NBA 4920/6921 Lecture 9

Applying Hold-out Methods

Murat Unal

9/28/2021

Agenda

- Quiz 7
- Hold-out methods in R

1. Validation set approach
2. Leave-one-out cross validation
3. K-fold cross validation

```r
rm(list=ls())
options(digits = 3, scipen = 999)
library(tidyverse)
library(ISLR)
library(cowplot)
library(ggcorrplot)
library(stargazer)
library(corrr)
library(lmtest)
library(sandwich)
library(MASS)
library(car)
library(jtools)
library(caret)
library(leaps)
library(future.apply)
data <- read.csv("HousePricesTrain.csv")
```

Recall the purpose of hold-out methods:

We utilize them, e.g. cross validation, and use training data to estimate test performance

The idea is to estimate the test error by holding out part of the data set from training, and then applying the trained method on these held out observations

This way we can achieve two things:

1. Select the appropriate level of model flexibility
2. Assess model performance

Today we will see an application of the first.

Let's build a regression model to predict house prices.

We're going to use the following variables:

1. `id`: The sale's id.
2. `sale_price`: The sale price in tens of thousands
3. `age`: The age of the house at the time of the sale. The difference between `YrSold` and `YearBuilt`.
4. `area`: The non-basement square-foot area of the house.

```r
# Keep only the desired columns
data <- data %>% transmute(
    id = Id,
    sale_price = SalePrice/10000,
    age = YrSold - YearBuilt,
    area = GrLivArea)
```

Let's start by creating a test set by randomly drawing 10% from our entire data.

We will not touch this set until we have a final model.

We'll use `sample_frac()` from `dplyr`, to which we can pass the argument size—the percentage of the original data frame that we would like to end up in our training sample.

```
set.seed(1)
# Set aside 10% for testing
data.test <- data %>% sample_frac(size = 0.1)
data.train = setdiff(data, data.test)
```

Let's check the data

```
str(data.train)
```

```
'data.frame':	1314 obs. of  4 variables:
 $ id        : int  1 2 3 4 5 6 7 8 9 10 ...
 $ sale_price: num  20.9 18.1 22.4 14 25 ...
 $ age       : int  5 31 7 91 8 16 3 36 77 69 ...
 $ area      : int  1710 1262 1786 1717 2198 1362 1694 209
```

```r
stargazer(data.train,type="text",summary.stat=
            c("n","mean","median","min","max"))
```

```
=================================================
Statistic     N      Mean     Median   Min   Max
-------------------------------------------------
id          1,314  729.000    729      1   1,460
sale_price  1,314  18.100   16.300  3.490 75.500
age         1,314  36.700     36       0    135
area        1,314 1,515.000 1,462.0  334   5,642
-------------------------------------------------
```

Report the number of missing values in each column.

```
sapply(data.train, function(y) sum(is.na(y)))
```

```
        id sale_price        age       area
         0          0          0          0
```

Game plan

1. We will train regression models with varying complexity using the training data.
2. For each model we will compute the `MSE` on the validation data.
3. Decide on the final model flexibility by analyzing the validation `MSE`.
4. Estimate the final model with all training data and obtain predictions for test set.

We will fit models of the following form:

$$
\begin{aligned}
price &= \beta_0 + \beta_1 age + \beta_2 age^2 + \beta_3 area + \beta_4 area^2 \\
&= \beta_5 age * area + \beta_6 age^2 * area + \beta_7 age * area^2 \\
&= \beta_8 age^2 * area^2
\end{aligned}
$$

Validation set approach

Let's create a validation set by randomly drawing 20% from our training set.

```
# Draw validation set
validation_data = data.train %>% sample_frac(size = 0.2)
# Create the remaining training set
training_data = setdiff(data.train, validation_data)
```

Let's check everything makes sense:

```
dim(data.train)
```

```
[1] 1314    4
```

```
dim(validation_data)
```

```
[1] 263   4
```

```
dim(training_data)
```

```
[1] 1051    4
```

Let's write a function that does the analysis

```
# Our model-fit function
fit_model = function(deg_age, deg_area) {
    # Estimate the model using the training data
    est_model = lm(
      sale_price ~ poly(age, deg_age, raw = T) *
                   poly(area, deg_area, raw = T),
      data = training_data
    )
    # Make predictions on the validation data
    y_hat = predict(est_model,
                    newdata = validation_data,
                    se.fit = F)
    # Calculate our validation MSE
    mean((validation_data$sale_price - y_hat)^2)
}
```

Now we have our function ready, we need to loop over varying degrees of `age` and `area`.

