Cross-Validation Tutorial Appendics

Q. Chelsea Song, Chen Tang, and Serena Wee

# Appendix A

Simulation is used to further demonstrate Observations 1, 2 and 3. Specifically, as each sample draw is influenced by sampling variation, results from the Shiny app may not always be as expected (i.e., by chance, it is possible for the calibrated model to fit better in the validation sample than in the calibration sample). We conducted three simulations, to help illustrate the expected result for each of the three observations. In the simulations, each sample draw and model fit were repeated 1,000 times, then the results were obtained by averaging across the 1,000 trials. Unless otherwise stated, we assume a quadratic relationship between arousal and performance. The overall simulation results support the observations that the model generalizability decreases as (a) the model becomes more complex, (b) calibration sample size decreases, and (c) effect size decreases.

# Define a function to generate samples for simulation  
generate\_sample <- function(N, R2){  
 mean\_arousal <- 0  
 sd\_arousal <- 3  
 arousal <- rnorm(n = N, mean = mean\_arousal, sd = sd\_arousal)  
 performance <- (300 - (arousal - 6) ^ 2) / 60  
   
 # add noise to performance according to the value of R2 (R-squared)  
 snr <- R2 / (1 - R2) # calculate signal-to-noise ratio  
 # (using equation from Hastie et al. (2009); p.401)  
   
 sd\_performance <- 0.6364 # this sd is approximated using simulation  
 sd\_noise <- sd\_performance / sqrt(snr)  
 noise <- rnorm(n = N, sd = sd\_noise) # generate noise around 0  
 performance <- performance + noise  
 dat <- data.frame(arousal = arousal, performance = performance)  
 return(dat)  
}

## Observation 1: The Model Overfits the Calibration Sample

In the following code, the lm() function was used to fit the regression models.

# Set a random seed  
set.seed(8424)  
  
# Set population effect size  
R2\_pop <- .25  
   
# Set calibration sample size  
n\_cal <- 50  
   
# Set model complexity (degree of polynomial)  
degree <- 2  
   
# Set validation sample size  
n\_val <- 1000  
  
# According to this population R-squared value,  
# the population MSE can be computed as  
MSE\_pop <- 0.6364^2 / (R2\_pop / (1 - R2\_pop))  
# Population MSE is the sum of the squared residuals,  
# which is equivalent to the variance of noise  
# in the generate\_sample() function  
  
# Create vectors to store the calibration sample R-squared  
# and MSE for each trial  
R2 <- NA  
MSE <- NA  
  
for(i in 1:1000) {  
 # The following procedure is repeated 1000 times.  
 data <- generate\_sample(N=n\_cal, R2=R2\_pop)  
  
 # Fit the regression model to the sample and save   
 # the resulting model as “ob1\_mod”  
 ob1\_mod <- lm(performance ~ poly(arousal, degree), data=data)  
   
 # Store R-squared and MSE for each trial  
 R2[i] <- cor(ob1\_mod$fitted.values, data$performance)^2  
 MSE[i] <- mean(summary(ob1\_mod)$residuals^2)  
}  
  
# Calculate the average calibration R-squared and MSE,  
# averaged across 1000 trials  
R2\_cal <- mean(R2)  
MSE\_cal <- mean(MSE)  
  
# Output the results  
cat(paste0(  
 "R-squared of the population model: ", round(R2\_pop, 2), "\n",  
 "R-squared of the current model: ", round(R2\_cal, 2), "\n",  
 "MSE of the population model: ", round(MSE\_pop, 2), "\n",  
 "MSE of the current model: ", round(MSE\_cal, 2), "\n"))

## R-squared of the population model: 0.25  
## R-squared of the current model: 0.27  
## MSE of the population model: 1.22  
## MSE of the current model: 1.14

The output showed that the for the current model is larger than the population . Meanwhile, for the current model is smaller than the population . This is because the model is capturing sample-specific variations that is unrepresentative of the population. As a result, the model is overfitted to the calibration sample.

