Week 9 - Homework

STAT 420, Summer 2017, Dalpiaz

Exercise 1 (`longley` Macroeconomic Data)

The built-in dataset `longley` contains macroeconomic data for predicting employment. We will attempt to model the `Employed` variable.

```
View(longley)
?longley
```

(a) What is the largest correlation between any pair of predictors in the dataset?

Solution:

```
sort(unique(as.numeric(cor(longley[-7]))), decreasing = TRUE)[2]
```

```
## [1] 0.9953
```

(b) Fit a model with `Employed` as the response and the remaining variables as predictors. Calculate and report the variance inflation factor (VIF) for each of the predictors. Which variable has the largest VIF? Do any of the VIFs suggest multicollinearity?

Solution:

```
employ_mod = lm(Employed ~ ., data = longley)
#summary(employ_mod)
library(car)
vif(employ_mod)
```

```
## GNP.deflator          GNP   Unemployed Armed.Forces   Population         Year
##      135.532     1788.513       33.619        3.589      399.151      758.981
```

```
vif(employ_mod)[which.max(vif(employ_mod))]
```

```
##  GNP
## 1789
```

The VIFs for every predictor except for `Armed.Forces` are extremely large, suggesting multicollinearity. `GNP` has the largest VIF.

(c) What proportion of the observed variation in `Population` is explained by a linear relationship with the other predictors?

Solution:

```
partfit_1 = lm(Employed ~ . - Population, data = longley)
partfit_2 = lm(Population ~ . - Employed, data = longley)
summary(partfit_2)$r.squared
```

```
## [1] 0.9975
```

99.75% of the variation of `Population` is explained by a linear relationship with the other *predictors*.

(d) Calculate the partial correlation coefficient for `Population` and `Employed` **with the effects of the other predictors removed**.

Solution:

```r
cor(resid(partfit_2), resid(partfit_1))
```

```
## [1] -0.07514
```

(e) Fit a new model with `Employed` as the response and the predictors from the model in **(b)** that were significant. (Use $\alpha = 0.05$.) Calculate and report the variance inflation factor for each of the predictors. Which variable has the largest VIF? Do any of the VIFs suggest multicollinearity?

Solution:

```r
summary(employ_mod)
```

```
##
## Call:
## lm(formula = Employed ~ ., data = longley)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.4101 -0.1577 -0.0282  0.1016  0.4554
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3.48e+03   8.90e+02   -3.91  0.00356 **
## GNP.deflator  1.51e-02   8.49e-02    0.18  0.86314
## GNP          -3.58e-02   3.35e-02   -1.07  0.31268
## Unemployed   -2.02e-02   4.88e-03   -4.14  0.00254 **
## Armed.Forces -1.03e-02   2.14e-03   -4.82  0.00094 ***
## Population   -5.11e-02   2.26e-01   -0.23  0.82621
## Year          1.83e+00   4.55e-01    4.02  0.00304 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.305 on 9 degrees of freedom
## Multiple R-squared:  0.995,  Adjusted R-squared:  0.992
## F-statistic:  330 on 6 and 9 DF,  p-value: 4.98e-10
```

```r
employ_mod_small = lm(Employed ~ Year + Armed.Forces + Unemployed, data = longley)
vif(employ_mod_small)
```

```
##         Year Armed.Forces   Unemployed
##        3.891        2.223        3.318
```

```r
vif(employ_mod_small)[which.max(vif(employ_mod_small))]
```

```
##  Year
## 3.891
```

None of these VIFs appear to be a problem. `Year` has the largest VIF. Note that we have fixed the multicollinearity, but that does not necessarily justify this model. This "procedure" for selecting variables is not justified in practice.

