Resampling Methods

Esam Mahdi

Data Analytics (ECMP 5005)
School of Mathematics and Statistics
Master of Engineering - Engineering Practice
Carleton University

September 18, 2023

Learning Objectives

By the end of this chapter, you should be able to do the following:

1. Assess the accuracy of the prediction model using cross validation methods.
2. Use bootstrapping methods to estimate of the standard errors, hypothesis testing, and confidence intervals for parameters.
3. Perform cross-validation and bootstrapping methods using R with some real-life applications.

- Recall, in regression and other prediction models, we make predictions from the training data and we need to know *how we can test the adequacy of our predictions?*
- We do this by comparing the *predicted values* with the *true values* for the response variable Y.
- In general, there are four methods to use:
  - ***Hold out set (test/train)*** cross validation.
  - ***K-fold*** cross validation.
  - ***Leave one out*** cross validation (LOOCV).
  - ***Bootstrap*** methods. Use one single training sample that we have to estimate parameters, like *standard errors*.

Recall: If we sample from the population over and over again, and each time, we recompute the estimator, then the *standard deviation* of these estimates is the *standard errors* under resampling.

- ***Training error:*** Obtained from the same observations that we used in the training set (the dataset we used to develop the model.)
- ***Test error:*** We split the data into two disjoint datasets: one is called *training data* which we use to train the model and then we apply this model on the *holding out set* that has not seen by the model, called *test data*, to calculate the *test error*.
- Note the training errors are too optimistic errors. The more we fit the data, the less error we get. On the other hand, the test errors get higher if we overfit well. Thus, the training errors can *underestimate* test error.

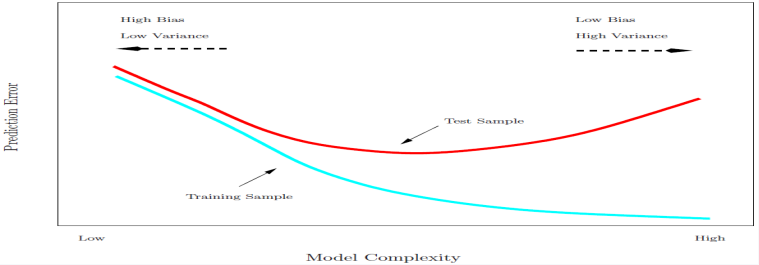
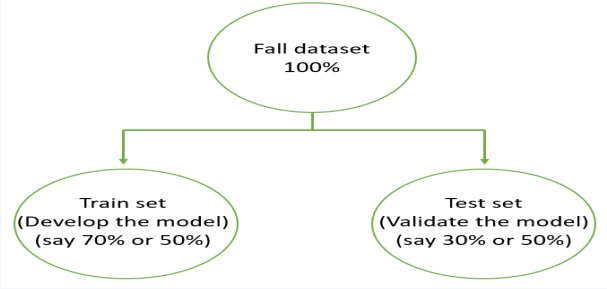


FIGURE: Source: An Introduction to Statistical Learning with Applications in R (2021).

