Lab 5: Inference in Linear Regression and Model Assessment

Johnny Hong

Stat 154, Fall 2017

Introduction

In the previous lab, we have studied the use of `lm()` for OLS regression. In this lab, we are going to study the inferential aspects. To performance statistical inference, distributional assumptions on the data are required. For the next two sections of this lab assignment, assume that the data comes from the following model:

$$y_i = x_i^T \beta + \epsilon_i, i = 1, ..., n,$$

where $y_i \in \mathbb{R}$ is the outcome variable, $x_i \in \mathbb{R}^{p+1}$ is the (fixed) predictor vector that includes 1 as its first entry, $\beta \in \mathbb{R}^{p+1}$ is the coefficient vector, and ϵ_i are i.i.d. Gaussian with mean 0 and variance σ^2.

In many applications, inference might not be the objective and our goal is to develop a model with high predictive power. We will study how to assess model predictive power toward the end of this lab.

Confidence intervals

Confidence intervals play a central role in frequentist inference. In the context of regression, often we would like to construct confidence intervals for the intercept coefficient and the slope coefficients.

- To begin, use `lm()` to regress `mpg` on `disp` and `hp` and display the output from `summary()`.
- Based on the output, construct a 95% confidence interval of the slope coefficient for `disp`.

For a regression model with an intercept, a $100(1-\alpha)\%$ confidence interval for β_k is

$$(\hat{\beta}_k - t_{n-p-1,\alpha/2}\widehat{se}(\hat{\beta}_k), \hat{\beta}_k + t_{n-p-1,\alpha/2}\widehat{se}(\hat{\beta}_k)),$$

where:

- $\hat{\beta}_k$ is the slope coefficient for the kth predictor,

- $t_{n-p-1,\alpha/2}$ is the upper $(\alpha/2)$-th quantile for a t-distribution with $n-p-1$ degrees of freedom,
- $\widehat{se}(\hat{\beta}_k)$ is the estimated standard error of $\hat{\beta}_k$.

Hint: You might want to use `qt()` to obtain $t_{n-p-1,\alpha/2}$.

```
reg <- lm(mpg ~ disp + hp, data = mtcars)
summary(reg)
```

```
##
## Call:
## lm(formula = mpg ~ disp + hp, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.7945 -2.3036 -0.8246  1.8582  6.9363
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 30.735904   1.331566  23.083  < 2e-16 ***
## disp        -0.030346   0.007405  -4.098 0.000306 ***
## hp          -0.024840   0.013385  -1.856 0.073679 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.127 on 29 degrees of freedom
## Multiple R-squared:  0.7482, Adjusted R-squared:  0.7309
## F-statistic: 43.09 on 2 and 29 DF,  p-value: 2.062e-09
```

```
reg$coefficients["disp"] +
  c(-1, 1) * qt(0.975, reg$df.residual) *
  coef(summary(reg))["disp", "Std. Error"]
```

```
## [1] -0.04549091 -0.01520165
```

```
confint(reg, "disp", 0.95)
```

```
##              2.5 %      97.5 %
## disp -0.04549091 -0.01520165
```

- As you might have expected, there is a convenient command in R that does the job for us. Check your answer with `confint()`. If you have never used this command before, run `?confint` for the documentation.

Hypothesis testing

Hypothesis testing is an important technique in statistical inference. The idea is to use the observed data to assess the validity of the null hypothesis. The typical routine is as follows:

1. Formulate the null hypothesis H_0 and the alternative hypothesis H_1.
2. Determine a suitable test statistic T.
3. Derive the null distribution of T. That is, determine the distribution of T under the assumption that the null hypothesis is true.
4. Determine a significance level α. Typical values for α are 0.05 and 0.01.
5. Determine the critical region, the set of values for T such that the null hypothesis is rejected. The critical region depends on α and the directionality of the alternative hypothesis (one-sided vs two-sided).
6. Compute T based on the data. This is referred as the observed value of the test statistic, T_{obs}.
7. Check if T_{obs} is inside the critical region. If so, we reject the null hypothesis. Otherwise, we fail to reject the null.

One can also report the p-value, which is the chance that the test statistic is at least as extreme as T_{obs} under the null hypothesis H_0. We reject the null if the p-value is lower than the significance level α.

Keep in mind that you can never prove the null.

Suppose we would like to test the hypothesis $H_0 : \beta_k = c$, where c is some prespecified value. A commonly used test statistic is the t-statistic:

$$T = \frac{\hat{\beta}_k - c}{\hat{se}(\hat{\beta}_k)}.$$

Under the null, T follows a t-distribution with $n - p - 1$ degrees of freedom.

- Note that the coefficient section from `summary()` has a column called `t value`. The values in this column are the observed t-statistic, corresponding a certain choice of c. What is this choice?
- The column `Pr(>|t|)` contains the p-values. What alternative hypothesis is assumed here? One-sided or two-sided? What are the meanings of `***` and `.` next to the p-values?
- Based on the results, should we reject $H_0 : \beta_2 = 0$ in favor of $H_1 : \beta_2 \neq 0$ at 5% level? Here β_2 refers to the slope coefficient for `hp`.
- Compute the p-value if the hypothesis are $H_0 : \beta_1 = -0.05$ and $H_1 : \beta_1 > -0.05$. What is your conclusion?

