Lab 2

IMC 490: Machine Learning for IMC

April 5, 2017

In this lab, we'll be going over the following topics:

- installing and loading packages
- fitting a linear regression model
- inspecting and visualizing a linear regression model and its diagnostics
- fitting a multiple regression model
- making predictions

Remember to use `?command` or `help(command)` in the R console to access documentation at any time if you have questions.

Setup

```
install.packages()
library()
require()
```

To get started, let's install and load the "ISLR" package, which contains the datasets used in the textbook.

```r
# you can install any package with install.packages("package")
install.packages("ISLR")
```

```r
# load the package
library(ISLR)
```

`library()` and `require()` both load packages - you can use either one.

Simple Linear Regression

```
lm()
summmary()
plot()
predict()
```

Let's start by loading the Auto data from last time, but using the "ISLR" package this time.

```r
data("Auto")
```

Our task is to predict the mpg (miles per gallon) of a vehicle. Always begin each analysis task by exploring the data using `names()` and `str()`.

```r
names(Auto)
```

```
## [1] "mpg"          "cylinders"    "displacement" "horsepower"
## [5] "weight"       "acceleration" "year"         "origin"
## [9] "name"
```

Now let's fit a single variable regression using weight to predict mpg. To fit a regression, we pass the regression equation into the `lm()` (linear model) function. Note that because the equality operator `=` is reserved for assignment in R, we must use the `~` operator in place of `=` to define the regression equation. Our regression equation is:

$$mpg = \beta_0 + \beta_1 * weight$$

This equation, translated into R code, would be: `mpg ~ weight`. *Note that the intercept (β_0) is implicit.* Now let's fit the model by passing in the equation and dataset into the `lm()` function, then inspect it using `summary()`.

```r
lm_fit_1 = lm(mpg ~ weight, data = Auto)
summary(lm_fit_1)
```

```
##
## Call:
## lm(formula = mpg ~ weight, data = Auto)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.9736  -2.7556  -0.3358   2.1379  16.5194
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 46.216524   0.798673   57.87   <2e-16 ***
## weight      -0.007647   0.000258  -29.64   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.333 on 390 degrees of freedom
## Multiple R-squared:  0.6926, Adjusted R-squared:  0.6918
## F-statistic: 878.8 on 1 and 390 DF,  p-value: < 2.2e-16
```

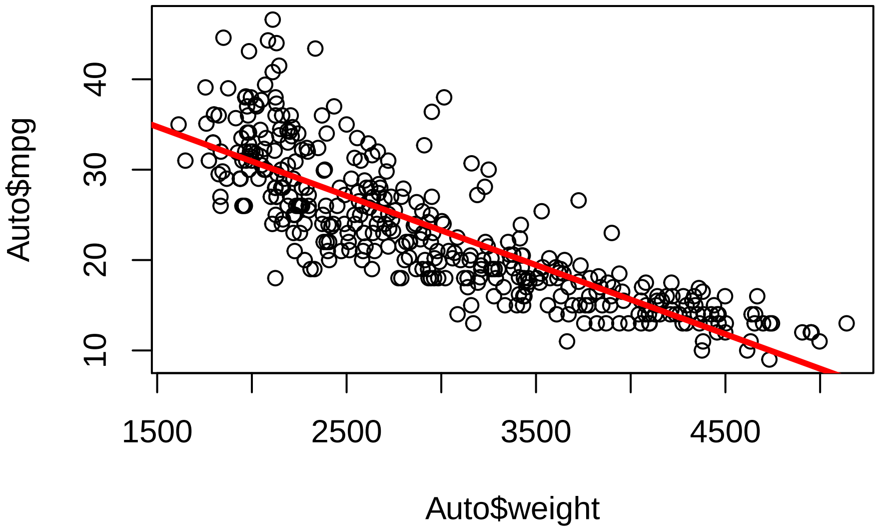
See "Anatomy of a Regression Summary" on Canvas (Labs -> Lab 2 - regression -> Regression Summary.pdf) for a visual explanation of this summary.

