ***
## col.JH       0.4575361  0.0893389   5.121 3.28e-07 ***
## V.G          0.0032457  0.2058089   0.016   0.9874
## V.M20        0.0458102  0.0597383   0.767   0.4432
## V.C          0.0185445  0.0356249   0.521   0.6027
## V.size      -0.0539337  0.0603745  -0.893   0.3718
## J.G          3.6506189  0.6325719   5.771 8.90e-09 ***
## J.M20        0.0003958  0.1341286   0.003   0.9976
## J.C          0.0952123  0.0672916   1.415   0.1572
## J.size      -1.2670987  0.2004796  -6.320 3.11e-10 ***
## H.G         -3.0735311  0.6366381  -4.828 1.47e-06 ***
## H.M20        0.2977807  0.1329093   2.240   0.0252 *
## H.C          0.1651146  0.0650521   2.538   0.0112 *
## H.size       1.7611946  0.1955678   9.006  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5274 on 2376 degrees of freedom
## Multiple R-squared:  0.6193, Adjusted R-squared:  0.6167
## F-statistic: 241.6 on 16 and 2376 DF,  p-value: < 2.2e-16
```

```
test.pred = predict(linreg,newdata=df.test)

predmse = mean((df.test$y - test.pred)^2)
predmse
```

```
## [1] 0.3054208
```

# Question 3

Install the `bestglm` package, if you do not have it installed already. Then load that library and use the function `bestglm()` to perform best subset selection on the training data. Do both AIC and BIC…and for each, display the best model. How many predictor variables are retained in the best models? (Don't include the intercepts.) Do the relative numbers of variables abide by your expectations? Is one model a subset of the other? (Hint: see the documentation for `bestglm()` and look at the part under "Value"…this describes the `R` object that `bestglm()` returns. The best model is included within that object.)

```
library(bestglm)
```

```
## Loading required package: leaps
```

```
bg.outBIC = bestglm(df.train,family=gaussian, IC="BIC")

bg.outBIC$BestModel
```

```
##
## Call:
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),
##     drop = FALSE], y = y))
##
## Coefficients:
## (Intercept)        mag.i       col.iJ       col.JH          J.G       J.size
##      1.3934       0.4522      -0.7410       0.4551       3.8603      -1.3295
##         H.G        H.M20          H.C       H.size
##     -3.1827       0.2964       0.2402       1.7821
```

```
bg.outAIC= bestglm(df.train,family=gaussian, IC="AIC")
bg.outAIC$BestModel
```

```
##
## Call:
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),
##     drop = FALSE], y = y))
##
## Coefficients:
## (Intercept)        mag.i       col.iJ       col.JH          J.G          J.C
##     1.39035      0.45437     -0.74382      0.44669      3.68935      0.09251
##      J.size          H.G        H.M20          H.C       H.size
##    -1.30111     -3.10045      0.31177      0.17098      1.76041
```

```
#"mag.i"  "col.Vi" "col.iJ" "col.JH" "V.G"    "V.M20"  "V.C"    "V.size" "J.G"    "J.M20"  "J.C"    "J.size"
#"H.G"  "H.M20"  "H.C"    "H.size" "y"
```

```
For BIC:

9 predictor variables retained


For AIC:

10 predictor variables retained


yes it is what I expected because using BIC, I know that each variable selected is important but there is a chance
that other important variables might have been left out of the final list whereas using AIC I know that it includes
all important variables but there is a chance of including unimportant ones as well. It makes sense that AIC has mo
re predictor variables retained.


from R documentation:

Value
A list with class attribute 'bestglm' and named components:

BestModel
An lm-object representing the best fitted regression.

Best model is a subset of bestglm
```

# Question 4

The output of `bestglm()` contains, as you saw above, `BestModel`. According to the documentation for `bestglm()`, `BestModel` is "[a]n lm-object representing the best fitted algorithm." That means you can pass it to `predict()` in order to generate predicted response values (where the response is in the `y` column of your data frames). Given this information: generate mean-squared error values for the BIC- and AIC-selected models. Are these values larger or smaller than the value you got for linear regression?

```
#MSE for BIC and AIC model
resp.predBIC = predict(bg.outBIC$BestModel,newdata=df.test)
BICMSE= mean((df.test$y - resp.predBIC)^2)
BICMSE
```

```
## [1] 0.3064319
```

```
resp.predAIC = predict(bg.outAIC$BestModel,newdata=df.test)
AICMSE= mean((df.test$y - resp.predAIC)^2)
AICMSE
```