- $c = 0$. - $H_1 : \beta_k \neq 0$. Two-sided test. '***' means that the p-value is below 0.001 and '.' means that the p-value is between 0.05 and 0.1. - We fail to reject the null $H_0 : \beta_2 = 0$ since the p-value is above 0.05. - We fail to reject the null $H_0 : \beta_1 = -0.05$ at 5% level. See below for the p-value.

```
reg_sum <- summary(reg)
1 - pt(q=(-0.05 - coef(reg_sum)[2, 1]) / coef(reg_sum)[2, 2], df=reg_sum$df[2])
```

```
## [1] 0.9936145
```

Assessment of model predictive power

We are going to use polynomial regression as an example. Consider the dataset `mtcars` and let the outcome variable y be `mpg` and the predictor variable x be `disp`. Consider the following models for $d = 1, 2, ..., 6$:

$$y_i = \beta_0 + \sum_{k=1}^{d} \beta_k x_i^k + \epsilon_i, \text{ for } i = 1, ..., n$$

where ϵ_i are uncorrelated with mean 0. Suppose we use the method of least squares to obtain parameter estimates.

The in-sample MSE is defined as

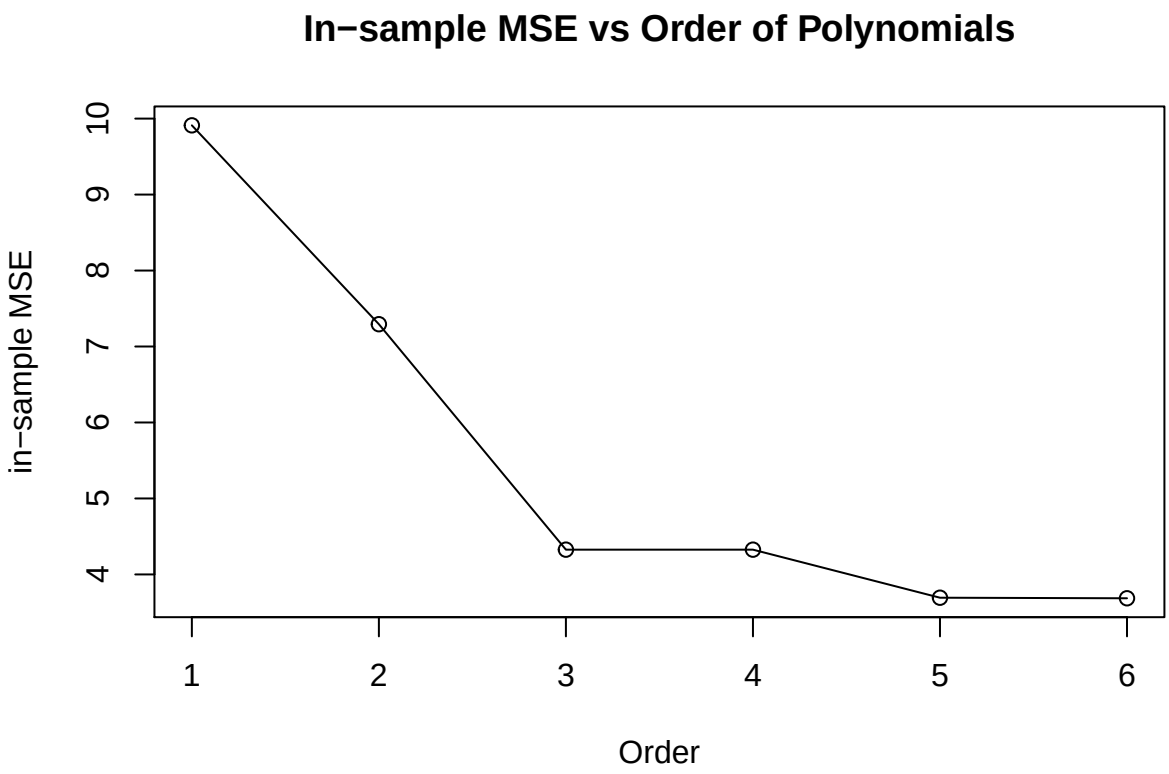
$$MSE_{in} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

where $\hat{y}_i$ is the fitted value for the ith observation.

- Compute the in-sample MSE for each model. Store it in a vector.
- Plot the in-sample MSEs against the order of the polynomial regression.
- Which model has the smallest in-sample MSE?
- Do you observe any trend in the graph?

Hint: `poly(..., raw=TRUE)` might be useful for this problem. For example, to fit a polynomial of degree 4, one can run `lm(y ~ poly(x, 4, raw=TRUE))`.

```
degs <- 1:6
models <- lapply(degs, function(deg) {
  lm(mpg ~ poly(disp, deg, raw=TRUE), data=mtcars)
})
inMSEs <- sapply(models, function(model) {
  mean(model$residuals^2)
})
plot(degs, inMSEs, type="o", main="In-sample MSE vs Order of Polynomials",
     xlab="Order", ylab="in-sample MSE")
```



The sixth degree model has the smallest in-sample MSE. As the order of polynomial goes up, the in-sample MSE always descreases.

As you might have noticed, in-sample MSEs are not an appropriate way to assess the predictive power of the models, since the training set, which is used for calibrating models, is also used for model testing. The in-sample MSEs tend to have an optimistic bias towards complicated models. There are in-sample metrics that take model complexity into account, such as Akaike's information criterion (AIC) and Bayesian Information Criterion (BIC), but these are not applicable when an explicit likelihood is not present in the model.

The most straightforward approach is to split the dataset into two parts: one for training and one for testing. This is known as the *holdout method*. A common split is 80-20: use 80% of the data to train the model and 20% to test the model.