(f) Use an F-test to compare the models in parts **(b)** and **(e)**. Report the following:

- The null hypothesis
- The test statistic
- The distribution of the test statistic under the null hypothesis
- The p-value
- A decision

- Which model you prefer, **(b)** or **(e)**

Solution:

```
anova(employ_mod_small, employ_mod)
```

```
## Analysis of Variance Table
##
## Model 1: Employed ~ Year + Armed.Forces + Unemployed
## Model 2: Employed ~ GNP.deflator + GNP + Unemployed + Armed.Forces + Population +
##     Year
##   Res.Df   RSS Df Sum of Sq    F Pr(>F)
## 1     12 1.323
## 2      9 0.836  3     0.487 1.75   0.23
```

- Null: $\beta_{GNP.def} = \beta_{GNP} = \beta_{Pop} = 0$
- TS: $F = 1.75$
- Distribution: F with degrees of freedom 3 and 9
- p-value: 0.23
- Decision: Do NOT reject the null hypothesis.
- Prefer: The smaller model, based on the F test

(g) Check the assumptions of the model chosen in part **(f)**. Do any assumptions appear to be violated?

Solution:

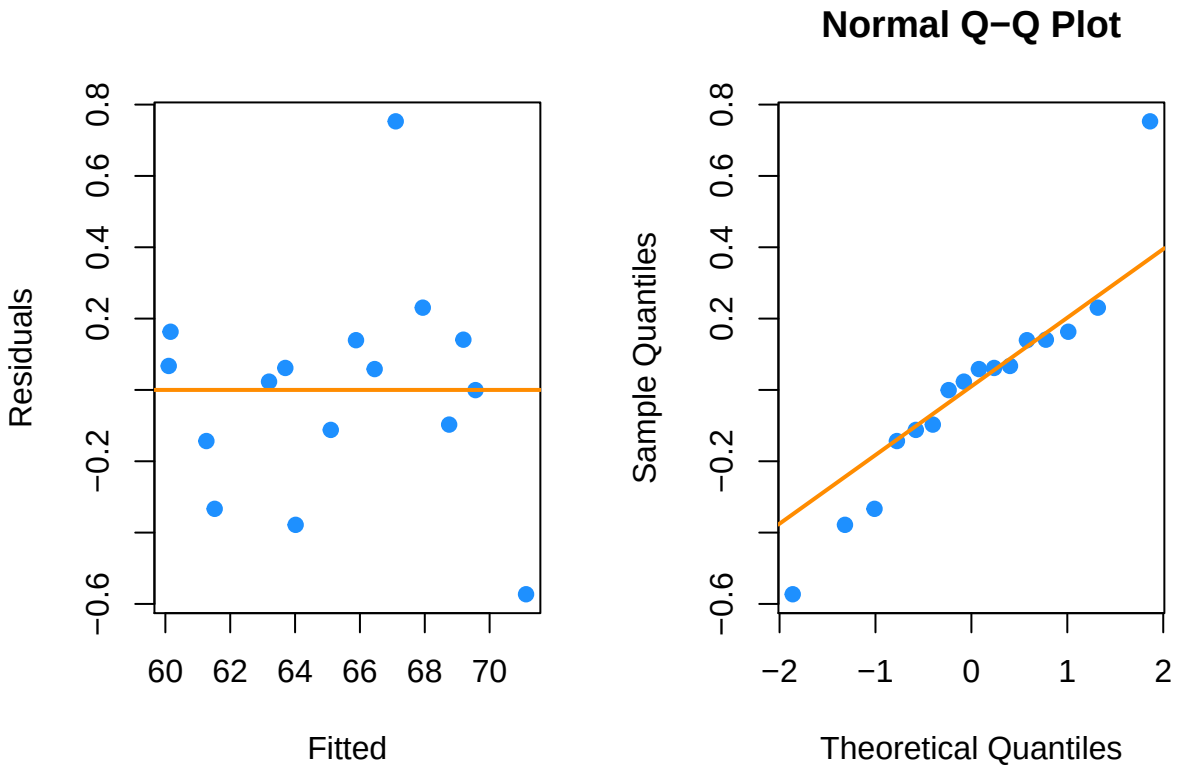
```
library(lmtest)
bptest(employ_mod_small)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  employ_mod_small
## BP = 2.5, df = 3, p-value = 0.5
```

```
shapiro.test(resid(employ_mod_small))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(employ_mod_small)
## W = 0.93, p-value = 0.2
```

```
par(mfrow = c(1, 2))
plot_fitted_resid(employ_mod_small)
plot_qq(employ_mod_small)
```



There do not appear to be any violations of assumptions.

