Cross-validation demonstration

Marc Taylor

2021-Mar-14

1 Introduction

The following script demonstrates the use of cross-validation (CV) for use in estimating the predictive power of a model. More broadly, CV offers a framework for model comparison and selection. In particular, CV can aid in avoiding problems of ***overfitting*** and allow for the comparison of models that use different predictive modeling procedure.

The basic idea of CV is to partition data into a "training" and "validation" sets. Model fitting is done on the training set, which is followed by prediction the response variable of the validation set. The error between observed and predicted values can be estimated according to a given index of choice; e.g.:

1. Mean square error: $MSE = \frac{1}{n}\sum(\hat{y}_i - y_i)^2$
2. Root mean square error: $RMSE = \sqrt{\frac{1}{n}\sum(\hat{y}_i - y_i)^2}$
3. Root mean square error (log): $RMSE = \sqrt{\frac{1}{n}\sum(\log\hat{y}_i - \log y_i)^2}$
4. Median absolute percentage error $MdAPE = \text{median}(\frac{|\hat{y}_i - y_i|}{y_i} \times 100)$

1.1 Types of CV

There are two main types of CV: 1. Exhaustive and 2. Non-exhaustive. Exhaustive CV includes "Leave-p-out" (LpO), which tests all possible partitions of the data. Some values for p can become computationally expensive due to the number of possible combinations. "Leave-one-out" (LOO, is a commonly used type of exhaustive CV given that the number of possible combinations is equal to the number of samples in the data set ($p = n$)

Non-exhaustive approaches test a subset of training / validation partitions, although the procedure may be repeated several times (*i.e.* permutations) in order to obtain a robust estimate of error. Examples include *k-fold* CV and *repeated random sub-sampling* CV (via Monte Carlo). In this demonstration we will focus on a permutated k-fold CV

k-fold CV divides the data into k partitions. Each partition is left out during one of the fitting routines, and later used as the validation set. For example, a 10-fold CV will split the data into 10 (approximately) equal subsets. The model is then fit 10 times, sequentially leaving out each of the partitions and fitting with the remaining data (approx. 90% of the samples). With each fitting, the model is used to predict the response values of the left-out partition, which results in each sample being used is used once in the validation set. Once completed, an overall error index is calculated and recorded. This final error index can then used to evaluate between different models. Note that it is important to reset the random value generator (e.g. set.seed()) at the beginning of the routine in order to ensure that the same partitions are used for each model being tested.

The following function can be used to determine the partitions given the number of samples in the full data set (n) and the number of partitions (k).

1.2 Required function (from https://github.com/marchtaylor/sinkr):

```
# Determines k-fold partitions for a given number of samples
# n is the number of samples; k is the number of partitions
kfold <- function(n, k=NULL){
  if(is.null(k)){ k <- n} # if undefined, assume leave-one-out (LOO) CV
  res <- vector(mode="list", k)
  n.remain <- seq(n)
  for(i in seq(k)){
    samp <- sample(seq(length(n.remain)), ceiling(length(n.remain)/(k-i+1)))
    res[[i]] <- n.remain[samp]
    n.remain <- n.remain[-samp]
  }
  return(res)
}
```

Example of kfold use:

```
# 5 samples, 4 folds
n <- 35; k <- 4
res <- kfold(n, k)
res # partition indices
```

```
## [[1]]
## [1] 33 25 32 20 17 13  4 19 18
##
## [[2]]
## [1] 21 22 30 23 24  2  6 12  7
##
## [[3]]
## [1]  5  9 10 11 31  1  3 34 29
##
## [[4]]
## [1] 27 35 16 26  8 14 28 15
```

```
unlist(lapply(res, length)) # number of samples in each partition
```

```
## [1] 9 9 9 8
```

```
COL <- rep(NaN, n)
PAL <- c("yellow", "blue", "green", "red")
for(i in seq(res)){COL[res[[i]]] <- i}
plot(x=seq(n), y=rep(1,n), bg=PAL[COL],
  pch=21, cex=2, yaxt="n", ylab="", xlab="sample")
legend("top", legend = seq(k), title = "fold number",
  pt.cex=2, pch=21, pt.bg = PAL, ncol=k, bty = "n")
```