Your turn - Select 20% of your dataset as holdout. You should use simple random sampling. - For each model, train on the remaining 80% of the data, predict the holdout data, and compute the MSE for the predictions. Identify which model gives the lowest holdout MSE. - Re-run the holdout method several times and see if the results vary greatly for different runs.

```
set.seed(32901)
p <- 0.2
holdoutInd <- sample(1:nrow(mtcars), floor(nrow(mtcars)) * p)
sapply(degs, function(deg) {
  model <- lm(mpg ~ poly(disp, deg, raw=TRUE), data=mtcars[-holdoutInd, ])
  mean((predict(model, mtcars[holdoutInd, ]) - mtcars[holdoutInd, "mpg"])^2)
})
```

```
## [1] 26.388677 17.274968  7.361311  7.392369 12.711777 14.518828
```

Model 3 seems to be the best model based on the holdout MSE. If you re-run the holdout method several times (without fixing the seed), you will often find that the results vary greatly.

Cross-validation

Cross-validation is an alternative to the holdout method. A useful package for such a task is `caret`. To generate folds, we can use `createFolds()`.

```
library(caret)
```

```
## Loading required package: lattice
```

```
set.seed(13491)
folds <- createFolds(mtcars$mpg)
folds
```

```
## $Fold01
## [1]  7 19
##
## $Fold02
## [1] 11 12 21
##
## $Fold03
## [1] 22 28 32
##
## $Fold04
## [1]  9 23 25
##
## $Fold05
## [1]  3 10 14 30
##
## $Fold06
## [1]  4  5 20
##
## $Fold07
## [1]  6 16 18
##
## $Fold08
## [1]  2  8 24 31
##
## $Fold09
## [1] 15 17 26 27
##
```

```
## $Fold10
## [1]  1 13 29
```

By default, ten folds are generated. Note that `folds` contains a list of vectors of indices. Since randomness is involved, you might get a different list.

Use `folds` to do a 10-fold cross validation to estimate the prediction error. Specifically,

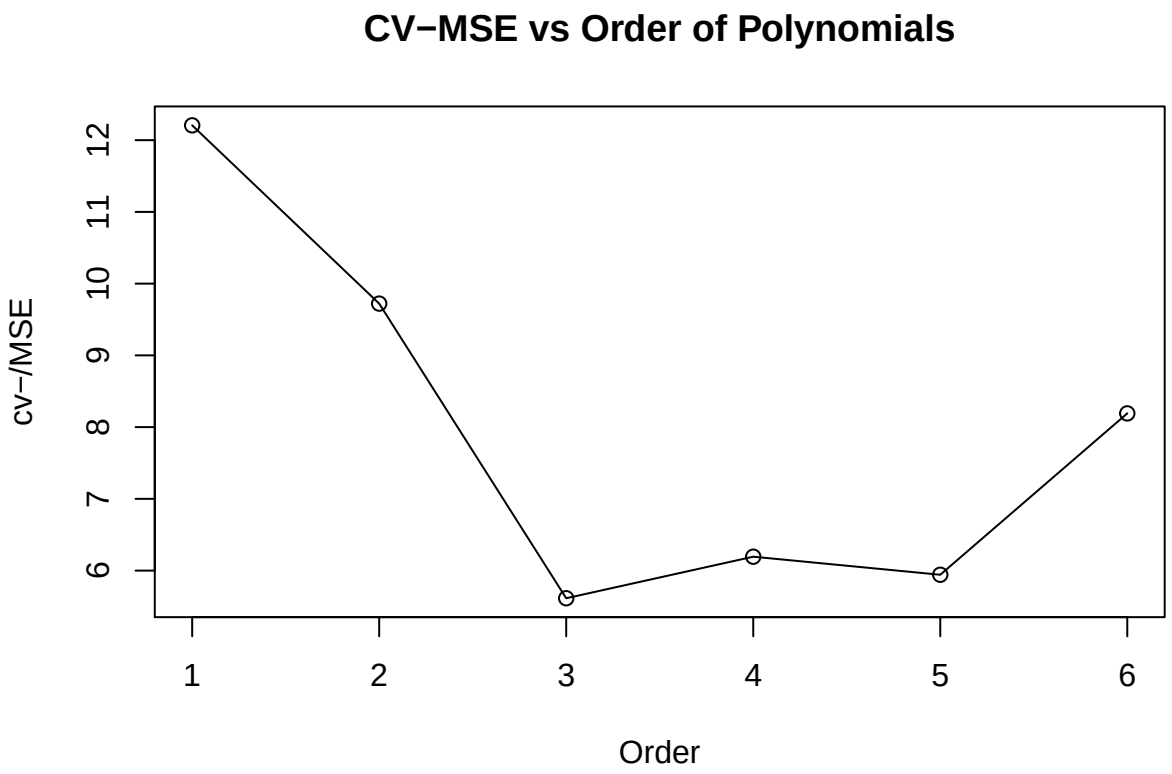
- For each fold,
  - For each order of polynomial,
    - Train the model based on all observations except the ones in the fold.
    - Predict the observation in the fold.
    - Compute the MSE of the predictions.

You should end up having a 6×10 matrix with rows corresponding to the polynomial orders and columns corresponding to the folds.

