Cross-Validation

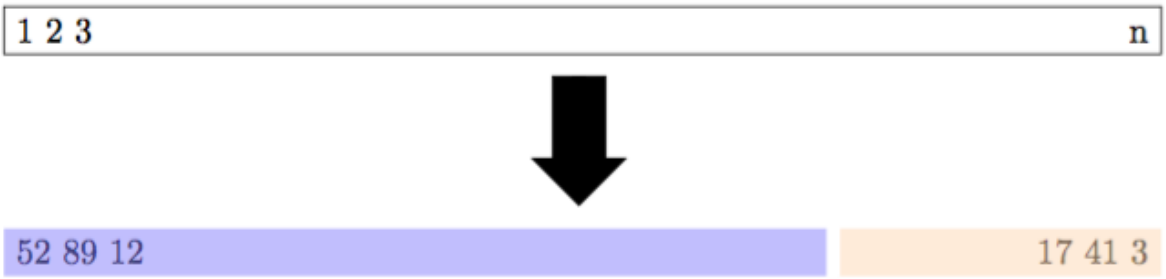
Alan Arnholt

3/22/2016

Note: Working definitions and graphs are taken from Ugarte, Militino, and Arnholt (2016)

The Validation Set Approach

The basic idea behind the validation set approach is to split the available data into a training set and a testing set. A regression model is developed using only the training set. Consider the Figure below which illustrates a split of the available data into a training set and a testing set.

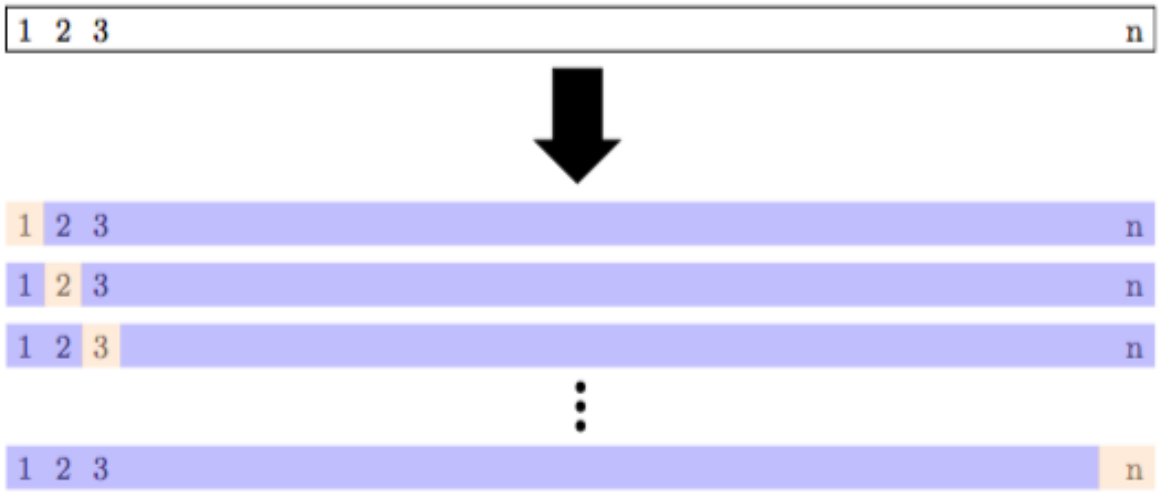


The percent of values that are allocated into training and testing may vary based on the size of the available data. It is not unusual to allocate 70–75% of the available data as the training set and the remaining 25–30% as the testing set. The predictive performance of a regression model is assessed using the testing set. One of the more common methods to assess the predictive performance of a regression model is the mean square prediction error (MSPE). The MSPE is defined as

$$MSPE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Leave-One-Out Cross Validation

The leave-one-out cross-validation (LOOCV) eliminates the problem of variability in MSPE present in the validation set approach. The LOOCV is similar to the validation set approach as the available n observations are split into training and testing sets. The difference is that each of the available n observations are split into n training and n testing sets where each of the n training sets consist of $n-1$ observations and each of the testing sets consists of a single different value from the original n observations. The Figure below provides a schematic display of the leave-one-out cross-validation process with testing sets (light shade) and training sets (dark shade) for a data set of n observations.