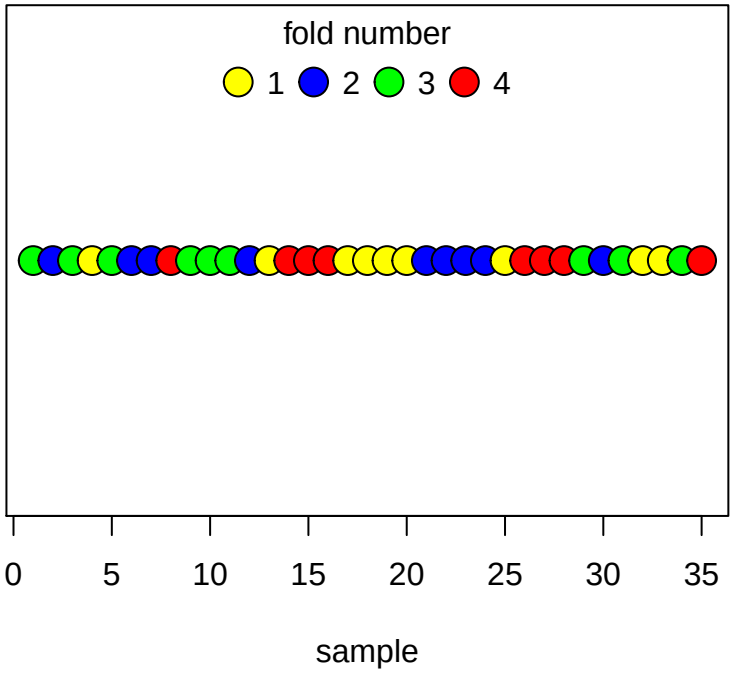

Figure 1: Example of a `r k` k-fold partition of `r n` samples.

2 Ex. 1. Generate synthetic dataset and fit polynomial functions of varying complexity

In the following examples, we will be fitting a polynomial function of varying complexity to a synthetic data set where the underlying polynomial used to generate the data in known (degree=3).

2.1 Synthetic data generation

```
set.seed(1111)
n <- 50
x <- sort(runif(n, -2.5, 2))
y <- 3*x^3 + 5*x^2 + 0.5*x + 20 # a 3 polynomial model
err <- rnorm(n, sd=3)
ye <- y + err
df <- data.frame(x, ye)
nterm <- c(1,2,3,5,7,9)

# Model fitting and visualization
plot(ye~x, df, ylab="y")
PAL <- colorRampPalette(c("blue", "cyan", "yellow", "red"))
COLS <- PAL(length(nterm))
for(i in seq(nterm)){
 fit <- lm(ye ~ poly(x, degree=nterm[i]), data=df)
 newdat <- data.frame(x=seq(min(df$x), max(df$x), length.out = 100))
 lines(newdat$x, predict(fit, newdat), col=COLS[i])
}
legend("topleft", legend=paste0(nterm, c("", "", "*", "", "", "")),
  title="polynomial degrees", bty="n", col=COLS, lty=1)
```

2.2 Summary of term significance using largest polynomial

It's obvious from a visual examination of the figure that polynomials of degree > 3 all fit the data pretty well, but it's less clear that the higher order polynomials are improving the predictive power to a significant degree.

The output of the summary() function gives a first idea about the significance of the polynomial terms:

```
fit <- lm(ye ~ poly(x, degree=9), data=df)
summary(fit)
```

```
##
## Call:
## lm(formula = ye ~ poly(x, degree = 9), data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.8833 -2.0514 -0.3533  1.8598  7.6898
##
## Coefficients:
```