## Observation 2: The Model Obtained from the Calibration Sample Tends to Not Generalize Well to New (Validation) Samples

Apply the model obtained from the calibration sample to a new (validation) sample and check the prediction accuracy, and .

# Set a random seed  
set.seed(8424)  
   
# Set population effect size  
R2\_pop <- .25  
   
# Set calibration sample size  
n\_cal <- 50  
   
# Set model complexity (degree of polynomial)  
degree <- 2  
   
# Set validation sample size  
n\_val <- 1000  
  
# Create vectors to store the results for each trial  
R2\_cal <- NA  
MSE\_cal <- NA  
R2\_val <- NA  
MSE\_val <- NA  
  
for(i in 1:1000) {  
  
 # The following procedure is repeated 1000 times.  
 data\_cal <- generate\_sample(N=n\_cal, R2=R2\_pop)  
 data\_val <- generate\_sample(N=n\_val, R2=R2\_pop)  
 ob2\_mod <- lm(performance ~ poly(arousal, degree), data=data\_cal)  
 R2\_cal[i] <- cor(ob2\_mod$fitted.values, data\_cal$performance)^2  
 MSE\_cal[i] <- mean(summary(ob2\_mod)$residuals^2)  
  
 # Use the fitted regression model to predict task performance   
 # from arousal in the validation sample  
 yhat\_val <- predict(ob2\_mod, data\_val)  
  
 # Calculate validation R-squared and MSE  
 R2\_val[i] <- cor(yhat\_val, data\_val$performance)^2  
 MSE\_val[i] <- mean((yhat\_val - data\_val$performance)^2)  
}  
  
# Calculate the average validation sample R-squared and MSE,  
# averaged across 1000 trials  
R2\_cal\_mean <- mean(R2\_cal)  
R2\_val\_mean <- mean(R2\_val)  
MSE\_cal\_mean <- mean(MSE\_cal)  
MSE\_val\_mean <- mean(MSE\_val)  
  
# Output the results  
cat(paste0(  
 "Average Calibration R-squared: ", round(R2\_cal\_mean, 2), "\n",  
 "Average Validation R-squared: ", round(R2\_val\_mean, 2), "\n",  
 "Average Calibration MSE: ", round(MSE\_cal\_mean, 2), "\n",  
 "Average Validation MSE: ", round(MSE\_val\_mean, 2), "\n"))

## Average Calibration R-squared: 0.27  
## Average Validation R-squared: 0.23  
## Average Calibration MSE: 1.13  
## Average Validation MSE: 1.3

Notice that is smaller than : there is a 15% reduction in R2 (). Also, is larger than : there is a 0.17 increase in MSE (). This suggests that when a model fitted on one (calibration) sample is used to make predictions in a new (validation) sample, the model performs less well in the new sample.

## Observation 3: Model Generalizability is Influenced by (a) Model Complexity, (b) Sample Size, and (c) Effect Size

In Observation 3, we will vary (a) model complexity, (b) calibration sample size, and (c) effect size and compare the results with Observation 2.

### Observation 3a: The model generalizes less well when the model is complex

Now let us explore how model complexity influences model generalizability. Increase the model complexity by specifying a cubic (instead of quadratic) regression model. Then, apply the calibrated models to new samples and obtain prediction accuracy.

# Set a random seed  
set.seed(8424)  
  
# Set population effect size  
R2\_pop <- .25  
   
# Set calibration sample size  
n\_cal <- 50  
   
# Set model complexity (degree of polynomial)  
degree <- 3 # Cubic regression, a more complex model  
   
# Set validation sample size  
n\_val <- 1000  
  
# Create vectors to store the calibration  
# and validation sample R-squared for each trial  
R2\_cal <- NA  
MSE\_cal <- NA  
R2\_val <- NA  
MSE\_val <- NA  
  
for(i in 1:1000) {  
   
 # The following procedures are repeated 1000 times  
 data\_cal <- generate\_sample(N=n\_cal, R2=R2\_pop)  
 data\_val <- generate\_sample(N=n\_val, R2=R2\_pop)  
 ob3a\_mod <- lm(performance ~ poly(arousal, degree), data=data\_cal)  
 R2\_cal[i] <- cor(ob3a\_mod$fitted.values, data\_cal$performance)^2  
 MSE\_cal[i] <- mean(summary(ob3a\_mod)$residuals^2)  
 # Use the fitted regression model to predict task performance   
 # from arousal in the validation sample  
 yhat\_val <- predict(ob3a\_mod, data\_val)  
   