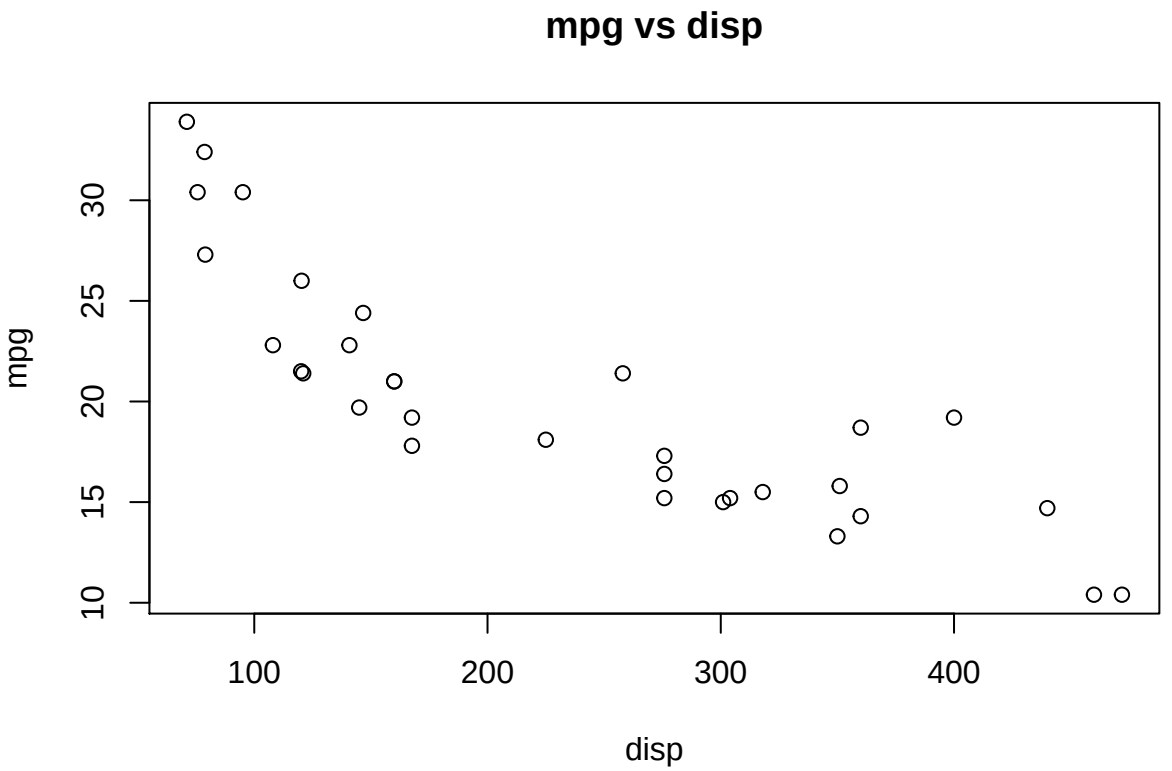
```
kfoldMSEs <- sapply(folds, function(ind) {
  sapply(degs, function(deg) {
    model <- lm(mpg ~ poly(disp, deg, raw=TRUE), data=mtcars[-ind, ])
    mean((predict(model, mtcars[ind, ]) - mtcars[ind, "mpg"])^2)
  })
})
cvMSEs <- rowMeans(kfoldMSEs)
cvMSEs
```

```
## [1] 12.206480  9.721606  5.614496  6.193341  5.941997  8.191140
```

```
plot(degs, cvMSEs, type="o", main="CV-MSE vs Order of Polynomials",
     xlab="Order", ylab="cv-/MSE")
```




```
plot(mpg ~ disp, data=mtcars, main="mpg vs disp")
```



The CV-MSE is defined as

$$MSE_{CV} = \frac{1}{\text{number of folds}} \sum_{\text{fold}} MSE_{\text{fold}},$$

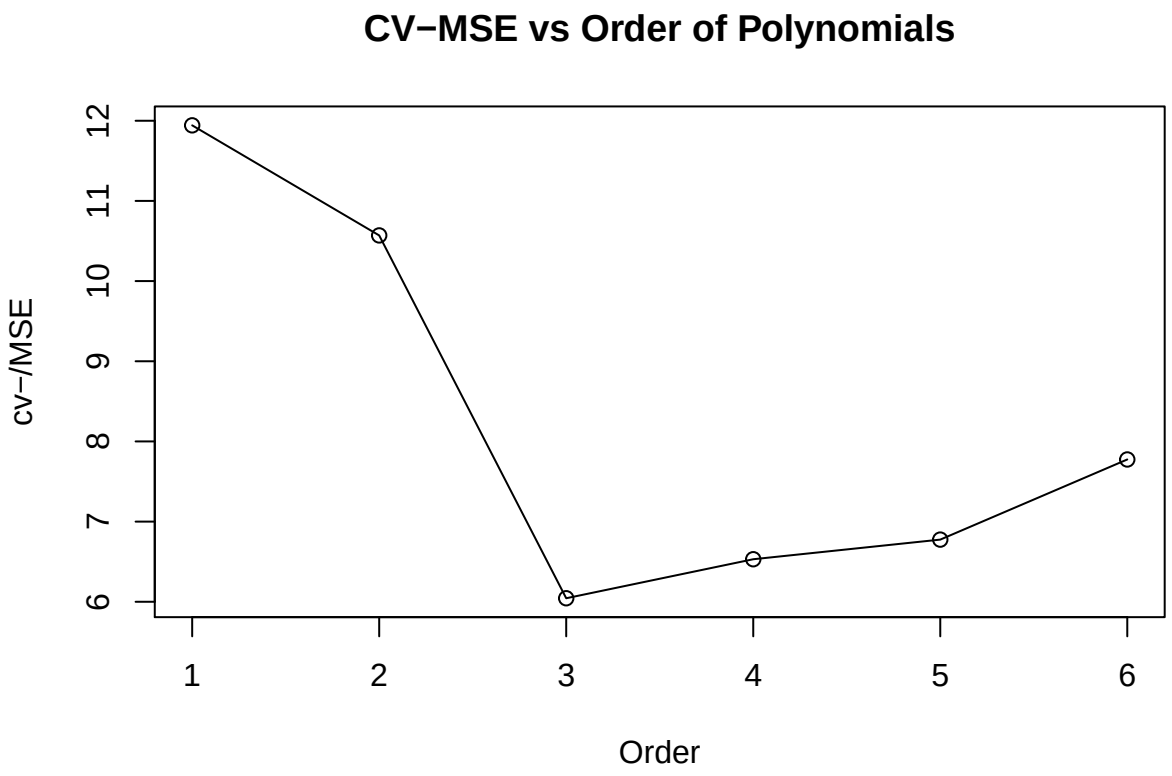
which is simply the average MSE over the folds.

