Exercise 6 - Machine Learning and Cross Validation

O’Malley, Conie

2025-02-05

Table of Contents

## Submission:

Download the **Ex6\_ML&CV\_YourLastName.Rmd** R Markdown file and save it with **your own last name**. Note, it is important that you conform to this file naming convention, otherwise it will affect the sorting of assignments for us. Please name your file as instructed and complete all your work in that template file, **Knit** the corresponding Word or PDF file. Your knitted file **MUST display your R commands**. In order to do this, please ensure that the knittr:: global option command above has echo=T.

**Important Technical Note:** R discovered problems with the sample() function, caused by some inconsistencies in the set.seed() **random number generator (RNG)**. They corrected this with R version 3.6.0. However, your results may be different than mine, depending on what is the default RNG in your R installation. The differences are usually due to rounding differences in randomly generated values. If you are working on your own project, this doesn’t affect you because your own results will be consistent. But for your results to match mine, we all need to set the RNG to the same default. Still, your numbers may not exactly match mine and that is OK, as long as they are not radically different. But let’s all set the same RNG default this way:

RNGkind(sample.kind="default")

## 1. Random Splitting (Holdout Sample) Cross Validation (RSCV)

**1.1 Train Vector.**

Load the **{MASS}** library, which contains the **Boston** Boston dataset. Run ?Boston from the console (not in the script) and inspect its variables.

Then use set.seed(1) so that you get the same results if you run your cross validation commands multiple times. Display the number of rows in the **Boston** dataset using the nrow() function. Then use the sample() and nrow() functions to create an index vector called **train** which you will later use to split the data set into **70%** train subsample. Then display the train vector.

library(MASS)  
set.seed(1) # set the random seed for CV reproducability   
nrow(Boston) # get the number of rows in the Boston dataset

## [1] 506

train <- sample(1:nrow(Boston), 0.7 \* nrow(Boston)) # create training data vector  
train # display the training vector

## [1] 505 324 167 129 418 471 299 270 466 187 307 481 85 277 362 438 330 263  
## [19] 329 79 213 37 105 217 366 165 290 492 382 89 428 463 289 340 419 326  
## [37] 490 42 422 111 404 412 20 44 377 343 70 121 40 172 25 375 248 198  
## [55] 378 39 435 298 390 280 160 14 130 45 402 22 206 230 193 371 104 501  
## [73] 255 450 436 103 331 13 296 483 176 345 279 110 84 359 29 141 252 406  
## [91] 221 465 108 304 33 443 149 287 102 145 488 461 339 118 346 413 107 64  
## [109] 224 431 316 51 416 480 138 503 500 282 143 285 170 48 204 295 24 181  
## [127] 214 476 225 498 442 163 43 1 420 78 433 284 116 233 293 61 86 327  
## [145] 423 355 496 300 49 396 242 246 305 306 247 239 219 135 467 464 395 53  
## [163] 444 401 65 421 484 124 77 218 98 194 19 273 31 174 237 75 16 458  
## [181] 265 353 92 122 152 392 207 249 446 229 140 126 445 368 328 271 344 342  
## [199] 333 212 127 133 41 36 297 399 391 360 117 408 50 286 254 72 437 168  
## [217] 313 113 234 459 73 27 405 15 62 132 35 338 473 185 153 332 485 434  
## [235] 231 28 389 148 325 60 275 93 202 336 241 415 281 449 364 427 478 274  
## [253] 414 283 314 351 209 251 82 97 398 482 425 56 288 272 199 350 235 178  
## [271] 201 269 489 101 370 417 2 393 131 348 166 253 156 183 197 220 440 495  
## [289] 374 250 373 173 352 83 151 68 208 493 397 268 302 180 162 455 74 453  
## [307] 188 504 358 190 161 447 139 472 470 267 90 222 112 291 384 211 243 456  
## [325] 226 81 383 215 205 150 432 3 147 196 128 320 4 191 309 354 106 195  
## [343] 409 23 157 259 319 119 99 238 177 164 258 294

**1.2 Train and Test Subsamples.**

Then create a train subsample using the train **vector** above as the index and all columns (i.e., Boston[train,]). Name this subsample **Boston.train**. Just to be certain, count and display the number of rows of Boston.train. Then create a subsample named **Boston.test** using the remaining observations, and also count and display the number of rows.