 # Calculate validation R-squared and MSE  
 R2\_val[i] <- cor(yhat\_val, data\_val$performance)^2  
 MSE\_val[i] <- mean((yhat\_val - data\_val$performance)^2)  
}  
  
# Calculate the average validation sample R-squared,  
# averaged across 1000 trials  
R2\_cal\_mean <- mean(R2\_cal)  
R2\_val\_mean <- mean(R2\_val)  
MSE\_cal\_mean <- mean(MSE\_cal)  
MSE\_val\_mean <- mean(MSE\_val)  
  
# Output the results  
cat(paste0(  
 "Average Calibration R-squared: ", round(R2\_cal\_mean, 2), "\n",  
 "Average Validation R-squared: ", round(R2\_val\_mean, 2), "\n",  
 "Average Calibration MSE: ", round(MSE\_cal\_mean, 2), "\n",  
 "Average Validation MSE: ", round(MSE\_val\_mean, 2), "\n"))

## Average Calibration R-squared: 0.29  
## Average Validation R-squared: 0.2  
## Average Calibration MSE: 1.1  
## Average Validation MSE: 1.43

Across 1,000 trials, is smaller for the cubic regression model () than for the quadratic regression model (, see Observation 2). Similarly, is larger for the cubic regression model () than for the quadratic regression model (, see Observation 2). Consistent with Observation 3a, a complex model generalizes less well than a simpler model when applied to a new sample.

### Observation 3b: The model generalizes less well when calibration sample size is small

Examine model generalizability for a smaller calibration sample size of 30 (as compared to 50). Apply the model obtained from the calibration sample to new samples (validation samples) and check the magnitude of prediction accuracy.

# Set a random seed  
set.seed(8424)  
  
# Set population effect size  
R2\_pop <- .25  
   
# Set calibration sample size  
n\_cal <- 30 # smaller sample size  
   
# Set model complexity (degree of polynomial)  
degree <- 2  
   
# Set validation sample size  
n\_val <- 1000  
   
# Create vectors to store the calibration  
# and validation sample R-squared and MSE for each trial  
R2\_cal <- NA  
MSE\_cal <- NA  
R2\_val <- NA  
MSE\_val <- NA  
  
for(i in 1:1000) {  
   
 # The following procedures are repeated 1000 times.  
 data\_cal <- generate\_sample(N=n\_cal, R2=R2\_pop)  
 data\_val <- generate\_sample(N=n\_val, R2=R2\_pop)  
 ob3b\_mod <- lm(performance ~ poly(arousal, degree), data=data\_cal)  
 R2\_cal[i] <- cor(ob3b\_mod$fitted.values, data\_cal$performance)^2  
 MSE\_cal[i] <- mean(summary(ob3b\_mod)$residuals^2)  
  
 # Use the fitted regression model to predict task performance   
 # from arousal in the validation sample  
 yhat\_val <- predict(ob3b\_mod, data\_val)  
   
 # Calculate validation R-squared and MSE  
 R2\_val[i] <- cor(yhat\_val, data\_val$performance)^2  
 MSE\_val[i] <- mean((yhat\_val - data\_val$performance)^2)  
}  
  
# Calculate the average calibration and validation sample R-squared and MSE,  
# averaged across 1000 trials  
R2\_cal\_mean <- mean(R2\_cal)  
R2\_val\_mean <- mean(R2\_val)  
MSE\_cal\_mean <- mean(MSE\_cal)  
MSE\_val\_mean <- mean(MSE\_val)  
  