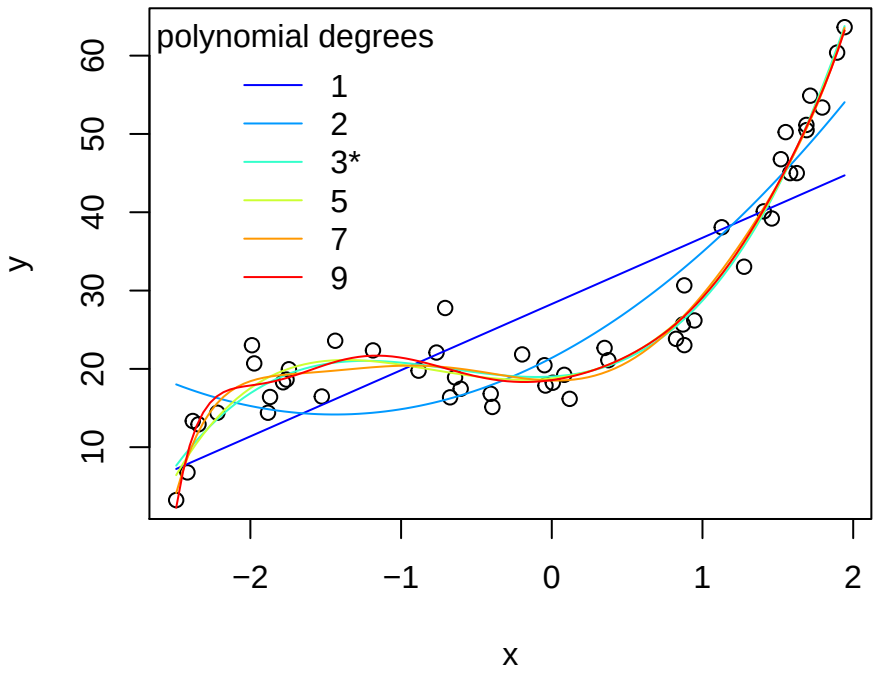

Figure 2: Fitted polynomial functions of varying degree. The '*' denotes the number of degrees used in the data generation.

```
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)             27.143      0.428  63.424  < 2e-16 ***
## poly(x, degree = 9)1    84.941      3.026  28.069  < 2e-16 ***
## poly(x, degree = 9)2    39.919      3.026  13.191 3.74e-16 ***
## poly(x, degree = 9)3    34.235      3.026  11.313 4.89e-14 ***
## poly(x, degree = 9)4    -3.004      3.026  -0.993    0.327
## poly(x, degree = 9)5     1.139      3.026   0.376    0.709
## poly(x, degree = 9)6    -2.277      3.026  -0.752    0.456
## poly(x, degree = 9)7     4.211      3.026   1.392    0.172
## poly(x, degree = 9)8    -3.501      3.026  -1.157    0.254
## poly(x, degree = 9)9     2.452      3.026   0.810    0.423
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.026 on 40 degrees of freedom
## Multiple R-squared:  0.9648, Adjusted R-squared:  0.9568
## F-statistic: 121.7 on 9 and 40 DF,  p-value: < 2.2e-16
```

3 Ex. 2. Cross validation

Subsequent removal of terms and comparison of **nested models** via ANOVA, Akaike information criterion (AIC), etc., is a typical strategy for determining the "best" model. However, for cases where models are un-nested, or are based on differing model construction (i.e. mixed models with random terms), such criteria may not possible due to large differences in their degrees of freedom. For these cases, CV offers a robust tool for model selection.

The following example uses the same synthetic example, but demonstrates the ability of CV to determine the best model. In addition to varying polynomial complexity, the "trials" also test the impact of sample size on prediction error.

3.1 Synthetic data generation

```
set.seed(1111)
n <- seq(50, 500, 50) # number of data points
nterm <- seq(1,9,1) # polynomial complexity
perms <- 2 # permutations
k <- 4 # number of "folds" used in data partitioning
trials <- expand.grid(
  n=n, nterm=nterm, k=k, AIC=NaN, train.rmse=NaN,  valid.rmse=NaN)
dim(trials)
```

```
## [1] 90  6
```

```
x <- sort(runif(max(trials$n), -2.5, 2))
y <- 3*x^3 + 5*x^2 + 0.5*x + 20 # a 3 polynomial model
err <- rnorm(max(trials$n), sd=3)
ye <- y + err
df <- data.frame(x, ye, train.err=NaN, valid.err=NaN)
dim(df)
```

```
## [1] 500   4
```

In each trial, a "full model" is fit to all data. This aids later sections looking at AIC as a criterion of model selection, but also provides a base model fitting that can be updated during the k-fold permutations (i.e. using update()). The code is structured in the following manner:

1. The for(i in seq(nrow(trials))){} loop runs through the trials
2. The for(j in seq(perms)){} loop runs through each permutation by trial
3. The for(k in seq(ks){} loop runs through each fold by permutation and trial

3.2 Loop through "trials"

```
# Loop through trials
for(i in seq(nrow(trials))){ # for each trial (variable sample size, polynomial degree)
  # subsample data
  set.seed(1)
  incl <- sample(nrow(df), trials$n[i])
  df.i <- df[incl,]

  # fit model to full data
  fit.i <- lm(ye ~ poly(x, degree=trials$nterm[i]), data=df.i)
  trials$AIC[i] <- AIC(fit.i) # record AIC

  # perform cross validation
  res <- vector("list", perms)
  for(j in seq(perms)){ # for each permutation j
    res[[j]] <- df.i
    ks <- kfold(n = trials$n[i], k = trials$k[i])
    for(k in seq(ks)){ # for each partition k
      train <- -ks[[k]]
      valid <- ks[[k]]
      fit.k <- update(fit.i, data=res[[j]][train,]) # efficient update of model
      # fit.k <- lm(ye ~ poly(x, degree=trials$nterm[i]), data=res[[j]][train,])
      res[[j]]$train.err[train] <- predict(fit.k) - res[[j]]$y[train]
      res[[j]]$valid.err[valid] <- predict(fit.k, newdata = res[[j]][valid,]) -
        res[[j]]$y[valid]
    }
  }
  res <- do.call("rbind", res)

  # record rmse of trial data
  trials$train.rmse[i] <- sqrt(mean(res$train.err^2))
  # record rmse of validation data
  trials$valid.rmse[i] <- sqrt(mean(res$valid.err^2))
  print(paste(i, "of", nrow(trials), "is completed"))
}
```