1. Plot the CV-MSEs against the order of polynomial regression.
2. Which model gives the lowest CV-MSE? Is it reasonable? Why or why not? You might find plotting y against x useful in answering this question.
3. Re-run your code a couple times. Do the values of CV-MSEs stay the same? Why or why not?
4. Redo 1-3 with 5-fold only. This can be done by setting `k = 5` in `createFolds()`.
5. Redo 1-3 with n-fold. This is known as *leave-one-out* cross-validation (LOOCV). How many observations are in each fold? Can you explain the nomenclature leave-one-out cross-validation?

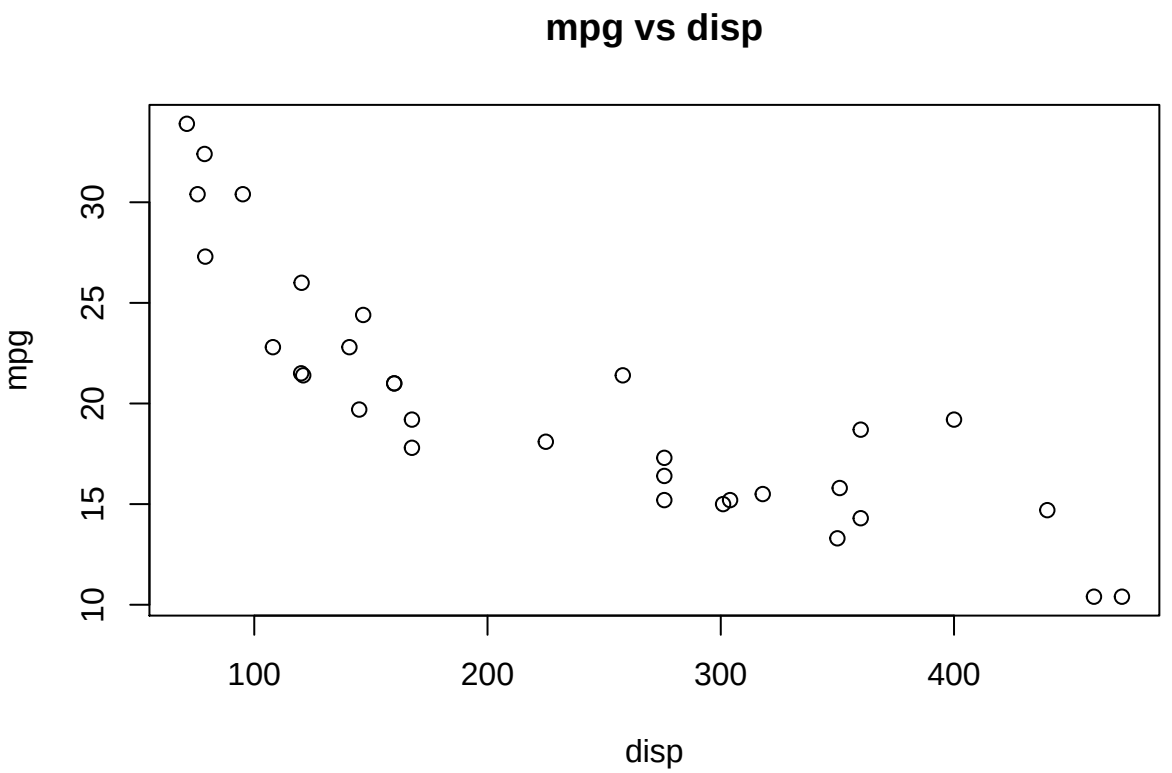
```
set.seed(13490)
folds <- createFolds(mtcars$mpg, k = 5)
kfoldMSEs <- sapply(folds, function(ind) {
  sapply(degs, function(deg) {
    model <- lm(mpg ~ poly(disp, deg, raw=TRUE), data=mtcars[-ind, ])
    mean((predict(model, mtcars[ind, ]) - mtcars[ind, "mpg"])^2)
  })
})
cvMSEs <- rowMeans(kfoldMSEs)
cvMSEs
```

```
## [1] 11.942528 10.569528  6.044101  6.530627  6.775306  7.774986
```

```
plot(degs, cvMSEs, type="o", main="CV-MSE vs Order of Polynomials",
     xlab="Order", ylab="cv-/MSE")
```




```
plot(mpg ~ disp, data=mtcars, main="mpg vs disp")
```




```
set.seed(41903)
folds <- createFolds(mtcars$mpg, k = nrow(mtcars))
```

```
folds
```

```
## $Fold01
## [1] 1
##
## $Fold02
## [1] 2
##
## $Fold03
## [1] 3
##
## $Fold04
## [1] 4
##
## $Fold05
## [1] 5
##
## $Fold06
## [1] 6
##
## $Fold07
## [1] 7
##
## $Fold08
## [1] 8
##
## $Fold09
## [1] 9
##
## $Fold10
## [1] 10
##
## $Fold11
## [1] 11
##
## $Fold12
## [1] 12
##
## $Fold13
## [1] 13
##
## $Fold14
## [1] 14
##
## $Fold15
```