# Output the results  
cat(paste0(  
 "Average Calibration R-squared: ", round(R2\_cal\_mean, 2), "\n",  
 "Average Validation R-squared: ", round(R2\_val\_mean, 2), "\n",  
 "Average Calibration MSE: ", round(MSE\_cal\_mean, 2), "\n",  
 "Average Validation MSE: ", round(MSE\_val\_mean, 2), "\n"))

## Average Calibration R-squared: 0.29  
## Average Validation R-squared: 0.21  
## Average Calibration MSE: 1.09  
## Average Validation MSE: 1.38

Notice that, across 1,000 trials, the average is smaller when the same quadratic regression model was fitted on a small sample size (), as compared to when the quadratic regression model was fitted on a larger sample size (; average , see Observation 2). Similarly, the average is larger when the same quadratic regression model was fitted on a small sample size (), as compared to when the quadratic regression model was fitted on a larger sample size (; average , see Observation 2). This suggests that a model fitted on a smaller calibration sample tends to generalize less well in a new sample, as compared to a model fitted on a larger calibration sample.

### Observation 3c: The model generalizes less well when the population effect size is small

Examine model generalizability when the population effect size is (as compared to ). Apply the model obtained from the calibration sample to new samples and check the magnitude of prediction accuracy.

# Set a random seed  
set.seed(8424)  
  
# Set population effect size  
R2\_pop <- .04 # smaller effect size  
   
# Set calibration sample size  
n\_cal <- 50  
   
# Set model complexity (degree of polynomial)  
degree <- 2  
   
# Set validation sample size  
n\_val <- 1000  
  
# Create vectors to store the calibration  
# and validation sample R-squared for each trial  
R2\_cal <- NA  
MSE\_cal <- NA  
R2\_val <- NA  
MSE\_val <- NA  
  
for(i in 1:1000) {  
   
 # The following procedures are repeated 1000 times.  
 data\_cal <- generate\_sample(N=n\_cal, R2=R2\_pop)  
 data\_val <- generate\_sample(N=n\_val, R2=R2\_pop)  
 ob3c\_mod <- lm(performance ~ poly(arousal, degree), data=data\_cal)  
 R2\_cal[i] <- cor(ob3c\_mod$fitted.values, data\_cal$performance)^2  
 MSE\_cal[i] <- mean(summary(ob3c\_mod)$residuals^2)  
   
 # Use the fitted regression model to predict task performance   
 # from arousal in the validation sample  
 yhat\_val <- predict(ob3c\_mod, data\_val)  
   
 # Calculate validation R-squared and MSE  
 R2\_val[i] <- cor(yhat\_val, data\_val$performance)^2  
 MSE\_val[i] <- mean((yhat\_val - data\_val$performance)^2)  
}  
  
# Calculate the average validation sample R-squared,  
# averaged across 1000 trials  
R2\_cal\_mean <- mean(R2\_cal)  
R2\_val\_mean <- mean(R2\_val)  
MSE\_cal\_mean <- mean(MSE\_cal)  
MSE\_val\_mean <- mean(MSE\_val)  
  
# Output the results  
cat(paste0(  
 "Average Calibration R-squared: ", round(R2\_cal\_mean, 2), "\n",  
 "Average Validation R-squared: ", round(R2\_val\_mean, 2), "\n",  
 "Average Calibration MSE: ", round(MSE\_cal\_mean, 2), "\n",  
 "Average Validation MSE: ", round(MSE\_val\_mean, 2), "\n"))

## Average Calibration R-squared: 0.08  
## Average Validation R-squared: 0.03  
## Average Calibration MSE: 9.03  
## Average Validation MSE: 10.42

Notice that, when the population effect size is .04, across 1,000 trials, the average is less than half the average (63% reduction versus 15% reduction in Observation 2), and the average is less than the average (1.39 increase versus 0.17 increase in Observation 2). This suggests that, when the population effect size is small, the model tend to generalize less well in a new sample.