Let's use up to 3 degrees for each variable.

We'll use `mapply`, which applies a function to multiple list or multiple vector arguments.

```r
# Take all possible combinations of our degrees
# from 1 to 3
deg_data = expand_grid(deg_age = 1:3, deg_area = 1:3)

# Iterate over set of possibilities (returns a vector
# of validation-set MSEs)
plan(multicore)
mse_v = future_mapply(
    FUN = fit_model,
    deg_age = deg_data$deg_age,
    deg_area = deg_data$deg_area)

# Add validation-set MSEs to 'deg_data'
deg_data$mse_v = mse_v
```

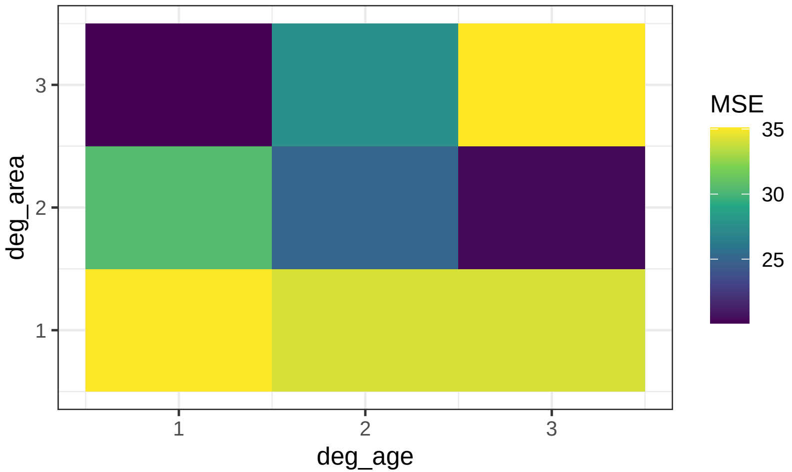
Which set of parameters minimizes validation-set MSE?

```
head(arrange(deg_data, mse_v))
```

deg_age	deg_area	mse_v
1	3	20.1
3	2	20.3
2	2	25.1
2	3	27.5
1	2	30.5
3	1	34.1

```
deg_age <- as.numeric(deg_data[which.min(deg_data$mse_v),
                               "deg_age"])
deg_area <- as.numeric(deg_data[which.min(deg_data$mse_v),
                                "deg_area"])
```

```r
ggplot(data = deg_data, aes(x = deg_age,
               y = deg_area,fill = mse_v)) +
geom_tile() +
scale_fill_viridis_c("MSE", option = "viridis") +
  theme_bw()
```



The validation exercise resulted in the optimal degrees of 1 and 3 for `age` and `area`, respectively.

Now, we would train the model with these degrees on the full data set and obtain the final model.

Once we have the final model, we could then use it to make predictions for the **unseen** test set.

```r
est_model = lm(
      sale_price ~ poly(age, deg_age, raw = T) *
                   poly(area, deg_area, raw = T),
      data = data.train
    )
# Make predictions on the test data
y_hat = predict(est_model,
                newdata = data.test,
                se.fit = F)
# Calculate our test MSE
val.test.MSE <- mean((data.test$sale_price - y_hat)^2)
val.test.MSE
```

```
[1] 14.9
```