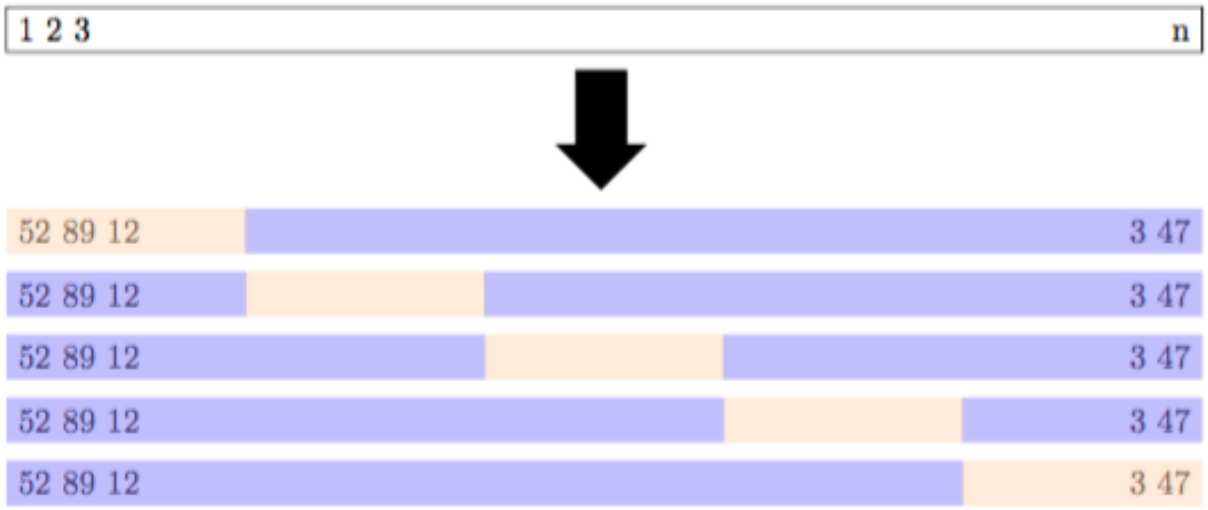


The MSPE is computed with each testing set resulting in n values of MSPE. The LOOCV estimate for the test MSPE is the average of these n MSPE values denoted as

$$CV_n = \frac{1}{n}\sum_{i=1}^{n} MSPE_i$$

k-Fold Cross Validation

k-fold cross-validation is similar to LOOCV in that the available data is split into training sets and testing sets; however, instead of creating n different training and testing sets, k folds/groups of training and testing sets are created where $k < n$ and each fold consists of roughly n/k values in the testing set and $1 - n/k$ values in the training set. The Figure below shows a schematic display of 5-fold cross-validation. The lightly shaded rectangles are the testing sets and the darker shaded rectangles are the training sets.



The MSPE is computed on each of the k folds using the testing set to evaluate the regression model built from the training set. The average of k MSPE values is denoted as

$$CV_k = \frac{1}{k}\sum_{i=1}^{k} MSPE_i$$

Note that LOOCV is a special case of k-fold cross-validation where k is set equal to n. An important advantage k-fold cross-validation has over LOOCV is that CV_k for $k = 5$ or $k = 10$ provides a more accurate estimate of the test error rate than does CV_n.

```
> str(DF)
```

```
'data.frame':   1000 obs. of  2 variables:
 $ x: num  5 5.01 5.02 5.03 5.04 ...
 $ y: num  -0.462 -1.2553 -0.3506 -1.0662 -0.0222 ...
```

Validation Set Approach

- Create a training set using 75% of the observtions in `DF`.
- Sort the observations in the training and testing sets.

```
> n <- nrow(DF)
> train <- sample(n, floor(0.75 * n), replace = FALSE)
> trainSET <- DF[train, ]
> trainSET <- trainSET[order(trainSET$x), ]
> testSET <- DF[-train, ]
> testSET <- testSET[order(testSET$x), ]
> dim(trainSET)
```