# Appendix B

## *k*-Fold cross-validation

The R code below provides a step-by-step demonstration of a 5-fold cross-validation process.

# Set random seed  
set.seed(8424)  
  
# Sample size  
n <- 300  
  
# Set population effect size  
R2\_pop <- .16  
  
cv\_data <- generate\_sample(N=n, R2=R2\_pop)  
  
# Specify a cubic regression model  
degree <- 3  
  
# Fit the cubic regression model using  
# the complete dataset and save the resulting model as “cv\_original\_mod”  
cv\_original\_mod <- lm(performance ~ poly(arousal, degree),  
 data=cv\_data)  
  
# Obtain R-squared and MSE of the model fitted to the complete dataset  
R2 <- cor(cv\_original\_mod$fitted.values, cv\_data$performance)^2  
MSE <- mean(summary(cv\_original\_mod)$residuals^2)  
  
# Specify the number of folds  
k <- 5  
  
# Split the index of each observation into k equal subsets  
subsets <- split(x=1:n, f=sort(rep\_len(x=1:k, length.out=n)))  
  
# Iterate through each fold  
R2\_test <- NA  
MSE\_test <- NA  
  
for(i in 1:k) {  
  
 # Split the dataset into k equal subsets  
 ind\_test <- subsets[[i]]  
   
 # For example, when i=1, the first subset of data  
 # (1/5 of the original dataset) is specified as the test set  
 data\_test <- cv\_data[ind\_test, ]  
   
 # Specify the rest of the data (4/5 of the original data)  
 # as the training set  
 data\_train <- cv\_data[-ind\_test, ]  
   
 # Fit the cubic regression model to the training set  
 mod\_tmp <- lm(performance ~ poly(arousal, degree), data=data\_train)  
   
 # Test the model on the test set and calculate  
 # R-squared and MSE for this fold (iteration)  
 yhat\_test <- predict(mod\_tmp, data\_test)  
 R2\_test[i] <- cor(yhat\_test, data\_test$performance)^2  
 MSE\_test[i] <- mean((yhat\_test - data\_test$performance)^2)  
}  
  
# Average Cross-validated R-squared and MSE (averaged across 5 folds)  
R2\_kfold <- mean(R2\_test)  
MSE\_kfold <- mean(MSE\_test)  
  
# Output the results  
cat(paste0(  
 "Model R-squared: ", round(R2, 2), "\n",  
 "Model MSE: ", round(MSE, 2), "\n",  
 "Cross-validated R-squared: ", round(R2\_kfold, 2), "\n",  
 "Cross-validated MSE: ", round(MSE\_kfold, 2), "\n"))

## Model R-squared: 0.2  
## Model MSE: 2.21  
## Cross-validated R-squared: 0.18  
## Cross-validated MSE: 2.3

## Monte Carlo Cross-Validation (MCCV)

The R code below provides a step-by-step demonstration of a Monte-Carlo cross-validation process. In this example, we use 80% of the sample to train the model and the remaining 20% to test the model. This is repeated 100 times to obtain an estimate of the cross-validated and .

# Using the same dataset and model as the above k-fold example  
  
set.seed(8424)  
  
# Specify to repeat 100 times  
rep <- 100  
  
# Set the test set size to 1/5 (20%) of the total sample size  
n\_test <- ceiling(n / 5) # n\_test = 60 for sample size 300  
  
R2\_test <- NA  
MSE\_test <- NA  
  
for(i in 1:rep) {  
   
 # Repeat 100 times, each time randomly drawing 60 observations  
 # as the test set and the rest as the training set  
  
 ind\_test <- sample(1:n, n\_test) # randomly draw 60 numbers from 1 to 300  
   
 # use the 60 numbers draw as row indices and get the test set  
 data\_test <- cv\_data[ind\_test, ]  
   
 # Specify the remaining 240 observations as the training set  
 data\_train <- cv\_data[-ind\_test, ]  
   
 # Fit the model with the training set   
 mod\_temp <- lm(performance ~ poly(arousal, degree), data=data\_train)  
   
 # Use the fitted model to make predictions in test set  
 yhat\_test <- predict(mod\_temp, data\_test)  
   
 # Save cross-validated R-squared and MSE for each repetition  
 R2\_test[i] <- cor(yhat\_test, data\_test$performance)^2  
 MSE\_test[i] <- mean((yhat\_test - data\_test$performance)^2)   
}  
  