Leave-one-out cross validation

Let's do the same exercise using LOOCV.

```
# Our model-fit function
loocv_model = function(deg_age, deg_area) {
    errors <- numeric(nrow(data.train))
    # Estimate the model using the training data
    for(each in 1:nrow(data.train)){
      train <- data.train[-each,]
      validate <- data.train[each,]
      est_model = lm(
      sale_price ~ poly(age, deg_age, raw = T) *
                   poly(area, deg_area, raw = T),
      data = train)
      # Make predictions on the left-out data
      y_hat = predict(est_model,
                      newdata = validate,
                      se.fit = F)
    # Calculate the SE
    errors[each] <-(validate$sale_price - y_hat)^2
    }
    mean(errors)
}
```

```r
# Take all possible combinations of our degrees
# from 1 to 3
deg_data = expand_grid(deg_age = 1:3, deg_area = 1:3)

# Iterate over set of possibilities (returns a vector
# of validation-set MSEs)
plan(multicore)
mse_v = future_mapply(
    FUN = loocv_model,
    deg_age = deg_data$deg_age,
    deg_area = deg_data$deg_area)
# Add validation-set MSEs to 'deg_data'
deg_data$mse_v = mse_v
```

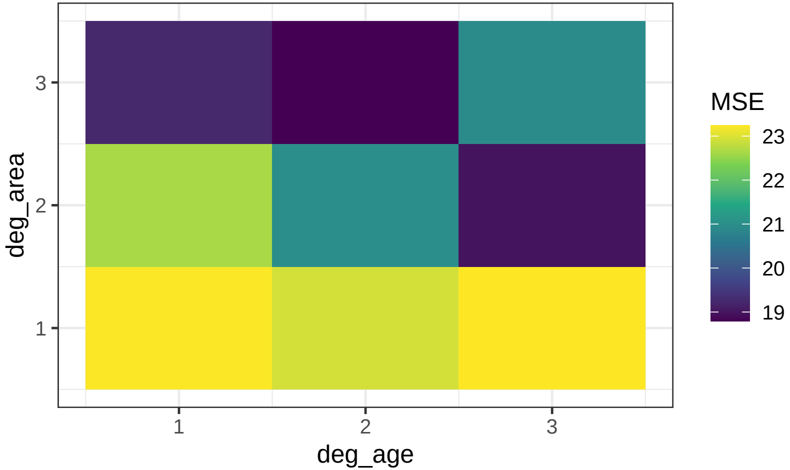
Which set of parameters minimizes LOOCV MSE?

```
head(arrange(deg_data, mse_v))
```

deg_age	deg_area	mse_v
2	3	18.8
3	2	19.0
1	3	19.2
3	3	20.9
2	2	21.0
1	2	22.6

```
deg_age <- as.numeric(deg_data[which.min(deg_data$mse_v),
                                "deg_age"])
deg_area <- as.numeric(deg_data[which.min(deg_data$mse_v),
                                 "deg_area"])
```

```r
ggplot(data = deg_data, aes(x = deg_age,
              y = deg_area,fill = mse_v)) +
geom_tile() +
scale_fill_viridis_c("MSE") +
  theme_bw()
```



The LOOCV exercise resulted in the optimal degrees of 2 and 3 for `age` and `area`, respectively.

Now, we would train the model with these degrees on the full data set and obtain the final model.

Once we have the final model, we could then use it to make predictions for the **unseen** test set.

Let's run the final model again, this time with the LOOCV optimal degrees.

```r
est_model = lm(
      sale_price ~ poly(age, deg_age, raw = T) *
                  poly(area, deg_area, raw = T),
      data = data.train
    )
# Make predictions on the test data
y_hat = predict(est_model,
                newdata = data.test,
                se.fit = F)
# Calculate our test MSE
LOOCV.test.MSE <- mean((data.test$sale_price - y_hat)^2)
LOOCV.test.MSE
```

```
[1] 14.8
```

K-fold cross validation

One more, this time let's use 10-fold CV.

```r
# Our model-fit function
fold_model = function(deg_age, deg_area) {
    # Number of folds
    nfold = 10
    # Create folds
    fold.list <- createFolds(rownames(data.train),nfold)
    # Empty vector to store the resulting MSEs
    MSE <- numeric(nfold)
```

