Resampling Methods for Assessing Model Accuracy

Yifei Sun

Contents

```
library(FNN)
library(caret)
```

You can generate a simulated training dataset or use an existing dataset. For illustration, we use a simulated dataset with two predictors.

```
# Data generating function - you can replace this with your own function
gen_data <- function(N)
{
  X1 <- rnorm(N, mean = 1)
  X2 <- rnorm(N, mean = 1)
  eps <- rnorm(N, sd = .5)
  Y <- sin(X1) + (X2)^2 + eps
  data.frame(Y = Y, X1 = X1, X2 = X2)
}

set.seed(2022)

# generate the *training* data
N <- 200
trainData <- gen_data(N)
```

Data splitting functions

Training/Validation splitting

Sampling in `createDataPartition()`: For factor `y` (e.g., classification), the random sampling is done within the levels of `y` in an attempt to balance the class distributions within the splits. For numeric `y`, the sample is split into groups sections based on percentiles and sampling is done within these subgroups.

```
vsSplits <- createDataPartition(y = trainData$Y,
                                times = 2,
                                p = 0.8,
                                groups = 5,
                                list = FALSE)

head(vsSplits)
```

```
##      Resample1 Resample2
## [1,]         1         2
## [2,]         2         3
## [3,]         4         4
## [4,]         5         5
## [5,]         6         6
## [6,]         8         7
```

(Repeated) K-fold CV

Sometimes we can repeat the K-fold CV multiple times and then calculate the average prediction error.

```
# ten-fold CV
set.seed(1)
cvSplits <- createFolds(y = trainData$Y,
                        k = 10,
                        returnTrain = TRUE)

str(cvSplits)
```

```
## List of 10
##  $ Fold01: int [1:180] 1 2 3 4 5 6 8 10 11 12 ...
##  $ Fold02: int [1:180] 2 3 4 5 6 7 8 9 10 11 ...
##  $ Fold03: int [1:180] 1 2 3 4 5 6 7 8 9 10 ...
##  $ Fold04: int [1:180] 1 2 4 5 6 7 8 9 11 12 ...
##  $ Fold05: int [1:180] 1 2 3 4 5 6 7 8 9 10 ...
##  $ Fold06: int [1:180] 1 3 4 5 6 7 8 9 10 12 ...
##  $ Fold07: int [1:180] 1 2 3 4 5 6 7 8 9 10 ...
##  $ Fold08: int [1:180] 1 2 3 4 5 6 7 8 9 10 ...
##  $ Fold09: int [1:180] 1 2 3 7 8 9 10 11 12 13 ...
##  $ Fold10: int [1:180] 1 2 3 4 5 6 7 9 10 11 ...
```

```
# repeated ten-fold CV
set.seed(1)
rcvSplits <- createMultiFolds(y = trainData$Y,
                              k = 10,
                              times = 5)
# Foldi.Repj - the ith section (of k) of the jth cross-validation set
length(rcvSplits)
```

```
## [1] 50
```

A simple example:

```
K <- length(rcvSplits)
mseK_lm <- rep(NA, K)
mseK_knn <- rep(NA, K)

for(k in 1:K)
{
  trRows <- rcvSplits[[k]]

  fit_lm <- lm(Y~X1+X2, data = trainData[trRows,])
  pred_lm <- predict(fit_lm, trainData[-trRows,])

  pred_knn <- knn.reg(train = trainData[trRows,2:3],
                      test = trainData[-trRows,2:3],
                      y = trainData$Y[trRows], k = 3)


  mseK_lm[k] <- mean((trainData$Y[-trRows] - pred_lm)^2)
  mseK_knn[k] <- mean((trainData$Y[-trRows] - pred_knn$pred)^2)
}

c(mean(mseK_lm), mean(mseK_knn))
```

```
## [1] 1.9092396 0.6483281
```

Specify the resampling method using `trainControl()`

All the resampling methods in the slides are available in `trainControl()`.

```
# K-fold CV
ctrl1 <- trainControl(method = "cv", number = 10)
# leave-one-out CV
ctrl2 <- trainControl(method = "LOOCV")
# leave-group-out / Monte Carlo CV
ctrl3 <- trainControl(method = "LGOCV", p = 0.75, number = 50)
# 632 bootstrap
ctrl4 <- trainControl(method = "boot632", number = 100)
# repeated K-fold CV
ctrl5 <- trainControl(method = "repeatedcv", repeats = 5, number = 10)
# user-specified folds
ctrl7 <- trainControl(index = rcvSplits)


# only fit one model to the entire training set
ctrl6 <- trainControl(method = "none")



set.seed(1)
lmFit <- train(Y~.,
               data = trainData,
               method = "lm",
               trControl = ctrl4)


set.seed(1)
knnFit <- train(Y~.,
                data = trainData,
                method = "knn",
                trControl = ctrl4)

# same training/validation splits?
identical(lmFit$control$index, knnFit$control$index)
```

```
## [1] TRUE
```

```
# compare with mean(mseK_lm) above
lmFit2 <- train(Y~.,
                data = trainData,
                method = "lm",
                trControl = ctrl7)

mean((lmFit2$resample$RMSE)^2)
```

```
## [1] 1.90924
```

```r
mean(mseK_lm)
```

```
## [1] 1.90924
```

```r
knnFit2 <- train(Y~.,
                 data = trainData,
                 method = "knn",
                 tuneGrid = data.frame(k = 3),
                 trControl = ctrl7)

mean((knnFit2$resample$RMSE)^2)
```

```
## [1] 0.6483281
```

```r
mean(mseK_knn)
```

```
## [1] 0.6483281
```

To compare these two models based on their cross-validation statistics, the `resamples()` function can be used with models that share a *common* set of resampled datasets.

```r
resamp <- resamples(list(lm = lmFit, knn = knnFit))
summary(resamp)
```

```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: lm, knn
## Number of resamples: 100
##
## MAE
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## lm  0.7945741 0.9186118 0.9725452 0.9765637 1.0374921 1.2930356    0
## knn 0.4894948 0.5780002 0.6134864 0.6188583 0.6713049 0.7955981    0
##
## RMSE
##          Min.   1st Qu.    Median      Mean   3rd Qu.     Max. NA's
## lm  1.0226799 1.2323791 1.3469670 1.3608658 1.4952322 1.870128    0
## knn 0.5976467 0.7256573 0.7786223 0.8720233 0.9974258 1.525634    0
##
## Rsquared
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## lm  0.5196270 0.6403360 0.6877803 0.6805366 0.7195194 0.7887630    0
## knn 0.7752867 0.8444285 0.8699985 0.8683677 0.8953846 0.9204918    0
```