Visualizing the linear relationship

```
plot()
abline()
```

To visualize the linear relationship between two predictors, first create a scatterplot, then pass the regression into `abline()` to draw the line. You can set the parameters `lwd` (line weight) and `col` (color) to something more aesthetically pleasing.

```r
plot(Auto$weight, Auto$mpg)
abline(lm_fit_1, lwd = 3, col = "red")
```

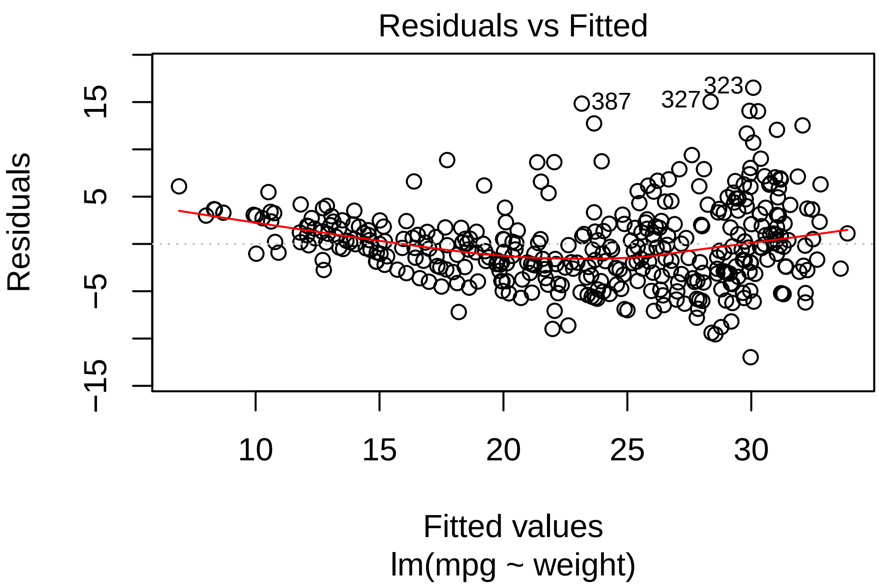


Diagnostic plots

When training regression models, it is very important to test for model misspecification. *Recall that an assumption of linear regression is independent and identically (normally) distributed residuals.*. To cycle through diagnostic plots, pass your model into the `plot()` function. To grab a specific plot, specify a number after the regression model.

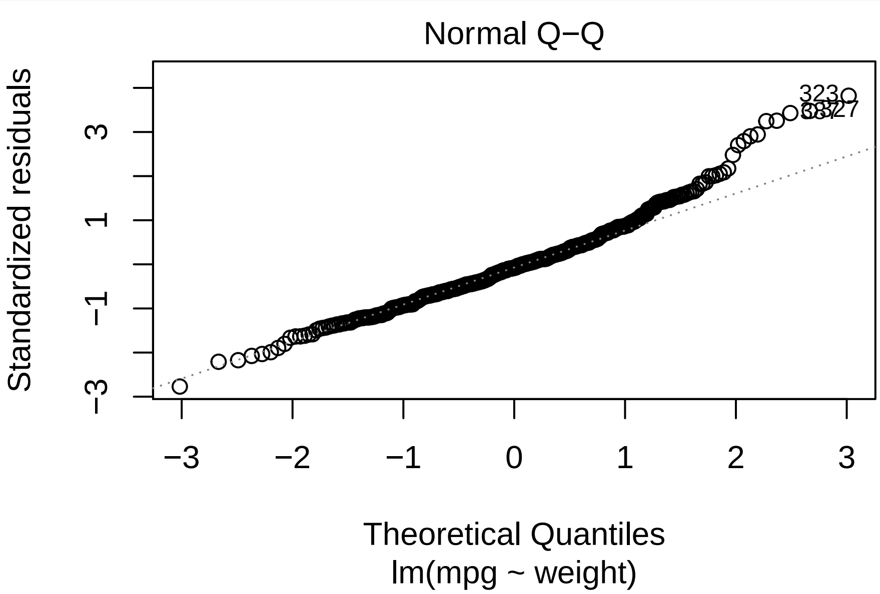
Residuals plot: Checks for randomly distributed residuals. This plot should be a random "snowstorm" of residuals. However, in this example we see a clear increase in residual variance across the x axis. This clues us in on the presence of heteroschedasticity, or non-constant variance.

```
plot(lm_fit_1, 1)
```



Q-Q plot: Checks for normality of residuals. Perfectly normal residuals should result in a 45 degree line. Since the residuals diverge from the 45 degree line, we conclude that the residuals do not satisfy the normality requirement very well.