Hold Out Set

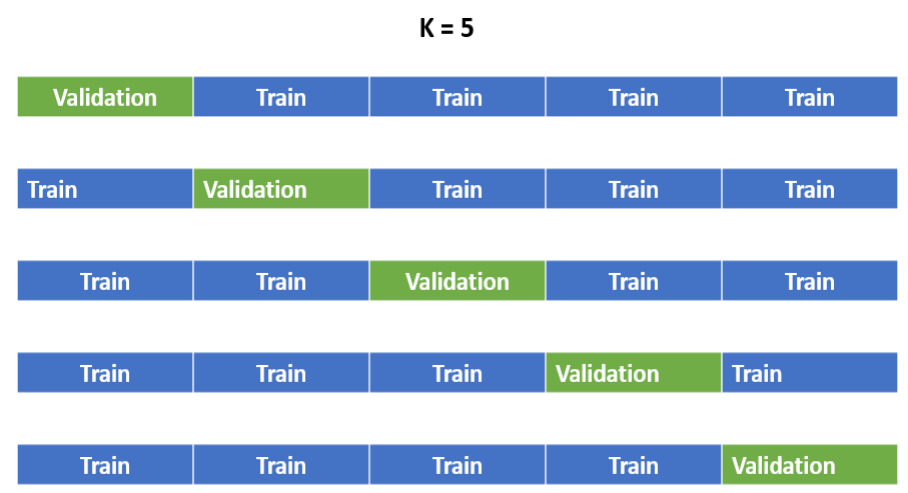
- Randomly divide the available set of samples into two parts: a *training set* and a *validation* or *hold-out set*.
- The model is fit on the training set, and this fitted model is used to predict the responses for the observations in the validation set.
- The resulting validation-set error provides an estimate of the test error. This is typically assessed using *mean Squared errors (MSE)* in the case of a *quantitative response* and *misclassification rate* in the case of a *qualitative (discrete) response*.



- This method suffers from issues of high variance and it is not recommended to use for small datasets.

K-FOLD CROSS-VALIDATION

Divide data randomly into K roughly equal-sized parts. Typically $K = 5$ or $K = 10$.



- Let the K parts be $C_1, C_2, \ldots C_K$, where C_k denotes the indices of the observations in part k. There are n_k observations in part k : if N is a multiple of K, then $n_k = n/K$.
- Compute

$$\mathrm{CV}_{(K)} = \sum_{k=1}^{K} \frac{n_k}{n} \mathrm{MSE}_k,$$

where

$$\mathrm{MSE}_k = \sum_{i \in C_k} (y_i - \hat{y}_i)^2 / n_k$$

and $\hat{y}_i$ is the fit for observation i, obtained from the data with part k removed.

LEAVE-ONE-OUT CROSS-VALIDATION (LOOCV)

Setting $K = n$ yields n-fold or *Leave-one-out cross-validation (LOOCV)*.

1. Here, we use the first observation as a test set and the remaining $n - 1$ observations to train the model.
2. The second observation serve as the test and the other remaining $n - 1$ values as a train set.
3. This algorithm is repeated until all observations have served as a test.

With regression models, the LOOCV can be obtained by fitting a single model as follows:

$$\mathrm{CV}_{(K)} = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1 - h_i} \right)^2,$$

where $\hat{y}_i$ is the ith fitted value from the original least squares fit, and $h_i = [H]_{ii}$ is the *leverage* (diagonal of the "hat" matrix, $H = X(X^T X)^{-1} X^T$, where X is the design matrix.) This is like the ordinary MSE, except the ith residual is divided by $1 - h_i$.

K-fold Cross Validation Details for Classification Problems

- Divide the data into K roughly equal-sized parts $C_1, C_2, \ldots C_K$, where C_k denotes the indices of the observations in part k. There are n_k observations in part k : if N is a multiple of K, then $n_k = n/K$.
- Compute