Exercise 2 (`Boston` Housing Data)

(a) Use the `Boston` data found in the `MASS` package to find a "good" model for `medv`. Use any methods seen in class. The model should reach a LOOCV-RMSE below `3.25` and the Breusch-Pagan test should fail to reject at an α of 0.01. Do not use any transformations of the response variable.

Store your model in a variable called `good_model`. Run the two given chunks to verify your model meets the requested criteria. If you cannot find a model that meets both criteria, partial credit will be given for meeting at least one of the criteria.

Solution:

```
library(lmtest)

get_bp_decision = function(model, alpha) {
  decide = unname(bptest(model)$p.value < alpha)
  ifelse(decide, "Reject", "Fail to Reject")
}

get_loocv_rmse = function(model) {
  sqrt(mean((resid(model) / (1 - hatvalues(model))) ^ 2))
}
```

```
library(MASS)
good_model = step(lm(medv ~ . ^ 2 + I(lstat ^ 2) + I(lstat ^ 3) + I(nox ^ 2) +
```

```
                        I(crim ^ 2) + I(indus ^ 2), data = Boston), trace = FALSE)
```

```
get_bp_decision(good_model, alpha = 0.01)
get_loocv_rmse(good_model)
```

```
## [1] "Fail to Reject"
```

```
## [1] 3.09
```

Exercise 3 (Ball Bearings)

For this exercise we will use the data stored in `ballbearings.csv`. It contains 210 observations, each of which reports the results of a test on a set of ball bearings. Manufacturers who use bearings in their products have an interest in their reliability. The basic measure of reliability in this context is the rating life, also known in engineering as fatigue failure. The objective is to model `L50`, the median lifetime of this sample of ball bearings. The variables in the dataset are:

- `L50` - median life: the number of revolutions that 50% of a group of identical bearings would be expected to achieve
- `P` - the load on the bearing in operation
- `Z` - the number of balls in the bearing
- `D` - the diameter of the balls
- `Company` - denotes who manufactured the ball bearing (A, B, C)
- `Type` - Company B makes several types of ball bearings (1, 2, 3); 0 otherwise

(a) Find a model for `log(L50)` that does not reject the Shapiro-Wilk test at $\alpha = 0.01$ and obtains an **adjusted** R^2 higher than 0.52. You may not remove any observations, but may consider transformations. Your model should use fewer than 10 β parameters.

Store your model in a variable called `good_model_a`. Run the two given chunks to verify your model meets the requested criteria. If you cannot find a model that meets all criteria, partial credit will be given for meeting at least some of the criteria.

Solution:

```
get_sw_decision = function(model, alpha) {
  decide = unname(shapiro.test(resid(model))$p.value < alpha)
  ifelse(decide, "Reject", "Fail to Reject")
}

get_num_params = function(model) {
  length(coef(model))
}

get_adj_r2 = function(model) {
  summary(model)$adj.r.squared
}
```

```
ballbearings = read.csv("ballbearings.csv")
ballbearings[ballbearings$Company == "C", ]$Type = -1
ballbearings$Type = as.factor(ballbearings$Type) # important!!!
str(ballbearings)
```

```
## 'data.frame':    210 obs. of  6 variables:
##  $ L50    : num  84.5 74.2 68.1 66.8 79.4 25.7 44.7 73.2 82.7 41.6 ...
##  $ P      : int  4240 4240 4240 4240 4240 2530 4240 4240 4240 4240 ...
##  $ Z      : int  8 8 8 8 8 9 8 8 8 8 ...
```

```
##  $ D      : num  0.688 0.688 0.688 0.688 0.688 ...
##  $ Company: Factor w/ 3 levels "A","B","C": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Type   : Factor w/ 5 levels "-1","0","1","2",..: 2 2 2 2 2 2 2 2 2 2 ...
```

```
good_model_a = lm(log(L50) ~ log(P) + log(Z) + log(D) + Type, data = ballbearings)
```

```
get_sw_decision(good_model_a, alpha = 0.01)
get_num_params(good_model_a)
get_adj_r2(good_model_a)
```

```
## [1] "Fail to Reject"
```

```
## [1] 8
```

```
## [1] 0.5634
```