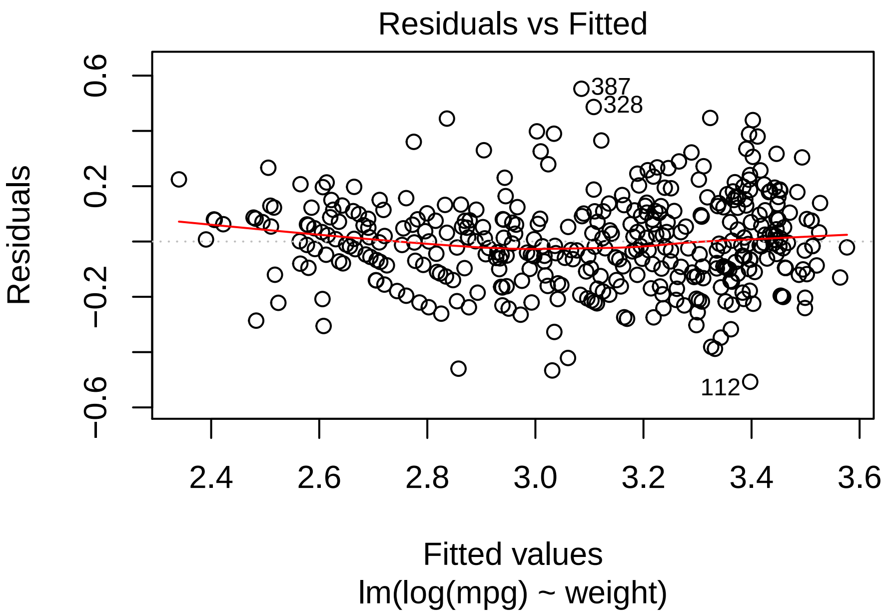
```
plot(lm_fit_1, 2)
```



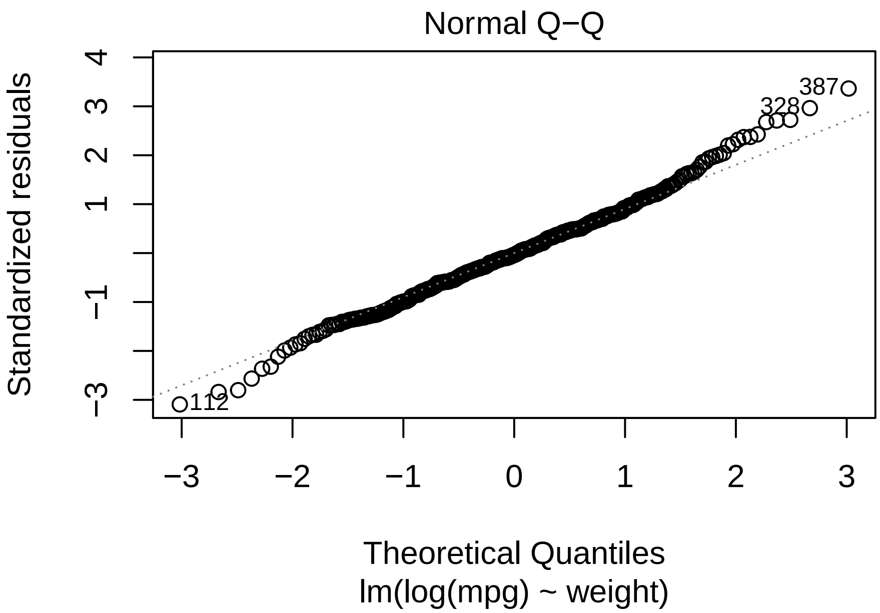
Let's see what happens when you take a variance stabilizing transformation of mpg.