Continued from previous...

```
    # Estimate the model using the training data
    for(each in 1:nfold){
      train <- data[-fold.list[[each]],]
      validate <- data.train[fold.list[[each]],]
      est_model = lm(
      sale_price ~ poly(age, deg_age, raw = T) *
                  poly(area, deg_area, raw = T),
      data = train)
      # Make predictions on the left-out data
      y_hat = predict(est_model,
              newdata = validate,
                 se.fit = F)
    # Calculate the SE
      MSE[each]<-sum((validate$sale_price-
                  y_hat)^2)/length(fold.list[[each]])
    }
    mean(MSE)
}
```

```r
# Take all possible combinations of our degrees
# from 1 to 3
deg_data = expand_grid(deg_age = 1:3, deg_area = 1:3)

# Iterate over set of possibilities (returns a vector
# of validation-set MSEs)
plan(multicore)
mse_v = future_mapply(future.seed=TRUE,
    FUN = fold_model,
    deg_age = deg_data$deg_age,
    deg_area = deg_data$deg_area)
# Add validation-set MSEs to 'deg_data'
deg_data$mse_v = mse_v
```

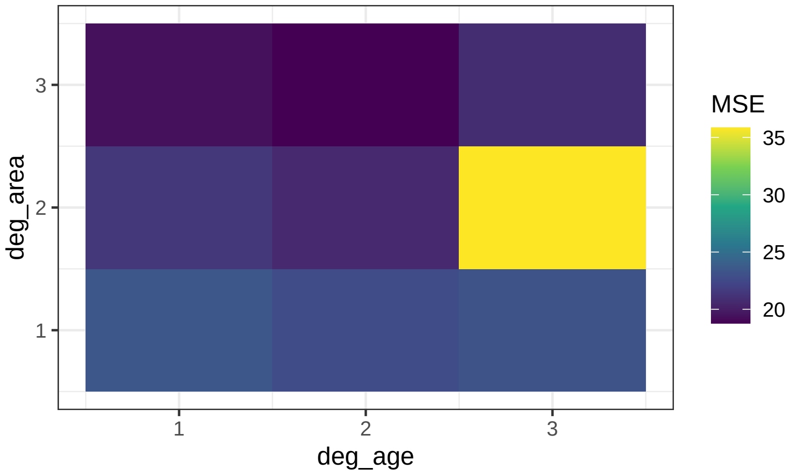
Which set of parameters minimizes CV MSE?

```r
head(arrange(deg_data, mse_v))
```

deg_age	deg_area	mse_v
2	3	18.8
1	3	19.3
2	2	20.6
3	3	20.8
1	2	21.4
2	1	22.7

```r
deg_age <- as.numeric(deg_data[which.min(deg_data$mse_v),
                               "deg_age"])
deg_area <- as.numeric(deg_data[which.min(deg_data$mse_v),
                                "deg_area"])
```

```
ggplot(data = deg_data, aes(x = deg_age,
                y = deg_area,fill = mse_v)) +
geom_tile() +
scale_fill_viridis_c("MSE") +
  theme_bw()
```



The K-fold CV exercise resulted in the optimal degrees of 2 and 3 for `age` and `area`, respectively.

Now, we would train the model with these degrees on the full data set and obtain the final model.

Once we have the final model, we could then use it to make predictions for the **unseen** test set.

Let's run the final model again, this time with the K-fold CV optimal degrees.

```r
est_model = lm(
      sale_price ~ poly(age, deg_age, raw = T) *
                   poly(area, deg_area, raw = T),
      data = data.train
    )
# Make predictions on the test data
y_hat = predict(est_model,
                newdata = data.test,
                se.fit = F)
# Calculate our test MSE
Kfold.test.MSE <- mean((data.test$sale_price - y_hat)^2)
Kfold.test.MSE
```

```
[1] 14.8
```

Compare test MSEs

```
val.test.MSE
```

```
[1] 14.9
```

```
LOOCV.test.MSE
```

```
[1] 14.8
```

```
Kfold.test.MSE
```

```
[1] 14.8
```

How would you apply this to a classification problem?