$$\mathrm{CV}_{(K)} = \sum_{k=1}^{K} \frac{n_k}{n} \mathrm{Err}_k,$$

where

$$\mathrm{Err}_k = \sum_{i \in C_k} I(y_i \neq \hat{y}_i)/n_k,$$

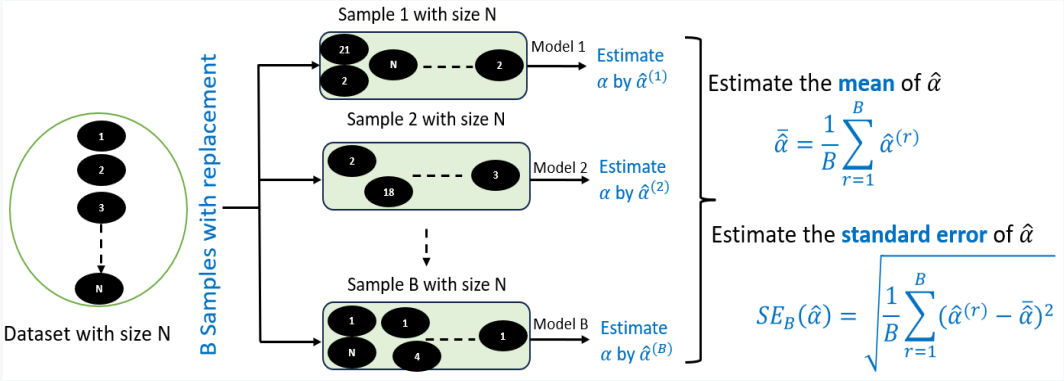
where $I(.)$ is the indicator function of two values 1 and 0.

- The estimated standard deviation of $\mathrm{CV}_{(K)}$ is

$$\widehat{\mathrm{SE}}(\mathrm{CV}_{(K)}) = \sqrt{\frac{1}{K} \sum_{k=1}^{K} \frac{(\mathrm{Err}_k - \overline{\mathrm{Err}_k})^2}{K-1}}$$

- *Bootstrapping* allows us to learn about a population by looking at a smaller set of the data.
- *Bootstrapping* is a method of resampling B times (say $B = 1000$ samples) from a single observed sample to estimate the statistical properties.
- Each of these *bootstrap data sets* is created by sampling *with replacement (repeat is allowed)*, and is the *same size* as the original dataset. As a result some observations may appear more than once in a given bootstrap data set and some not at all.
- It can provide an estimate of the standard errors, hypothesis testing, or confidence intervals for parameters.
- Note, if the data is a time series, then the observations are correlated. Thus, we can't generate bootstrap samples by sampling with replacement. We can instead use the block bootstrap methods.

A General Picture for the Bootstrap



CROSS-VALIDATION AND THE BOOTSTRAPPING IN R

Recall that we previously fit the following polynomial regression to the `Auto` dataset in the **ISLR2** package:

$$\text{mpg} = 56.9001 - 0.4662 \times \text{horsepower} + 0.0012 \times \text{horsepower}^2$$

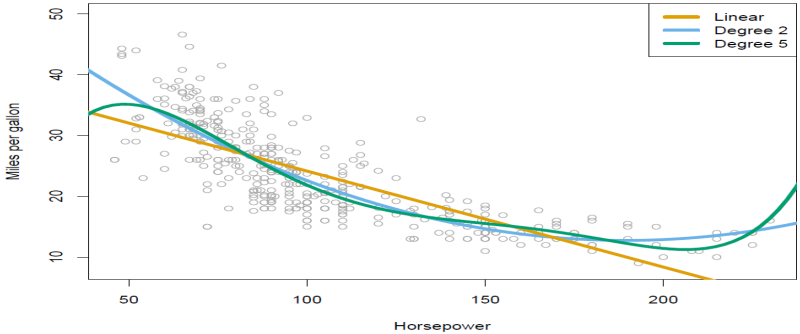


FIGURE: Auto data and fitted polynomial regressions

In this example, we need to explore the resampling techniques covered in this chapter.

LEAVE-ONE-OUT CROSS-VALIDATION (LOOCV): USE THE **boot** PACKAGE

```
require("ISLR2") #to get the "Auto" dataset
require("boot") #to perform cv and bootstrapping methods for GLM
fit_1 <- glm(mpg ~ poly(horsepower, 2), data = Auto)
cv.glm(data = Auto, glmfit = fit_1)$delta #default: K=nrow(dataset)

## [1] 19.24821 19.24787
```

The `cv.glm()` function produces a list with several components. The two numbers in the `delta` vector contain the cross-validation results. The first component of delta is the average mean-squared error that we obtain from performing a cross validation method. The second component is the average mean-squared error but with a bias correction. Our cross-validation estimate for the test error is approximately 19.25.

