Lab 3: Polynomial Regression

STAT 632, Spring 2020

Note: This lab borrows from Ch. 3, pp. 115-117 of *An Introduction to Statistical Learning*.

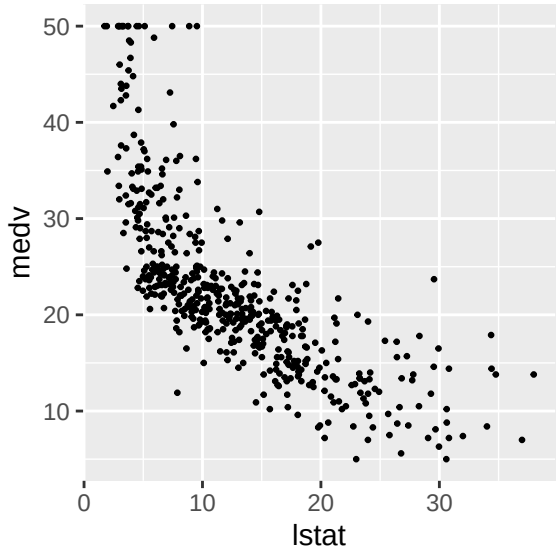
A polynomial regression model can be written as

$$Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_p x^p + e$$

One way to choose p is to keep adding terms until the added (highest order) term is no longer significant. It is recommended to keep all lower order terms in the model, even if they are not statistically significant. For example, if we fit a cubic model, then we should keep that x and x^2 terms in the model.

As an example, we consider a data set called `Boston` on housing values and other information about Boston suburbs. The data set is from the `MASS` package. We are interested in fitting a polynomial regression model with $Y =$ `medv` as the response and $x =$ `lstat` as the predictor. The variable `medv` is the median house value (in \$1000), and `lstat` is the percent of households with low socioeconomic status. A scatterplot of the data is shown below.

```
library(MASS)
library(ggplot2)
ggplot(data=Boston, aes(lstat, medv)) + geom_point(size=0.5)
```



Let's start with a quadratic polynomial model:

```
lm2 <- lm(medv ~ lstat + I(lstat^2), data = Boston)
summary(lm2)

##
## Call:
## lm(formula = medv ~ lstat + I(lstat^2), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.2834  -3.8313  -0.5295   2.3095  25.4148
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.862007   0.872084   49.15   <2e-16 ***
## lstat       -2.332821   0.123803  -18.84   <2e-16 ***
## I(lstat^2)   0.043547   0.003745   11.63   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.524 on 503 degrees of freedom
## Multiple R-squared:  0.6407,Adjusted R-squared:  0.6393
## F-statistic: 448.5 on 2 and 503 DF,  p-value: < 2.2e-16
```

The near-zero p-value indicates that quadratic term is significant and should be included in the model.

To fit a cubic regression model, we can add a predictor of the form `I(X^3)`. However this can start to get cumbersome as we continue to add higher order terms. A better approach is to use the `poly()` function. By default `poly()` uses what are called orthogonal polynomials (this means the columns of the design matrix are orthogonal). This can improve numeric stability. However, if we set the argument `raw=TRUE`, then `poly()` will do the same thing as if we had manually entered the terms using `I()`. Either way, the predictions will be the same when using `predict()` (it's just the coefficients that change when using orthogonal polynomials).

Moving forward we consider higher order terms:

```
lm3 <- lm(medv ~ poly(lstat, 3), data=Boston)
lm4 <- lm(medv ~ poly(lstat, 4), data=Boston)
lm5 <- lm(medv ~ poly(lstat, 5), data=Boston)
lm6 <- lm(medv ~ poly(lstat, 6), data=Boston)
```

The p-values are significant up to the fifth order term.