```
## [1] 0.3060815
```

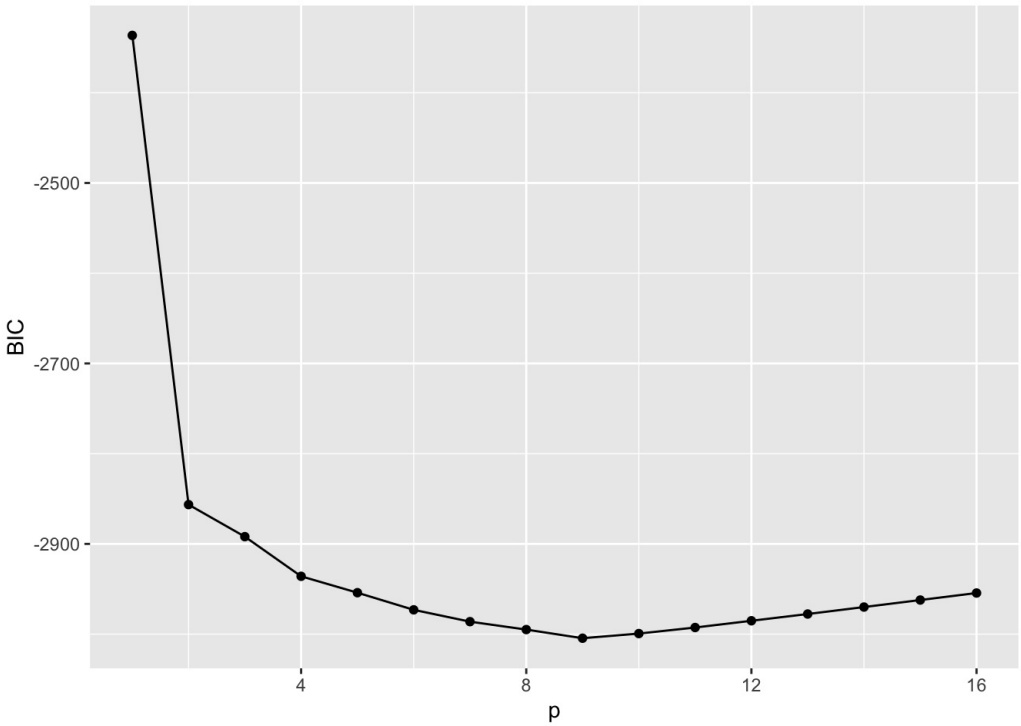
```
MSE value from linear regression: 0.3054208
MSE value using predicted response values for BIC model: 0.3064319, larger than value from lin reg
MSE value using predicted response values for AIC model: 0.3060815, larger than value from lin reg
```

# Question 5

In Q3, I asked you to output information about the best models for AIC and BIC. Here, choose one of those criteria, and extract the values for that criterion for each $p$ value, where $p$ is the number of retained predictor variables. (Look under `Value` on the `bestglm()` documentation page to see which component of the outputted `R` object contains the information you need.) Then use `ggplot()` to plot the criterion values

versus $p$. Zoom in (using `ylim()`) to decrease the dynamic range of the plot and to see more clearly how BIC changes as a function of $p$ near the minimum value. (So as to not hardcode numbers, use, e.g., `min(...)` as the first argument to `ylim`, where you'd replace the `...` with the name of the variable containing the BIC values.)

```
library(ggplot2)
df.bgBIC= data.frame(1:16,bg.outBIC$Subsets$BIC[-1])
names(df.bgBIC) = c("p", "BIC")
ggplot(data=df.bgBIC,mapping=aes(x=p,y=BIC))+
  geom_point() + geom_line()
```




# Question 6

Run the `summary()` function with the best AIC model from Q3. This produces output akin to that of the output from summarizing a linear model (e.g., one output by `lm()`). What is the adjusted $R^2$ value? What does the value imply about the quality of the linear fit with the best subset of variables?

```
summary(bg.outAIC$BestModel)
```

```
##
## Call:
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),
##     drop = FALSE], y = y))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.5140 -0.2805 -0.0260  0.2374  4.0422
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.39035    0.10587  13.133  < 2e-16 ***
## mag.i        0.45437    0.01034  43.930  < 2e-16 ***
## col.iJ      -0.74382    0.02971 -25.037  < 2e-16 ***
## col.JH       0.44669    0.08770   5.093 3.80e-07 ***
## J.G          3.68935    0.58905   6.263 4.46e-10 ***
## J.C          0.09251    0.05710   1.620  0.10536
## J.size      -1.30111    0.18470  -7.045 2.43e-12 ***
## H.G         -3.10045    0.62961  -4.924 9.04e-07 ***
## H.M20        0.31177    0.07178   4.343 1.46e-05 ***
## H.C          0.17098    0.05933   2.882  0.00399 **
## H.size       1.76041    0.19359   9.093  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5269 on 2382 degrees of freedom
## Multiple R-squared:  0.619,  Adjusted R-squared:  0.6174
## F-statistic:   387 on 10 and 2382 DF,  p-value: < 2.2e-16
```