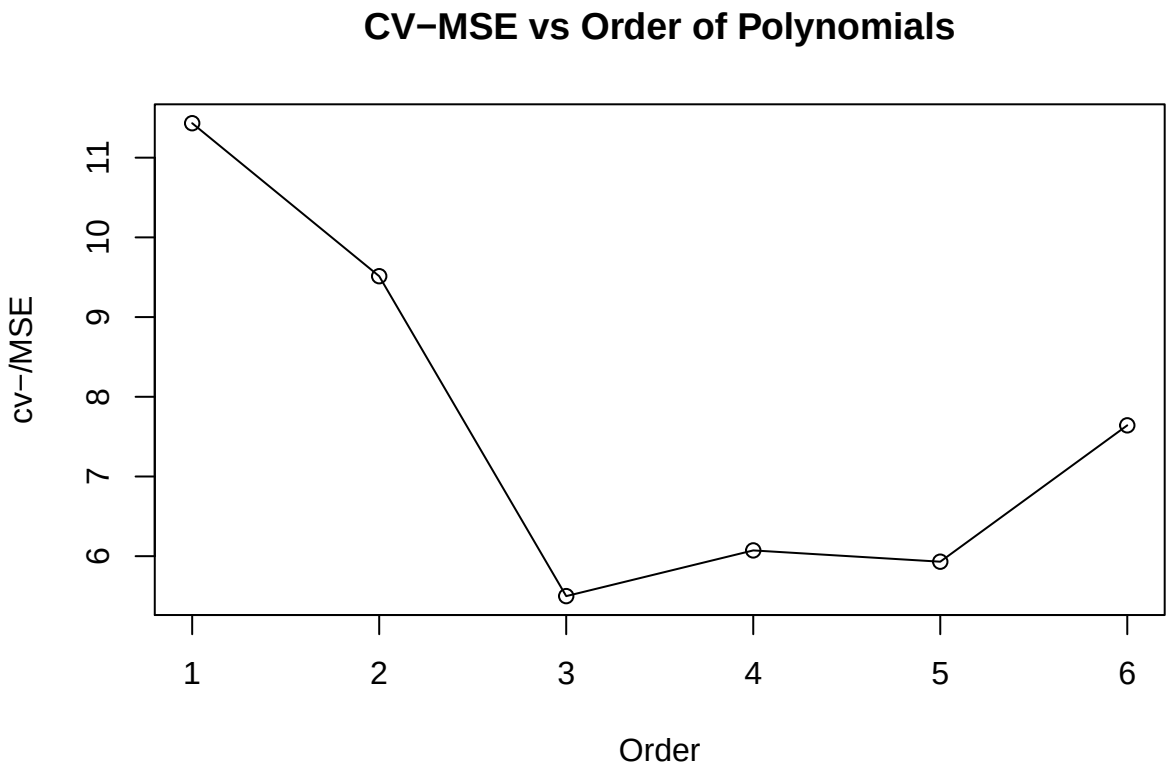
```
## [1] 15
##
## $Fold16
## [1] 16
##
## $Fold17
## [1] 17
##
## $Fold18
## [1] 18
##
## $Fold19
## [1] 19
##
## $Fold20
## [1] 20
##
## $Fold21
## [1] 21
##
## $Fold22
## [1] 22
##
## $Fold23
## [1] 23
##
## $Fold24
## [1] 24
##
## $Fold25
## [1] 25
##
## $Fold26
## [1] 26
##
## $Fold27
## [1] 27
##
## $Fold28
## [1] 28
##
## $Fold29
## [1] 29
##
## $Fold30
```

```
## [1] 30
##
## $Fold31
## [1] 31
##
## $Fold32
## [1] 32
```