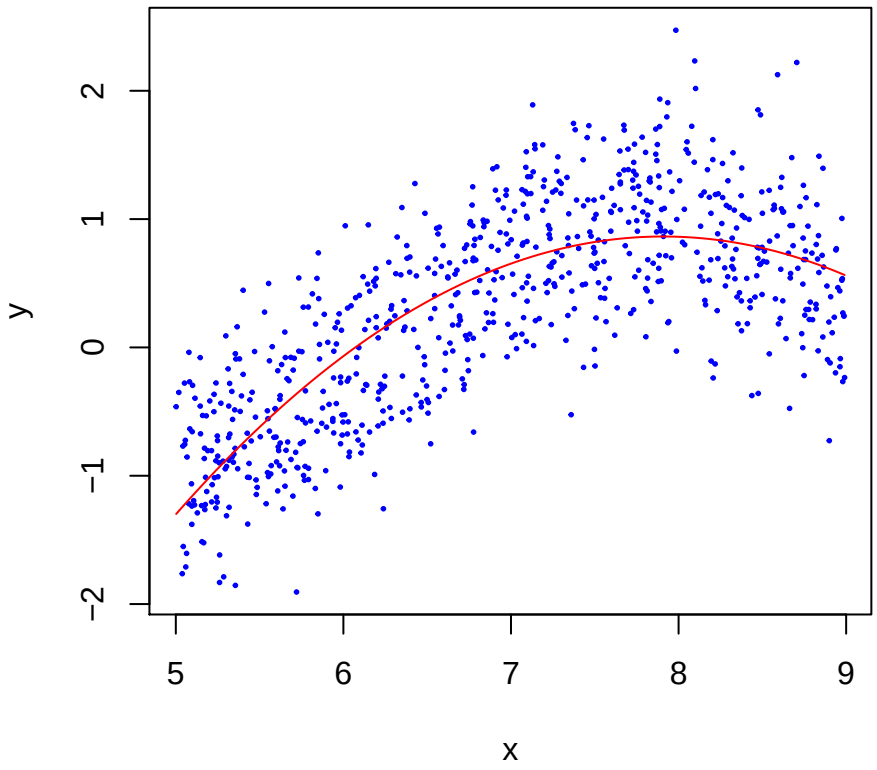
```
[1] 750   2
```

```
> dim(testSET)
```

```
[1] 250   2
```

- Fit a quadratic model using the training set (`trainSET`).

```
> modq <- lm(y ~ poly(x, 2, raw = TRUE), data = trainSET)
> yhat <- predict(modq, data = trainSET)
> plot(y ~ x, data = trainSET, pch = 19, cex = .25, col = "blue")
> lines(trainSET$x, yhat, col = "red")
```




```
> summary(modq)
```

```
Call:
lm(formula = y ~ poly(x, 2, raw = TRUE), data = trainSET)

Residuals:
     Min       1Q   Median       3Q      Max
-1.54481 -0.37908  0.00049  0.36025  1.60889

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)              -15.14741    0.73452  -20.62   <2e-16 ***
poly(x, 2, raw = TRUE)1    4.04929    0.21451   18.88   <2e-16 ***
poly(x, 2, raw = TRUE)2   -0.25601    0.01532  -16.71   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5059 on 747 degrees of freedom
Multiple R-squared:  0.6106,    Adjusted R-squared:  0.6096
F-statistic: 585.7 on 2 and 747 DF,  p-value: < 2.2e-16
```

```
> anova(modq)
```

```
Analysis of Variance Table

Response: y
                         Df Sum Sq Mean Sq F value    Pr(>F)
```

```
poly(x, 2, raw = TRUE)   2 299.80 149.898   585.7 < 2.2e-16 ***
Residuals              747 191.18   0.256
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Compute the training MSPE

```
> MSPE <- mean(resid(modq)^2)
> MSPE
```

```
[1] 0.254904
```

Compute the testing MSPE

```
> yhtest <- predict(modq, newdata = testSET)
> MSPEtest <- mean((testSET$y - yhtest)^2)
> MSPEtest
```

```
[1] 0.2967722
```

Fit a cubic model.