```
adjusted r^2 value is 0.6174 which suggests that the quality of the linear fit with the best subset of variables
```

# Data Part 2

We'll continue by importing data about variable stars:

```
file.path = "https://raw.githubusercontent.com/pefreeman/36-290/master/EXAMPLE_DATASETS/TD_CLASS/css_data.Rdata"
load(url(file.path))
tmp = rep("NON-CB",length(response))
tmp[response==1] = "CB"
df = data.frame(predictors,"y"=factor(tmp))
rm(file.path,predictors,response,tmp)
```

A description of these data are here (https://github.com/pefreeman/36-290/tree/master/EXAMPLE_DATASETS/TD_CLASS). Here, the data frame that is input contains measurements for each of 46,808 stars, of which 30,582 are identified as "contact binaries." The rest are simply *not* contact binaries. Contact binaries are two stars that orbit a common center of mass and which share an envelope of gas. Think of a contact binary as being like a rotating dumbbell (one stellar core at each end), as opposed to the appearance of a single star, which is a simple sphere. Depending on one's vantage point, a rotating dumbbell of gas will have a brightness that varies over the course of one rotation.

The data contain 17 predictor variables that are summary statistics describing the variability of an observed star. The question for now is, which of these 17 are actually informative when we attempt to learn a statistical model relating variability statistics to the type of variable star?

# Question 7

You ultimately would want to repeat Q1, changing the `family` to one that is appropriate for two-class classification, but there's an issue. `bestglm()` in a logistic regression setting limits the number of predictor variables to 15. And given the amount of data we have here, `bestglm()` will be *slow*.So implement either `log_forward()` or `log_backward()` as they are given in the notes to determine the set of variables to keep here. Before doing so, remove `max.slope` from the data frame because some of its values are `Inf`. Don't worry about data splitting here…just run the code and see what you get! Pass all columns but `max.slope` and `y` as the first argument, and the column `y` as the second argument.

```
names(df)
```

```
##  [1] "amp"          "beyond.std"   "flux.mid20"   "flux.mid35"   "flux.mid50"
##  [6] "flux.mid65"   "flux.mid80"   "kurt"         "linear.trend" "mad"
## [11] "max.slope"    "mean"         "med.buff"     "per.amp"      "per.diff"
## [16] "skew"         "std"          "y"
```

```
df.no.maxslope = subset(df, select = -c(max.slope))

names(df.no.maxslope)
```

```
##  [1] "amp"          "beyond.std"   "flux.mid20"   "flux.mid35"   "flux.mid50"
##  [6] "flux.mid65"   "flux.mid80"   "kurt"         "linear.trend" "mad"
## [11] "mean"         "med.buff"     "per.amp"      "per.diff"     "skew"
## [16] "std"          "y"
```

```
log_forward = function(df.no.maxslope,y)

{
  var.num = ncol(df.no.maxslope)
  var.keep = aic.keep = c()
  var.rem = 1:var.num

  var = 0
  while ( var < var.num ) {
    var = var+1
    aic.tmp = rep(0,length(var.rem))
    for ( ii in 1:length(var.rem) ) {
      var.set = c(var.keep,var.rem[ii])
      df = df.no.maxslope[,var.set]
      if ( var == 1 ) df = data.frame(df)
      aic.tmp[ii] = summary(suppressWarnings(glm(y~.,data=df,family=binomial)))$aic
    }
    if ( length(aic.keep) == 0 || min(aic.tmp) < min(aic.keep) ) {
      aic.keep = append(aic.keep,min(aic.tmp))
      w = which.min(aic.tmp)
      var.keep = append(var.keep,var.rem[w])
      var.rem = var.rem[-w]
    } else {
      break

    }
  }
  return(sort(names(df.no.maxslope[var.keep])))
}



log_forward
```

```
## function(df.no.maxslope,y)
##
## {
##   var.num = ncol(df.no.maxslope)
##   var.keep = aic.keep = c()
##   var.rem = 1:var.num
##
##   var = 0
##   while ( var < var.num ) {
##     var = var+1
##     aic.tmp = rep(0,length(var.rem))
##     for ( ii in 1:length(var.rem) ) {
##       var.set = c(var.keep,var.rem[ii])
##       df = df.no.maxslope[,var.set]
##       if ( var == 1 ) df = data.frame(df)
##       aic.tmp[ii] = summary(suppressWarnings(glm(y~.,data=df,family=binomial)))$aic
##     }
##     if ( length(aic.keep) == 0 || min(aic.tmp) < min(aic.keep) ) {
##       aic.keep = append(aic.keep,min(aic.tmp))
##       w = which.min(aic.tmp)
##       var.keep = append(var.keep,var.rem[w])
##       var.rem = var.rem[-w]
##     } else {
##       break
##
##     }
##   }
##   return(sort(names(df.no.maxslope[var.keep])))
## }
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js