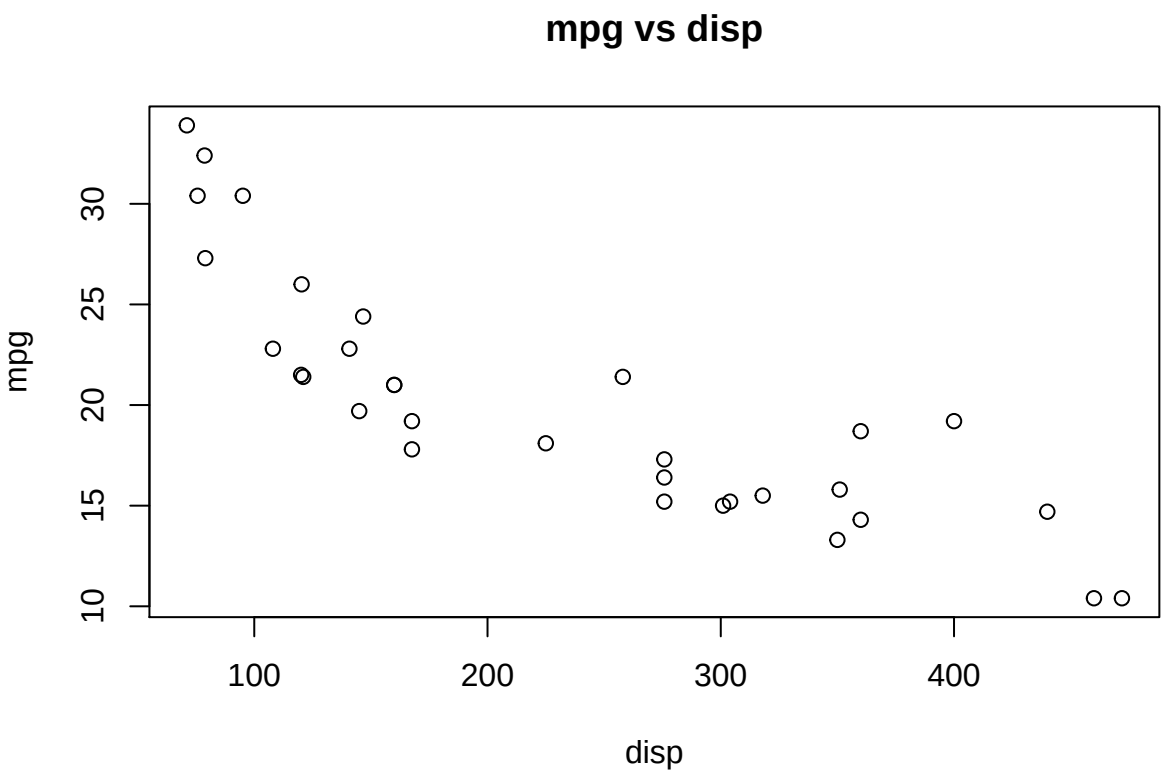
```
kfoldMSEs <- sapply(folds, function(ind) {
  sapply(degs, function(deg) {
    model <- lm(mpg ~ poly(disp, deg, raw=TRUE), data=mtcars[-ind, ])
    mean((predict(model, mtcars[ind, ]) - mtcars[ind, "mpg"])^2)
  })
})
cvMSEs <- rowMeans(kfoldMSEs)
cvMSEs
```

```
## [1] 11.432175  9.512764  5.499047  6.072497  5.930802  7.641600
```

```
plot(degs, cvMSEs, type="o", main="CV-MSE vs Order of Polynomials",
     xlab="Order", ylab="cv-/MSE")
```




```
plot(mpg ~ disp, data=mtcars, main="mpg vs disp")
```



Model 3 gives the lowest CV-MSE. As shown in the scatterplot, there seems to be a cubic relationship between 'mpg' and 'disp'. When you re-run the CV several times (if you don't fix the seed), you should be able to see that the values of CV-MSEs change, since randomness is involved in the fold partition. In LOOCV, there is exactly one observation in each fold. The nomenclature comes from the fact that in every iteration we train the model on the entire dataset except for one observation, and then we test the trained model on the observation that is left out.

Bootstrap

Bootstrap is another popular approach for model assessment. The idea is to iterate the following procedure many times: first, sample with replacement from the data (this serves as the training set); second, train the model on the sampled data; third, test the model on the data that is not in the sample and compute the performance metric, typically the MSE for regression problems. Finally, compute the average MSE over all the iterations, and this is referred as the *bootstrap MSE*.

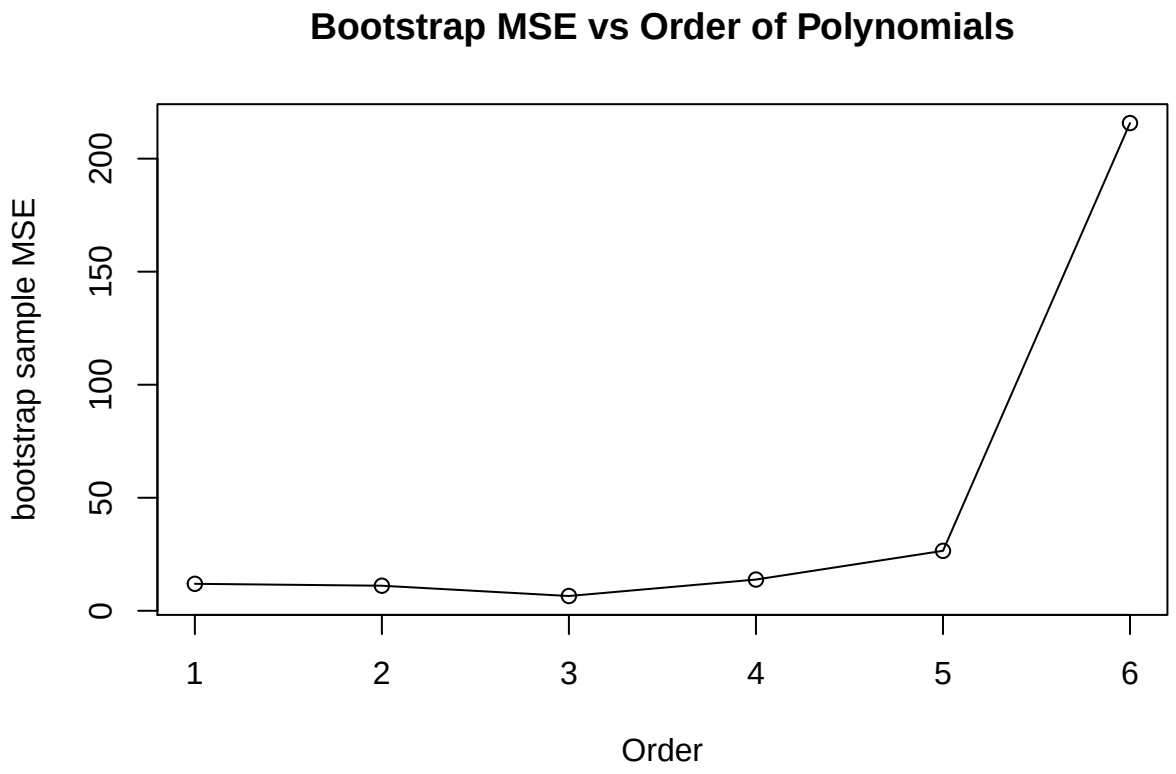
Do the following tasks with 400 bootstrap samples.

```
set.seed(114390)
resamples <- createResample(mtcars$mpg, 400)
resamplepMSEs <- sapply(resamples, function(resample) {
  sapply(degs, function(deg) {
    model <- lm(mpg ~ poly(disp, deg, raw=TRUE), data=mtcars[resample, ])
```