We'll use `log(mpg)`

```
lm_fit_2 = lm(log(mpg) ~ weight, Auto)
plot(lm_fit_2, 1)
```




```
plot(lm_fit_2, 2)
```



Multiple Regression

predict()

Now let's train a regression with multiple predictors. To regress on multiple predictors, simply define the regression equation with additional predictors using the addition operator.

```
lm_fit_3 = lm(mpg ~ weight + horsepower + year, data = Auto)
lm_fit_3
```

```
##
## Call:
## lm(formula = mpg ~ weight + horsepower + year, data = Auto)
##
## Coefficients:
## (Intercept)       weight   horsepower         year
##  -13.719360    -0.006448    -0.005000     0.748705
```

Tip: you can train on all the predictors by putting a dot (.) on the right side of the regression equation instead of variable names.

```
lm_fit_4 = lm(mpg ~ ., data = Auto[ ,-9])
lm_fit_4
```

```
##
## Call:
## lm(formula = mpg ~ ., data = Auto[, -9])
##
## Coefficients:
##  (Intercept)     cylinders  displacement    horsepower        weight
##   -17.218435     -0.493376      0.019896     -0.016951     -0.006474
## acceleration          year        origin
##     0.080576      0.750773      1.426140
```

Finally, to make predictions using your model, pass the model and your new data into the `predict()` function. Let's compare the predictions using our simple regression model versus the multiple regression model.

```
Auto$mpg[10:12]
```

```
## [1] 15 15 14
```

```
predict(lm_fit_1, Auto[10:12, ])
```

```
##       10       11       12
## 16.77426 18.96904 18.61727
```

```
predict(lm_fit_4, Auto[10:12, ])
```

```
##       10       11       12
## 13.11319 15.29186 14.14690
```

Our multiple regression model is more accurate than our simple regression model, predicting values closer to the true `mpg`.

Analysis of correlation

`cor()`
`car::vif()`

Now that we've begun using libraries/packages, we will introduce the `::` operator. `::` allows you to check which functions belong to a package. For instance, our function for calculating variance inflation factor `vif()` is a part of the "car" package. To use this function, we can either load the entire library using `library(car)` and then use `vif(lm_fit)`, or we can use the `::` operator to pull the function directly `car::vif(lm_fit)`.

When performing regression using multiple predictors, it is important to check for correlation between the predictors, or multicollinearity.

Correlation matrix:

```r
cor(Auto[ ,-9])
```

```
##                     mpg  cylinders displacement horsepower     weight
## mpg           1.0000000 -0.7776175   -0.8051269 -0.7784268 -0.8322442
## cylinders    -0.7776175  1.0000000    0.9508233  0.8429834  0.8975273
## displacement -0.8051269  0.9508233    1.0000000  0.8972570  0.9329944
## horsepower   -0.7784268  0.8429834    0.8972570  1.0000000  0.8645377
## weight       -0.8322442  0.8975273    0.9329944  0.8645377  1.0000000
## acceleration  0.4233285 -0.5046834   -0.5438005 -0.6891955 -0.4168392
## year          0.5805410 -0.3456474   -0.3698552 -0.4163615 -0.3091199
## origin        0.5652088 -0.5689316   -0.6145351 -0.4551715 -0.5850054
##              acceleration       year     origin
## mpg             0.4233285  0.5805410  0.5652088
## cylinders      -0.5046834 -0.3456474 -0.5689316
## displacement   -0.5438005 -0.3698552 -0.6145351
## horsepower     -0.6891955 -0.4163615 -0.4551715
## weight         -0.4168392 -0.3091199 -0.5850054
## acceleration    1.0000000  0.2903161  0.2127458
## year            0.2903161  1.0000000  0.1815277
## origin          0.2127458  0.1815277  1.0000000
```

Variance inflation factor:

```r
library(car)
vif(lm_fit_4)
```

```
##    cylinders displacement   horsepower       weight acceleration
##    10.737535    21.836792     9.943693    10.831260     2.625806
##         year       origin
##     1.244952     1.772386
```