Trying some simple transformations on a small model quickly yields a correct answer. Notice that we modified the `Type` variable. It now contains all of the information that `Company` contains as well. There are essentially five types.

- Company A (Coded as `0`)
- Company C (Coded as `-1`)
- Company B, Type 1 (Coded as `1`)
- Company B, Type 2 (Coded as `2`)
- Company B, Type 3 (Coded as `3`)

(b) Find a model for `log(L50)` that does not reject the Shapiro-Wilk test at $\alpha = 0.01$ and obtains an **adjusted** R^2 higher than 0.60. You may not remove any observations, but may consider transformations. Your model should use fewer than 20 β parameters.

Store your model in a variable called `good_model_b`. Run the given chunk to verify your model meets the requested criteria. If you cannot find a model that meets all criteria, partial credit will be given for meeting at least some of the criteria.

Solution:

```
huge_fit = lm(log(L50) ~ poly(P, 3) + poly(Z, 3) + poly(D, 3) + Type *
              (P:Z + P:D + Z:D + P:Z:D), data = ballbearings)
good_model_b = step(huge_fit, direction = "backward",
                    trace = 0, k = log(nrow(ballbearings)))
```

```
get_sw_decision(good_model_b, alpha = 0.01)
get_num_params(good_model_b)
get_adj_r2(good_model_b)
```

```
## [1] "Fail to Reject"
```

```
## [1] 15
```

```
## [1] 0.6594
```

We start with a very large model and use selection to find a model that meets the stated criteria.

Exercise 4 (Does It Work?)

In this exercise, we will investigate how well backwards AIC and BIC actually perform. For either to be "working" correctly, they should result in a low number of both **false positives** and **false negatives**. In model selection,

- **False Positive**, FP: Incorrectly including a variable in the model. Including a *non-significant* variable

- **False Negative**, FN: Incorrectly excluding a variable in the model. Excluding a *significant* variable

Consider the **true** model

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6 + \beta_7 x_7 + \beta_8 x_8 + \beta_9 x_9 + \beta_{10} x_{10} + \epsilon$$

where $\epsilon \sim N(0, \sigma^2 = 9)$. The true values of the β parameters are given in the `R` code below.

```
beta_0  = 1
beta_1  = 0
beta_2  = 1
beta_3  = 0
beta_4  = 2
beta_5  = 0
beta_6  = 1
beta_7  = 0
beta_8  = 2
beta_9  = 0
beta_10 = 1
sigma = 3
```

Then, as we have specified them, some variables are significant, and some are not. We store their names in `R` variables for use later.

```
not_sig  = c("x_1", "x_3", "x_5", "x_7", "x_9")
signif = c("x_2", "x_4", "x_6", "x_8", "x_10")
```

We now simulate values for these `x` variables, which we will use throughout part **(a)**.

```
set.seed(42)
n = 100
x_1  = runif(n, 0, 10)
x_2  = runif(n, 0, 10)
x_3  = runif(n, 0, 10)
x_4  = runif(n, 0, 10)
x_5  = runif(n, 0, 10)
x_6  = runif(n, 0, 10)
x_7  = runif(n, 0, 10)
x_8  = runif(n, 0, 10)
x_9  = runif(n, 0, 10)
x_10 = runif(n, 0, 10)
```

