

# 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 perform 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:

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

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  $\epsilon_i$  are i.i.d. Gaussian with mean 0 and variance  $\sigma^2$ .

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  $\beta_k$  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  $k$ th 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  $t_{n-p-1, \alpha/2}$ .

- 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  $\alpha$ . Typical values for  $\alpha$  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  $\alpha$  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  $\alpha$ .

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:

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

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  $\beta_2$  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?

## 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  $d = 1, 2, \dots, 6$ :

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


where  $\epsilon_i$  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  $i$ th 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))`. 

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.



## 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
```

```
folds <- createFolds(mtcars$mpg)
folds
```

```
## $Fold01
## [1]  1  6  9 25
##
## $Fold02
## [1] 11 30 31
##
## $Fold03
## [1] 17 26 28
##
## $Fold04
## [1]  4  8 12 24
##
## $Fold05
## [1] 14 21 27 29
##
## $Fold06
## [1] 23 32
##
## $Fold07
## [1]  2  7 13
##
## $Fold08
## [1] 15 18 22
##
## $Fold09
## [1]  3 10 20
##
## $Fold10
## [1]  5 16 19
```

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  $6 \times 10$  matrix with rows corresponding to the polynomial orders and columns corresponding to the folds.

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-MSEs? 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?



## 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.

1. Plot the bootstrap MSEs against the order of polynomial regression.
2. Which model gives the lowest bootstrap MSEs? 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?