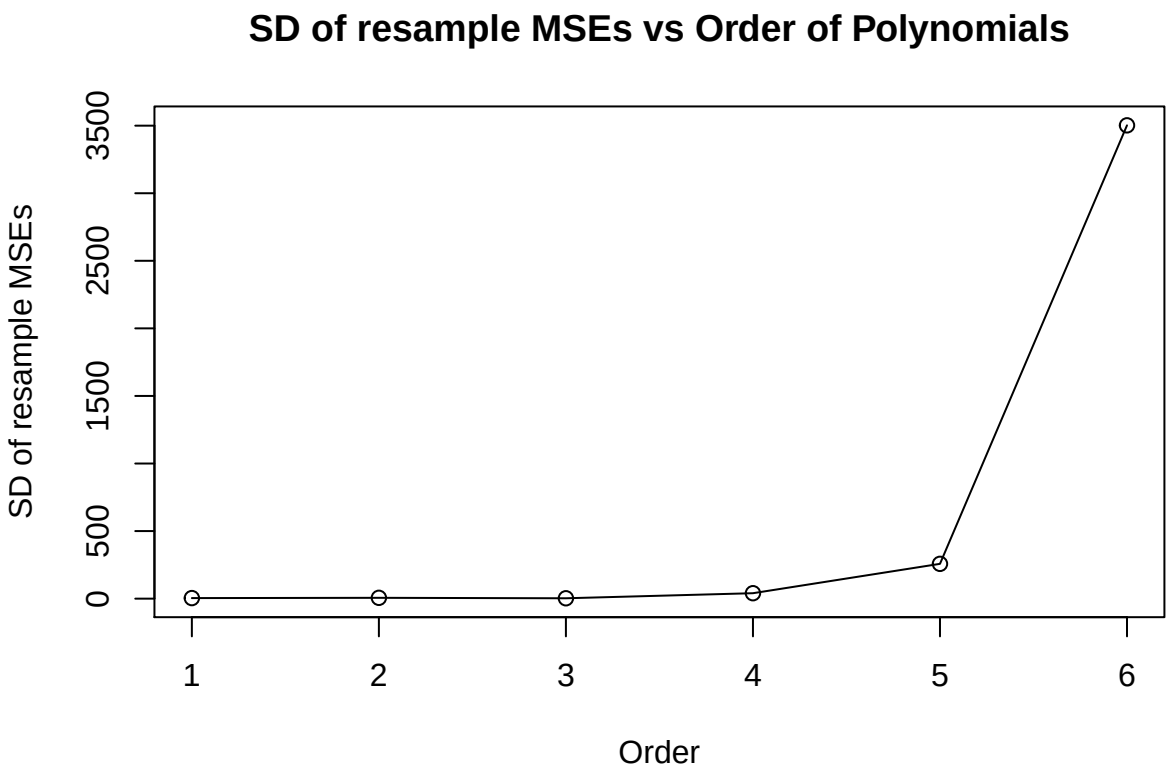
```
    mean((predict(model, mtcars[-resample, ]) - mtcars[-resample, "mpg"])^2)
  })
})
bootstrapMSEs <- rowMeans(resamplepMSEs)
bootstrapMSEs
```

```
## [1]  11.925396  11.091135   6.543014  13.823164  26.519796 215.727663
```

```
plot(degs, bootstrapMSEs, type="o", main="Bootstrap MSE vs Order of Polynomials",
     xlab="Order", ylab="bootstrap sample MSE")
```




```
plot(degs, apply(resamplepMSEs, 1, sd), type="o",
     main="SD of resample MSEs vs Order of Polynomials",
     xlab="Order", ylab="SD of resample MSEs")
```



1. Plot the bootstrap MSEs against the order of polynomial regression.
2. Which model gives the lowest bootstrap MSE? Is it reasonable? Why or why not? You might find plotting y against x useful in answering this question.
3. For each polynomial order, compute the SD of the 400 MSEs. Plot the SD against the polynomial order. What do you notice?
4. For each polynomial order, make a histogram of the 400 MSEs. What do you notice?
5. Based on what you saw in 3 and 4, do you think the bootstrap estimate is reliable? Why or why not?

Model 3 gives the lowest bootstrap MSE. As shown in the scatterplot, there seems to be a cubic relationship between 'mpg' and 'disp'. Note that the SDs increases very quickly as the degree goes up. From the histograms, we can see that the distribution of the estimated MSEs is more and more skewed to the right as the degree increases. Based on the aforementioned observation, the bootstrap estimate is not reliable since the estimated MSEs seems to have high variability and outlying values are often present, especially for models with high complexity.