We then combine these into a data frame and simulate `y` according to the true model.

```
sim_data_1 = data.frame(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_10,
  y = beta_0 + beta_2 * x_2 + beta_4 * x_4 + beta_6 * x_6 + beta_8 * x_8 +
      beta_10 * x_10 + rnorm(n, 0 , sigma)
)
```

We do a quick check to make sure everything looks correct.

```
head(sim_data_1)
```

```
##     x_1   x_2   x_3    x_4   x_5   x_6   x_7   x_8   x_9   x_10     y
## 1 9.148 6.262 8.851 4.8377 0.227 1.365 4.982 7.655 9.090 0.9615 37.66
## 2 9.371 2.172 5.171 4.4457 5.132 1.771 2.828 5.050 8.999 0.3690 27.05
## 3 2.861 2.166 8.519 0.6039 6.307 5.196 7.764 2.025 1.923 4.0094 17.62
## 4 8.304 3.889 4.428 3.2751 4.188 8.111 3.039 7.171 5.323 0.1154 34.42
```

```
## 5 6.417 9.425 1.579 8.7843 8.793 1.154 5.156 7.681 5.221 5.6789 48.03
## 6 5.191 9.626 4.423 9.3060 1.080 8.934 4.780 7.064 1.603 9.4495 61.16
```

Now, we fit an incorrect model.

```
fit = lm(y ~ x_5 + x_6 + x_7, data = sim_data_1)
coef(fit)
```

```
## (Intercept)         x_5         x_6         x_7
##    29.93131    -0.07864     1.22372     0.08668
```

Notice, we have coefficients for `x_5`, `x_6`, and `x_7`. This means that `x_5` and `x_7` are false positives, while `x_2`, `x_4`, `x_8`, and `x_10` are false negatives.

To detect the false negatives, use:

```
# which are false negatives?
!(signif %in% names(coef(fit)))
```

```
## [1]  TRUE  TRUE FALSE  TRUE  TRUE
```

To detect the false positives, use:

```
# which are false positives?
names(coef(fit)) %in% not_sig
```

```
## [1] FALSE  TRUE FALSE  TRUE
```

Note that in both cases, you could `sum()` the result to obtain the number of false negatives or positives.

(a) Set a seed equal to your birthday; then, using the given data for each `x` variable above in `sim_data_1`, simulate the response variable `y` 200 times. Each time,

- Fit an additive model using each of the `x` variables.
- Perform variable selection using backwards AIC.
- Perform variable selection using backwards BIC.
- Calculate and store the number of false negatives for the models chosen by AIC and BIC.
- Calculate and store the number of false positives for the models chosen by AIC and BIC.

Calculate the rate of false positives and negatives for both AIC and BIC. Compare the rates between the two methods. Arrange your results in a well formatted table.

Solution:

First some setup:

```
num_sims = 200
fp_aic = rep(0, num_sims)
fn_aic = rep(0, num_sims)
fp_bic = rep(0, num_sims)
fn_bic = rep(0, num_sims)
```

Then, running the simulation:

```
set.seed(42)
for (i in 1:num_sims) {

  sim_data_1$y = beta_0 + beta_2 * x_2 + beta_4 * x_4 + beta_6 * x_6 +
                 beta_8 * x_8 + beta_10 * x_10 + rnorm(n, 0 , sigma)

  fit = lm(y ~ ., data = sim_data_1)
  fit_back_aic = step(fit, direction = "backward", trace = 0)
  fit_back_bic = step(fit, direction = "backward", trace = 0, k = log(nrow(sim_data_1)))
```

```
  fn_aic[i] = sum(!(signif %in% names(coef(fit_back_aic))))
  fp_aic[i] = sum(names(coef(fit_back_aic)) %in% not_sig)
  fn_bic[i] = sum(!(signif %in% names(coef(fit_back_bic))))
  fp_bic[i] = sum(names(coef(fit_back_bic)) %in% not_sig)
}
```

Finally, checking the results:

```
results = data.frame(
  FP = c(mean(fp_aic), mean(fp_bic)),
  FN = c(mean(fn_aic), mean(fn_bic))
)

rownames(results) = c("AIC", "BIC")
knitr::kable(results)
```

	FP	FN
AIC	0.865	0
BIC	0.235	0

We see that with both AIC and BIC, no false negatives are produced, only false positives. That means, on average, both methods are returning models that are slightly too big. Also, BIC returns less false positives, which matches our intuition, as BIC will always return a smaller model for this sample size.