```
> plot(y ~ x, data = trainSET, pch = 19, cex = .25, col = "blue")
> modc <- lm(y ~ poly(x, 3, raw = TRUE), data = trainSET)
> summary(modc)
```

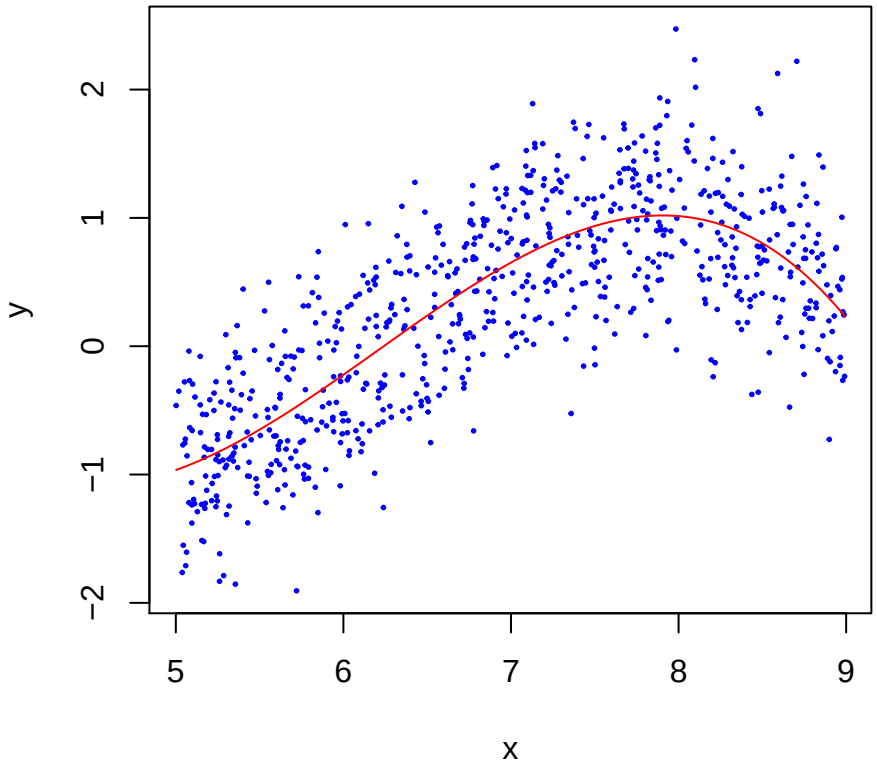
```
Call:
lm(formula = y ~ poly(x, 3, raw = TRUE), data = trainSET)

Residuals:
     Min       1Q   Median       3Q      Max
-1.43528 -0.35553 -0.01833  0.35667  1.60773

Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)             19.81600    4.88653   4.055 5.54e-05 ***
poly(x, 3, raw = TRUE)1 -11.45433    2.15380  -5.318 1.39e-07 ***
poly(x, 3, raw = TRUE)2   1.99571    0.31171   6.402 2.70e-10 ***
poly(x, 3, raw = TRUE)3  -0.10721    0.01482  -7.232 1.18e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4894 on 746 degrees of freedom
Multiple R-squared:  0.6361,    Adjusted R-squared:  0.6347
F-statistic: 434.7 on 3 and 746 DF,  p-value: < 2.2e-16
```

```
> yhat <- predict(modc, data = trainSET)
> lines(trainSET$x, yhat, col = "red")
```



Compute the training MSPE

```
> MSPE <- mean(resid(modc)^2)
> MSPE
```

```
[1] 0.238204
```

Compute the testing MSPE

```
> yhtest <- predict(modc, newdata = testSET)
> MSPEtest <- mean((testSET$y - yhtest)^2)
> MSPEtest
```

```
[1] 0.2801967
```

Your Turn

- Create a training set (80%) and testing set (20%) of the observations from the data frame `HSWRESTLER` from the `PASWR2` package. Store the results from regressing `hwfat` onto `abs` and `triceps` in the object `modf`.

- Compute the test `MSPE`.
- Note how answers the class computes are different. The validation estimate of the test MSPE can be highly variable.