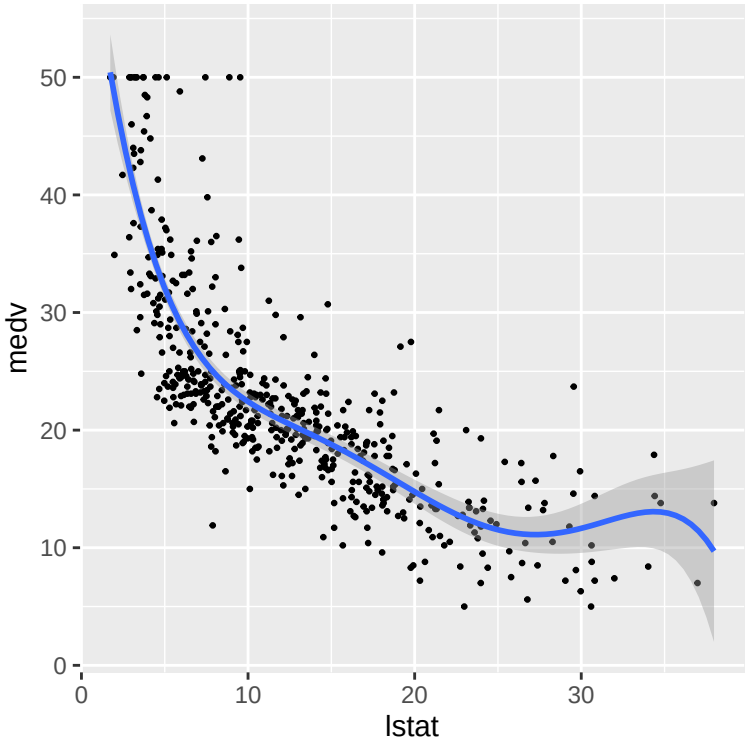
```
summary(lm5)

##
## Call:
## lm(formula = medv ~ poly(lstat, 5), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.5433  -3.1039  -0.7052   2.0844  27.1153
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       22.5328     0.2318  97.197  < 2e-16 ***
## poly(lstat, 5)1 -152.4595     5.2148 -29.236  < 2e-16 ***
## poly(lstat, 5)2   64.2272     5.2148  12.316  < 2e-16 ***
## poly(lstat, 5)3  -27.0511     5.2148  -5.187 3.10e-07 ***
## poly(lstat, 5)4   25.4517     5.2148   4.881 1.42e-06 ***
## poly(lstat, 5)5  -19.2524     5.2148  -3.692 0.000247 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.215 on 500 degrees of freedom
## Multiple R-squared:  0.6817,Adjusted R-squared:  0.6785
## F-statistic: 214.2 on 5 and 500 DF,  p-value: < 2.2e-16
```

The p-value for the sixth order term is not significant, so there is no improvement by considering terms beyond `lstat`^{5}.

```
summary(lm6)

##
## Call:
## lm(formula = medv ~ poly(lstat, 6), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.7317  -3.1571  -0.6941   2.0756  26.8994
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       22.5328     0.2317  97.252  < 2e-16 ***
## poly(lstat, 6)1 -152.4595     5.2119 -29.252  < 2e-16 ***
## poly(lstat, 6)2   64.2272     5.2119  12.323  < 2e-16 ***
## poly(lstat, 6)3  -27.0511     5.2119  -5.190 3.06e-07 ***
## poly(lstat, 6)4   25.4517     5.2119   4.883 1.41e-06 ***
## poly(lstat, 6)5  -19.2524     5.2119  -3.694 0.000245 ***
## poly(lstat, 6)6    6.5088     5.2119   1.249 0.212313
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.212 on 499 degrees of freedom
## Multiple R-squared:  0.6827,Adjusted R-squared:  0.6789
## F-statistic: 178.9 on 6 and 499 DF,  p-value: < 2.2e-16
```

Next we plot the estimated fifth degree polynomial regression curve on the scatterplot. We also add a confidence interval band (`ggplot2` does this by default).

```
ggplot(data=Boston, aes(lstat, medv)) +
  geom_point(size=0.5) +
  stat_smooth(method = 'lm', formula = y ~ poly(x, 5))
```



Cross-validation

The p-value approach suggested that we select the 5th degree polynomial regression model. An alternative way of selecting the degree is to use cross-validation. The idea is to split the data into two parts: a training set and a validation (or test) set. We then estimate the models on the training set, and make predictions on the withheld validation set. The models are then compared in terms of their root mean square prediction error:

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

where n_v is the number of observations in the validation set; and y_i and $\hat{y}_i$ are the actual and predicted values for the i^{th} observation in the validation set. This approach will give us an idea of how well our models perform on out-of-sample data.