Note that in R, the `lm()` function (stands for *linear model*) is used to fit the traditional linear regression model, which is a special case of the `glm()` function (*generalized linear model*). If we use the `glm()` function without specifying the `family` (i.e., use the default argument of `family = gaussian`), then we get the exact result of using the `lm()` function (i.e., `glm() = lm()` in this case). We will discuss the `glm()` function in more details when we study the logistic regression and other generalized linear models later!

LEAVE-ONE-OUT CROSS-VALIDATION (LOOCV): USE THE **caret** PACKAGE

```
require(caret)
train(mpg ~ poly(horsepower, 2),
                data = Auto,
                method = "lm",
        trControl = trainControl(method = "LOOCV"))

## Linear Regression
##
## 392 samples
##   1 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 391, 391, 391, 391, 391, 391, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   4.387279  0.6832308  3.272041
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

LOOCV OUTPUT RESULTS FROM `TRAIN()` FUNCTION IN **caret** PACKAGE

- *RMSE: root mean squared error*. It measures the average difference between the predictions made by the model and the actual observations. The lower of the RMSE is the more accurate of the model predictions.

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

- *Rsquared: Coefficient of determination*. It measures how much variability in the response variable that can be explained by the model. The higher the R-squared, the more closely a model can predict the actual observations.

- *MAE: mean absolute error*. It measures the average absolute difference between the predictions made by the model and the actual observations. The lower the MAE, the better the model.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|.$$

In practice, we typically fit several different models and compare these metrics (RMSE, R-squared, and MAE) to decide which model produces the lowest test error rates and is therefore the best model to use.

TRAINING/TEST (HOLD OUT) CROSS VALIDATION

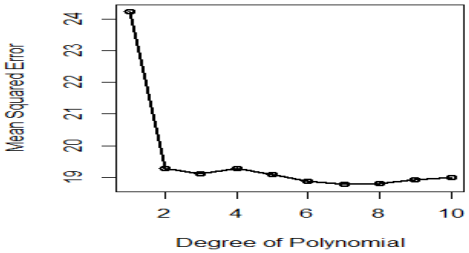
```
set.seed(123) # to replicate the random selections
# select randomly 50% of the Auto dataset
Index <- sample(1:392, 196) #default: sample(replace = FALSE)

# Alternatively, use createDataPartition() function in "caret" package
#Index <- createDataPartition(Auto$mpg, p = 0.5, list = FALSE)

traindata <- Auto[Index,]  # training set
testdata <- Auto[-Index,]  # test set
fit_2 <- lm(mpg ~ poly(horsepower, 2),
                        data = traindata)
pred <- predict(fit_2, testdata)

# Calculate the RMSE for the test data
sqrt(mean((testdata$mpg - pred)**2))

## [1] 4.059694
```

10-FOLD CROSS VALIDATION

```
kcv <- trainControl(method = "cv", number = 10)
train(mpg ~ poly(horsepower, 2),
            data = Auto,
            method = "lm",
            trControl = kcv)

## Linear Regression
##
## 392 samples
##   1 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 353, 353, 354, 352, 353, 353, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   4.365403  0.6961696  3.27712
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

USE CV TO SELECT THE DEGREE OF POLYNOMIAL

```
set.seed(1)
cv.error.10 = rep(0,10) # to save the results from the loop
for (i in 1:10){
        glm.fit = glm(mpg ~ poly(horsepower, i), data = Auto)
        cv.error.10[i] = cv.glm(Auto, glm.fit, K=10)$delta[1]
}
cv.error.10

##  [1] 24.21538 19.28327 19.12998 19.29201 19.09471 18.87874 18.78127 18.80484
##  [9] 18.92676 18.99439

plot(1:10,cv.error.10, type = "o", lwd = 2,
        xlab = "Degree of Polynomial", ylab = "Mean Squared Error")
```



BOOTSTRAP (OUTPUT NEXT PAGE)

Here, we use bootstrapping method to estimate the standard errors of the coefficient estimates $\hat{\beta}_0, \hat{\beta}_1$, and $\hat{\beta}_2$ using the `boot()` function in the package **boot** as follows:

STEP1 Write a function to extract the coefficient estimates $\hat{\beta}_0, \hat{\beta}_1$, and $\hat{\beta}_2$ from the fitted regression model. This function must take at least two arguments. The first argument passed will always be the original data. The second will be a vector of indices which define the bootstrap sample:

```
boot_fn <- function (data ,index){
    fit <- lm(mpg~horsepower+I(horsepower^2),data=data,subset=index)
   return(coef(fit)) }
boot_fn(Auto,1:nrow(Auto))
```

STEP2 Use the `boot()` function to calculate the standard errors of $B = R = 1,000$ bootstrap estimates for the regression coefficients:

```
set.seed(1)
bootstrap <- boot(data = Auto, statistic = boot_fn, R = 1000)
```

STEP3 You may compare the standard errors you got from Step 2 with those obtained from the following code:

```
summary(lm(mpg~horsepower+I(horsepower^2),data=Auto))$coef
```

You may also construct confidence intervals for the coefficients. For example, a 95% confidence interval for a given coefficient β is given by $\left[\hat{\beta} \pm 2 \times \text{SE}(\hat{\beta})\right]$.

BOOTSTRAP (OUTPUT RESULTS FROM PREVIOUS CODE)

```
##     (Intercept)      horsepower I(horsepower^2)
##    56.900099702    -0.466189630     0.001230536
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Auto, statistic = boot_fn, R = 1000)
##
##
## Bootstrap Statistics :
##         original        bias     std. error
## t1* 56.900099702  3.511640e-02 2.0300222526
## t2* -0.466189630 -7.080834e-04 0.0324241984
## t3*  0.001230536  2.840324e-06 0.0001172164
```

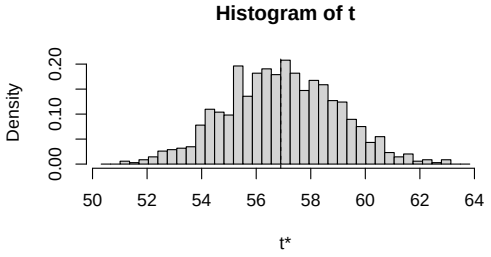
```
##                     Estimate   Std. Error   t value      Pr(>|t|)
## (Intercept)     56.900099702 1.8004268063  31.60367 1.740911e-109
## horsepower      -0.466189630 0.0311246171 -14.97816  2.289429e-40
## I(horsepower^2)  0.001230536 0.0001220759  10.08009  2.196340e-21
```

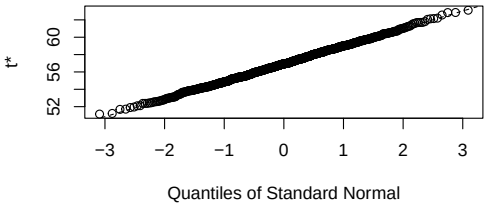
Confidence interval around β_0 : $[56.9001 \pm 2 \times 2.03002] \approx [52.84006, 60.96014]$

Confidence interval around β_1 : $[-0.46619 \pm 2 \times 0.0324241984] \approx [-0.53103, -0.40135]$

Confidence interval around β_2 : $[0.001231 \pm 2 \times 0.000117] \approx [0.000997, 0.001465]$

PLOT THE BOOTSTRAP SAMPLING AND GET THE CONFIDENCE INTERVALS






```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bootstrap, type = "norm")
##
## Intervals :
## Level      Normal
## 95%   (52.89, 60.84 )
## Calculations and Intervals on Original Scale
```

What Next?

You need to

- Read and understand the examples and code in Section 5.3 Lab: Cross-Validation and the Bootstrap from the textbook *"An Introduction to Statistical Learning: With Applications in R"*
- Solve question 5 in Exercise 5.4.
- Solve question 6 in Exercise 5.4.
- Solve question 8 in Exercise 5.4.