```
> # Your Code Here
> #
> #
> #
> #
> #
> #
> #
> #
> #
> #
> #
```

k Fold Cross Validation

- Create $k = 5$ folds.
- Compute the $CV_{k=5}$ for `modq`.

```
> set.seed(1)
> k <- 5
> MSPE <- numeric(k)
> folds <- sample(x = 1:k, size = nrow(DF), replace = TRUE)
> xtabs(~folds)
```

```
folds
  1   2   3   4   5
200 205 201 196 198
```

```
> sum(xtabs(~folds))
```

```
[1] 1000
```

```
> for(j in 1:k){
+   modq <- lm(y ~ poly(x, 2, raw = TRUE), data = DF[folds != j, ])
+   pred <- predict(modq, newdata = DF[folds ==j, ])
+   MSPE[j] <- mean((DF[folds == j, ]$y - pred)^2)
+ }
> MSPE
```

```
[1] 0.2604303 0.2601093 0.2656551 0.2614124 0.2873636
```

```
> mean(MSPE)
```

```
[1] 0.2669941
```

Your Turn

- Compute the CV_8 for `modf`.

```
> # Your Code Here
> #
> #
> #
> #
> #
> #
> #
> #
> #
> #
> #
> #
```

Using `cv.glm` from `boot`

```
> set.seed(1)
> library(boot)
> glm.fit <- glm(y ~ poly(x, 2, raw = TRUE), data = DF)
> cv.err <- cv.glm(data = DF, glmfit = glm.fit, K = 5)$delta[1]
> cv.err
```

```
[1] 0.2663178
```

Your Turn

- Compute CV_8 for `modf`

```
> # Your Code Here
> #
> #
> #
```

Leave-One-Out Cross-Validation

```
> set.seed(1)
> k <- nrow(DF)
> MSPE <- numeric(k)
> folds <- sample(x = 1:k, size = nrow(DF), replace = FALSE)
> # Note that replace changes to FALSE for LOOCV...can you explain why?
> for(j in 1:k){
+   modq <- lm(y ~ poly(x, 2, raw = TRUE), data = DF[folds != j, ])
+   pred <- predict(modq, newdata = DF[folds ==j, ])
+   MSPE[j] <- mean((DF[folds == j, ]$y - pred)^2)
+ }
> mean(MSPE)
```

```
[1] 0.2663587
```

Your Turn

- Compute CV_n for `modf`.

```
> # Your Code Here
> #
> #
> #
> #
> #
> #
> #
> #
> #
```

Recall

$$CV_n = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1 - h_i}\right)^2$$

```
> modq <- lm(y ~ poly(x, 2, raw = TRUE), data = DF)
> h <- hatvalues(modq)
> CVn <- mean(((DF$y - predict(modq))/(1 - h))^2)
> CVn
```

```
[1] 0.2663587
```

Your Turn

- Compute CV_n for `modf` using the mathematical shortcut.

```
> # Your Code Here
> #
> #
> #
```

Using `cv.glm` from `boot`

- Note: If one does not use the `K` argument for the number of folds, `gv.glm` will compute LOOCV.

```
> library(boot)
> glm.fit <- glm(y ~ poly(x, 2, raw = TRUE), data = DF)
> cv.err <- cv.glm(data = DF, glmfit = glm.fit)$delta[1]
> cv.err
```

```
[1] 0.2663587
```

Your Turn

- Compute CV_n for `modf` using `cv.glm`.

```
> # Your Code Here
> #
> #
> #
```

References

Ugarte, María Dolores, Ana F. Militino, and Alan T. Arnholt. 2016. *Probability and Statistics with R*. Second edition. Boca Raton: CRC Press, Taylor & Francis Group.