Boston.train <- Boston[train,] # subset the training data  
nrow(Boston.train) # count rows of train subsample

## [1] 354

Boston.test <- Boston[-train,] # subset the test data   
nrow(Boston.test) # count rows of test subsample

## [1] 152

**1.3 Train the Model.**

Fit a linear model to predict the median value of houses in Boston counties **medv** using the **train subsample**. Use crim+chas+rm+age+tax+ptratio+lstat as predictors. Store your resulting model in an object named **fit.train** and display the summary() results. Quick note: the dataset Boston is capitalized and the response variable salary is not, so be sure to spell them accordingly.

# fit a linear model  
fit.train <- lm(medv ~ crim + chas + rm + age + tax + ptratio + lstat, Boston.train)  
summary(fit.train) # view summary statistics of fit

##   
## Call:  
## lm(formula = medv ~ crim + chas + rm + age + tax + ptratio +   
## lstat, data = Boston.train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -12.2149 -3.0275 -0.6874 1.9967 25.9513   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 9.550306 4.692502 2.035 0.0426 \*   
## crim -0.027680 0.042756 -0.647 0.5178   
## chas 4.273020 0.996441 4.288 2.34e-05 \*\*\*  
## rm 5.518057 0.504357 10.941 < 2e-16 \*\*\*  
## age 0.006894 0.012565 0.549 0.5836   
## tax -0.002928 0.002282 -1.283 0.2002   
## ptratio -0.803964 0.146131 -5.502 7.33e-08 \*\*\*  
## lstat -0.494596 0.065036 -7.605 2.72e-13 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 4.991 on 346 degrees of freedom  
## Multiple R-squared: 0.7329, Adjusted R-squared: 0.7275   
## F-statistic: 135.6 on 7 and 346 DF, p-value: < 2.2e-16

**1.4 Test MSE and RMSE**

Compute the **Random Split Cross-Validation Test MSE** using **fit.train** as the prediction model and **Boston.test** as the test subset. Store the resulting MSE in an object named **test.mse.rs** and display the result.

Tip: the best thing is to build the **test.mse.rs** formula in steps. First type Boston.test$medv, which is a vector with the actual **medv** values from the test subsample. Then subtract the results from predict() using **fit.train** as the prediction model and **Boston test** as the prediction test subset. This subtraction will yield a vector with all the prediction errors. You can view it the results in the console (not in the script). You can then square the results, which will yield a vector with all the squared errors. Then, take the mean() of all that to get the **test.mse.rs**.

Once you have calculated the **test.mse** use the c() function to display a vector with the **test.mse.rs** and the sqrt(test.mse.rs). Label the two vector elements **“RSCV Test MSE”** and **“RSCV Test RMSE”**.

test.mse.rs <- mean((Boston.test$medv - predict(fit.train, Boston.test))^2) # calculate RMSE  
c("RSCV Test MSE" = test.mse.rs, "RSCV Test RMSE" = sqrt(test.mse.rs)) # concatenate results

## RSCV Test MSE RSCV Test RMSE   
## 31.593799 5.620836

**1.5 Discussion.**

Briefly comment on these results. The Test MSE is difficult to interpret, which is why we computed the Test RMSE. In one sentence, what does the Test RMSE result tell you?

#### Analysis

The RMSE in this context indicates that on average, our model’s predictions are off by approximately $4,500 when compared to the actual median value of homes in the Boston dataset. This suggests that while the model is reasonably accurate, there may still be room for improvement in terms of capturing the nuances and complexities of the data.

## 2. Leave One Out Cross-Validation (LOOCV)

**2.1 Fit GLM Model.**

Using the **Boston** data set, fit a **GLM** model to predict **medv** using crim+chas+rm+age+tax+ptratio+lstat as predictors. Display the summary results. Store the results in an object named **glm.fit**. Tip: when you use the glm() function you need to specify the family and the link function. In this case, however, you are fitting an OLS model with the glm() function. As we discussed earlier in the semester, Since **gaussian** is the default distribution, you don’t need to specify a familty or a link. Just specify the model as if you were using the lm() function and the result will be an OLS model.Then display the summary() of the **glm.fit** model.