# Average cross-validated R-squared (averaged across 100 repetitions)  
R2\_mccv <- mean(R2\_test)  
MSE\_mccv <- mean(MSE\_test)  
  
# Output the results  
cat(paste0(  
 "Model R-squared: ", round(R2, 2), "\n",  
 "Model MSE: ", round(MSE, 2), "\n",  
 "Cross-validated R-squared: ", round(R2\_mccv, 2), "\n",  
 "Cross-validated MSE: ", round(MSE\_mccv, 2), "\n"))

## Model R-squared: 0.2  
## Model MSE: 2.21  
## Cross-validated R-squared: 0.18  
## Cross-validated MSE: 2.28

# Appendix C

# Empirical Example using Machiavellianism Dataset

In this example, we treat the 70,798 observations as the population and fit a regression model to obtain the population estimates (see R code below). As a result, in the population, , and .

# If the caret package is not installed run below line to install it  
# install.packages("caret")  
  
# Load the package  
library(caret)  
  
# The following dataset (from https://openpsychometrics.org/\_rawdata/)  
# is used by the current overfitting and cross-validation example.  
# R Code for data preparation is available from the authors.  
  
# Import data  
# data <- read.csv("https://git.io/JfsiA")  
  
  
data <- read.csv("data/mach\_data\_all.csv")  
  
# f <- as.formula("mach ~ as.factor(gender) + age + as.factor(education) +   
# as.factor(urban) + as.factor(engnat) +   
# as.factor(religion) + as.factor(orientation) +   
# as.factor(race) + as.factor(voted) + as.factor(married) +   
# O + C + E + A + N")  
  
f <- as.formula("mach ~ age + as.factor(gender) + as.factor(education) + as.factor(engnat) +  
 as.factor(married) + O + C + E + A + N")  
  
# Fit the original model on the complete dataset  
mod <- lm(formula=f, data=data)  
  
# Summary of the regression model fitted on the complete dataset  
R2\_pop <- round(summary(mod)$r.squared, 2)  
MSE\_pop <- round(mean(mod$residuals^2), 2)  
cat(paste0("R2 of the population model is ", R2\_pop, ";\n",  
 "MSE of the population model is ", MSE\_pop, "."))

## R2 of the population model is 0.28;  
## MSE of the population model is 0.45.

### Model Overfitting

Now we randomly draw a sample from this population and fit the same model.

# sample\_cal <- read.csv("https://git.io/JfsPt")  
# This is a random sample drawn from the whole Mach dataset using below code.  
# Interested readers can vary the random seed to test with different samples.  
set.seed(8424)  
n <- 150  
sample\_cal <- data[sample(nrow(data), n), ]  
  
# mod\_cal <- lm(mach ~ as.factor(gender) + age + O + C + E + A + N,  
# data=sample\_cal)  
  
mod\_cal <- lm(f, data=sample\_cal)  
  
# Calculate and save results  
R2\_cal <- round(summary(mod\_cal)$r.squared, 2)  
MSE\_cal <- round(mean(mod\_cal$residuals^2), 2)  
  
# Print results  
cat(paste0("R2 of the calibration model is ", R2\_cal, ";\n",  
 "MSE of the calibration model is ", MSE\_cal, "."))

## R2 of the calibration model is 0.38;  
## MSE of the calibration model is 0.34.

We notice that in the sample (), the regression model has a larger and a smaller than the population model.