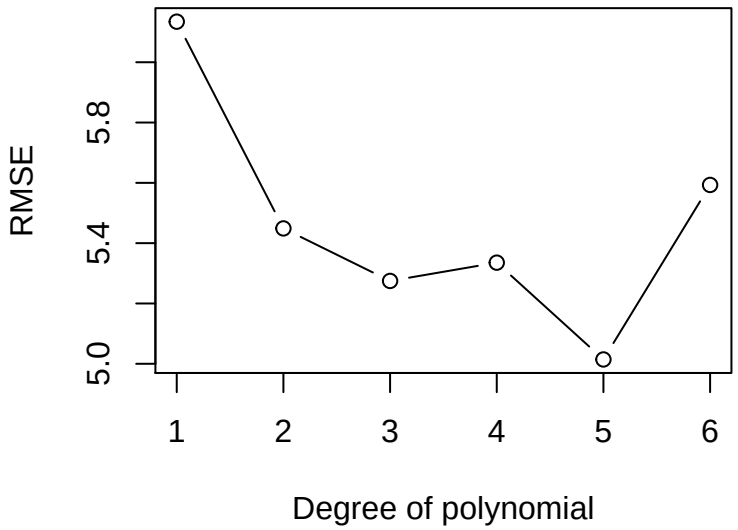
First, I wrote a function in R to compute the RMSE.

```
compute_rmse <- function(y, y_pred) {
  n <- length(y)
  sqrt((1 / n) * sum((y - y_pred)^2))
}
```

The Boston data set has $n = 506$ observations (rows). We randomly split the 506 observations into two sets, a training set containing 354 observations (70% of the data), and a validation set containing 152 observations (30% of the data).

```
set.seed(100)
n <- nrow(Boston)
train <- sample(1:n, size=354)

lm1_train <- lm(medv ~ lstat, data=Boston, subset=train)
lm2_train <- lm(medv ~ poly(lstat, 2), data=Boston, subset=train)
lm3_train <- lm(medv ~ poly(lstat, 3), data=Boston, subset=train)
lm4_train <- lm(medv ~ poly(lstat, 4), data=Boston, subset=train)
lm5_train <- lm(medv ~ poly(lstat, 5), data=Boston, subset=train)
lm6_train <- lm(medv ~ poly(lstat, 6), data=Boston, subset=train)

rmse <- rep(0, 6)
y_test <- Boston$medv[-train]
new_x <- data.frame(lstat = Boston$lstat[-train])
y1_pred <- predict(lm1_train, newdata = new_x)
rmse[1] <- compute_rmse(y_test, y1_pred)
y2_pred <- predict(lm2_train, newdata = new_x)
rmse[2] <- compute_rmse(y_test, y2_pred)
y3_pred <- predict(lm3_train, newdata = new_x)
rmse[3] <- compute_rmse(y_test, y3_pred)
y4_pred <- predict(lm4_train, newdata = new_x)
rmse[4] <- compute_rmse(y_test, y4_pred)
y5_pred <- predict(lm5_train, newdata = new_x)
rmse[5] <- compute_rmse(y_test, y5_pred)
y6_pred <- predict(lm6_train, newdata = new_x)
rmse[6] <- compute_rmse(y_test, y6_pred)
```

```r
# print RMSE vector
rmse

## [1] 6.134497 5.448901 5.274728 5.335416 5.014329 5.593054

par(mar=c(4.5,4,1,1)) # adjust margins
plot(c(1:6), rmse, xlab="Degree of polynomial", ylab="RMSE", type='b')
```



The cross-validation results agree with p-value approach, since the degree 5 polynomial has the lowest RMSE.

Note that it would have been better to write the code above as a `for-loop`. Also, note that the validation set approach can be highly variable. Another random split might select a different model. There are more stable approaches like leave-one-out cross-validation and 10-fold cross-validation. You can also take the average RMSE over several 70/30 splits of the data.

Regression Splines

We don't have to limit ourselves to just using polynomials to model nonlinearities. In fact, one shortcoming of polynomial regression is that each point affects the fit globally; polynomials can also start to overfit the data when the degree p is greater than 4. Regression splines are an alternative technique that fits polynomials locally. That is, polynomial models (usually cubic) are fit piecewise to the data, under some constraints for continuity and smoothness. `ggplot2` provides a convenient way to visualize and experiment with splines. You can read more about regression splines in Chapter 7 of *An Introduction to Statistical Learning*.

```
ggplot(data=Boston, aes(lstat, medv)) +
  geom_point(size=0.5) +
  stat_smooth(method = 'lm', formula = y ~ splines::bs(x,7))
```