(b) Set a seed equal to your birthday; then, using the given data for each `x` variable below in `sim_data_2`, simulate the response variable `y` 200 times. Each time,

- Fit an additive model using each of the `x` variables.
- Perform variable selection using backwards AIC.
- Perform variable selection using backwards BIC.
- Calculate and store the number of false negatives for the models chosen by AIC and BIC.
- Calculate and store the number of false positives for the models chosen by AIC and BIC.

Calculate the rate of false positives and negatives for both AIC and BIC. Compare the rates between the two methods. Arrange your results in a well formatted table. Also compare to your answers in part **(a)** and suggest a reason for any differences.

```
set.seed(42)
x_1  = runif(n, 0, 10)
x_2  = runif(n, 0, 10)
x_3  = runif(n, 0, 10)
x_4  = runif(n, 0, 10)
x_5  = runif(n, 0, 10)
x_6  = runif(n, 0, 10)
x_7  = runif(n, 0, 10)
x_8  = x_6 + rnorm(n, 0, 0.1)
x_9  = x_6 + rnorm(n, 0, 0.1)
x_10 = x_4 + rnorm(n, 0, 0.1)

sim_data_2 = data.frame(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_10,
  y = beta_0 + beta_2 * x_2 + beta_4 * x_4 + beta_6 * x_6 + beta_8 * x_8 +
      beta_10 * x_10 + rnorm(n, 0 , sigma)
)
```

Solution:

First some setup:

```
num_sims = 200
fp_aic = rep(0, num_sims)
fn_aic = rep(0, num_sims)
fp_bic = rep(0, num_sims)
fn_bic = rep(0, num_sims)
```

Running the simulations on the new `x` data:

```
set.seed(42)
for (i in 1:num_sims) {

  sim_data_2$y = beta_0 + beta_2 * x_2 + beta_4 * x_4 + beta_6 * x_6 +
                 beta_8 * x_8 + beta_10 * x_10 + rnorm(n, 0 , sigma)

  fit = lm(y ~ ., data = sim_data_2)
  fit_back_aic = step(fit, direction = "backward", trace = 0)
  fit_back_bic = step(fit, direction = "backward", trace = 0, k = log(length(resid(fit))))

  fn_aic[i] = sum(!(signif %in% names(coef(fit_back_aic))))
  fp_aic[i] = sum(names(coef(fit_back_aic)) %in% not_sig)
  fn_bic[i] = sum(!(signif %in% names(coef(fit_back_bic))))
  fp_bic[i] = sum(names(coef(fit_back_bic)) %in% not_sig)
}
```

Lastly, checking the results:

```
results = data.frame(
  FP = c(mean(fp_aic), mean(fp_bic)),
  FN = c(mean(fn_aic), mean(fn_bic))
)

rownames(results) = c("AIC", "BIC")
knitr::kable(results)
```

	FP	FN
AIC	1.020	2.07
BIC	0.425	2.19

Again, we see that BIC returns fewer false positives than AIC. However, both now return false negatives! This is a result of the higher correlations within this set of predictors.

```
cor(sim_data_2[, c(4, 6, 8, 9, 10)])
```

```
##          x_4     x_6     x_8     x_9    x_10
## x_4  1.00000 0.02546 0.02889 0.02535 0.99941
## x_6  0.02546 1.00000 0.99948 0.99934 0.01858
## x_8  0.02889 0.99948 1.00000 0.99879 0.02198
## x_9  0.02535 0.99934 0.99879 1.00000 0.01867
## x_10 0.99941 0.01858 0.02198 0.01867 1.00000
```