# fit a glm model  
glm.fit <- glm(medv ~ crim + chas + rm + age + tax + ptratio + lstat, data = Boston)  
summary(glm.fit) # view summary statistics

##   
## Call:  
## glm(formula = medv ~ crim + chas + rm + age + tax + ptratio +   
## lstat, data = Boston)  
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 16.753143 3.958253 4.232 2.75e-05 \*\*\*  
## crim -0.053497 0.033552 -1.594 0.111463   
## chas 3.222183 0.920203 3.502 0.000504 \*\*\*  
## rm 4.435494 0.431602 10.277 < 2e-16 \*\*\*  
## age 0.015144 0.010899 1.389 0.165333   
## tax -0.001721 0.001961 -0.878 0.380436   
## ptratio -0.828633 0.124067 -6.679 6.45e-11 \*\*\*  
## lstat -0.566526 0.053750 -10.540 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for gaussian family taken to be 26.4633)  
##   
## Null deviance: 42716 on 505 degrees of freedom  
## Residual deviance: 13179 on 498 degrees of freedom  
## AIC: 3103.4  
##   
## Number of Fisher Scoring iterations: 2

**2.2 Leave On Out (LOOCV)**

Load the **{boot}** library and use the cv.glm() function and the **glm.fit** object above to compute the **LOOCV MSE** (Leave One Out) for this model (the first element $delta[1]) contains the raw MSE. Store the result in an object named **test.mse.loo**. Then, as you did above, use the c() function to display a vector with the **test.mse.loo** and the sqrt(test.mse.loo). Label the two vector elements “LOOCV Test MSE” and “LOOCV Test RMSE”.

test.mse.loo <- boot::cv.glm(Boston, glm.fit)$delta[1] # fit a CV glm  
# display LOOCV test mse and rmse  
c("LOOCV Test MSE" = test.mse.loo, "LOOCV Test RMSE" = sqrt(test.mse.loo))

## LOOCV Test MSE LOOCV Test RMSE   
## 27.679679 5.261148

## 3. K-Fold Validation (KFCV)

**3.1 10-Fold Validation (10FCV)**

Using the same **cv.glm()** function and **glm.fit** model object, compute and display the **10-Fold** cross validation MSE for this model. Store the result in an object named **test.mse.10f**. Then, as you did above, use the c() function to display a vector with the **test.mse.10f** and the sqrt(test.mse.10f). Label the two vector elements “10FCV Test MSE” and “10FCV Test RMSE”

test.mse.10f <- boot::cv.glm(Boston, glm.fit, K = 10)$delta[1] # fit a CV glm  
# display 10FCV test mse and rmse  
c("10FCV Test MSE" = test.mse.10f, "10FCV Test RMSE" = sqrt(test.mse.10f))

## 10FCV Test MSE 10FCV Test RMSE   
## 28.022003 5.293581

**3.2 Compare Results**

Display a table (data frame) that shows the Test MSE and Test RMSE in two columns for each of the 3 cross-validation methods. Refer to the table in the solution below and use these functions: cbind() to bind the respective table columns together; and rbind() to bind the rows. Store the resulting dataframe as **mse.all**. Then use colnames() to give labels to the columns of **mse.all** (use c() to create a vector with the respective column names); and rownames() to give labels to the rows (use c() to create a vector with the respective row names). Use the column and row labels shown in the output below. Then display **mse.all**.

# create a data frame of results  
mse.all <- rbind(cbind(test.mse.rs, sqrt(test.mse.rs)),   
 cbind(test.mse.loo, sqrt(test.mse.loo)),   
 cbind(test.mse.10f, sqrt(test.mse.10f)))  
rownames(mse.all) <- c("RSCV", "LOOCV", "10FCV") # change row names  
colnames(mse.all) <- c("MSE", "RMSE") # change column names  
mse.all

## MSE RMSE  
## RSCV 31.59380 5.620836  
## LOOCV 27.67968 5.261148  
## 10FCV 28.02200 5.293581

**2.5 Commentary**.

Provide a brief commentary of the results above. Is there a meaning to the difference between these 3 MSE Cross-Validation result? Briefly explain

#### Analysis

The differences in the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) across different cross-validation methods (RSCV, LOOCV, 10FCV) can be interpreted as follows: RSCV provides a balance between bias and variance, while LOOCV tends to overfit the model due to its high number of folds. On the other hand, 10FCV strikes a compromise by using a moderate number of folds, providing a good trade-off between computational efficiency and model performance. The lower MSE and RMSE values indicate that the models have performed well in terms of prediction accuracy across all cross-validation methods. All things considered, the models are very similar with a minimal difference in RMSE amongst them (~5% difference).