### Cross-Validation

#### *k*-fold

We will carry out a 10-fold cross-validation. First, call the trainControl() function, and specify the method as *k*-fold cross-validation (i.e., “CV”), where the number of folds is equal to 10. Save the specified information in an object named kfold\_train\_control. Then, call the train() function to implement the k-fold cross-validation. The cross-validation results are saved in the object, “kfold\_cv”.

set.seed(8424) # Set a random seed to replicate results  
  
k <- 10 # Number of folds  
  
kfold\_train\_control <- trainControl(method="cv", number=k)  
  
# kfold\_cv <- train(mach ~ as.factor(gender) + age + O + C + E + A + N,  
# data=sample\_cal, method="lm", trControl=kfold\_train\_control)  
  
kfold\_cv <- train(f, data=sample\_cal, method="lm", trControl=kfold\_train\_control)  
  
# Calculate and save results  
MSE\_kfold <- round(mean(kfold\_cv$resample$RMSE^2), 2)  
R2\_kfold <- round(kfold\_cv$results$Rsquared, 2)  
  
# Print results  
cat(paste0("k-fold cross-validated R2 is ", R2\_kfold, ";\n",  
 "k-fold cross-validated MSE is ", MSE\_kfold, "."))

## k-fold cross-validated R2 is 0.34;  
## k-fold cross-validated MSE is 0.41.

It looks like there is a lot going on in this code, so let us talk about each of the input arguments (e.g., data, method, trControl) that are being specified. First, data = sample\_cal specifies that the sample\_cal dataset, the dataset that was used in the original analysis, is used to conduct the cross-validation. Next, method = “lm” specifies the statistical model as we need to indicate the model that will be used. In this case, we are using a linear (regression) model, and so we specify that method = “lm” and provide the actual form of the regression model that should be fitted. This is exactly the same model as the one fitted earlier. Lastly, to specify the cross-validation method to be used, trControl = kfold\_train\_control tells the function that should correspond to the 10-fold cross-validation method that we specified earlier.

All the results are saved in the object called kfold\_cv, we retrieved and from the kfold\_cv object. Note that caret calculates by default, so is calculated as the mean of squares the ’s for all folds.

#### Monte Carlo cross-validation (MCCV)

To conduct MCCV, we specify that method = “LGOCV” (i.e., leave-group-out cross-validation, which is another term for Monte-Carlo cross-validation) in the trainControl() function. We set the number of repetitions to be 200 (number = 200; i.e., we ask for the train-then-test procedure to be conducted 200 times). In addition, we specify the proportion of data that should be randomly held out as a test set in each of the 200 repetitions. For example, p = .8 means that 75% of the data set will be used as the *training* set, and therefore 40% (i.e., 1 - p) of the data set will be used as the *test* set. These pre-specifications will once again be saved in an object named mc\_train\_control.

Next, the train() function will again be used to implement MCCV. As with conducting *k*-fold cross-validation using train(), we specify the data, analytic method, and linear regression equation to be used when conducting the MCCV. Similar to the *k*-fold example, results are retrieved and calculated from the final mc\_cv object.

set.seed(8424) # Set a random seed to replicate results  
  
R <- 200 # Number of repetitions  
  
mc\_train\_control <- trainControl(method="LGOCV", p=.8, number=R)  
  
# mc\_cv <- train(mach ~ as.factor(gender) \* age + O + C + E + A + N,  
# data=sample\_cal, method="lm", trControl=mc\_train\_control)  
  
mc\_cv <- train(f, data=sample\_cal, method="lm", trControl=mc\_train\_control)  
  
# Calculate and save results  
MSE\_mc <- round(mean(mc\_cv$resample$RMSE^2), 2)  
R2\_mc <- round(mc\_cv$results$Rsquared, 2)  
  
# Print results  
cat(paste0("Monte Carlo cross-validated R2 is ", R2\_mc, ";\n",  
 "Monte Carlo cross-validated MSE is ", MSE\_mc, "."))

## Monte Carlo cross-validated R2 is 0.28;  
## Monte Carlo cross-validated MSE is 0.44.