3.3 Results

3.3.1 Influence of model complexity on prediction error

As was seen with the results of summary(), polynomials of degree > 3 are shown to increase prediction error in the validation set. By contrast, the error in estimating the training data set continues to decrease with increasing model complexity (*i.e.* over-fitting).

```
# n = 50
trials.sub <- subset(trials, n==50)
plot(train.rmse ~ nterm, trials.sub, log="y", t="o",
  pch=c(1,16)[(trials.sub$train.rmse == min(trials.sub$train.rmse))+1],
  ylim=range(trials.sub$train.rmse, trials.sub$valid.rmse), ylab="RMSE")
lines(valid.rmse ~ nterm, trials.sub, t="o", col=2,
  pch=c(1,16)[(trials.sub$valid.rmse == min(trials.sub$valid.rmse))+1])
legend("topright", legend=c("training", "validation"), col=1:2, lty=1, pch=1)
abline(h=sqrt(mean((err)^2)),lty=2, col=8)
abline(v=3, lty=2, col=8)
mtext("Sample size (n = 50)")
```

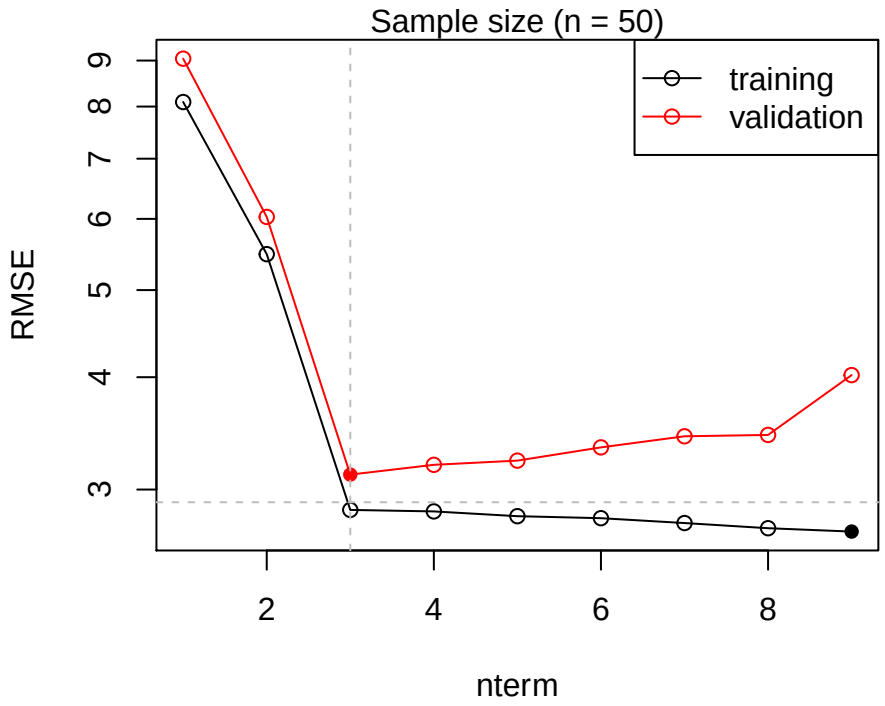

Figure 3: Error (RMSE) as a function of polynomial degree (n=50). Minimum error is designated by the filled symbol. The actual observation error of the synthetic dataset is shown by dotted grey line.

3.3.2 Influence of model complexity and sample size on prediction error

Another interesting aspect of the trial data set is regarding the impact of sample size on over-fitting. The following plot shows that the difference between fitting the correct degree polynomial (3) and an over-fit polynomial (7) converge at higher sample size (*e.g.* n>150). So, smaller data-sets are more likely to suffer from poor prediction performance due to issues of over-fitting due to stronger impact of each sample. Such an exercise may be used in order to help plan the amount of data collection needed to improve prediction error.

```
# Influence of sample size (actual error is approached from opposite directions
# by training and validation sets)
# overfitting is largely remedied by sample size (i.e. n >= 150)
YLIM <- range(subset(trials, nterm %in% c(3,7))[,c("train.rmse", "valid.rmse")])
plot(train.rmse ~ n, trials, subset = (nterm==7), log="y", t="o", pch=16,
     ylim=YLIM, ylab="RMSE")
lines(valid.rmse ~ n, trials, subset = (nterm==7), t="o", col=2, pch=16)

lines(train.rmse ~ n, trials, subset = (nterm==3), t="o", col=1, lty=2)
lines(valid.rmse ~ n, trials, subset = (nterm==3), t="o", col=2, lty=2)

legend("topright",
  legend=c(
    "training (nterm=7)",
    "validation (nterm=7)",
    "training (nterm=3)",
    "validation (nterm=3)"
  ),
  col=c(1,2,1,2), lty=c(1,1,2,2), pch=c(16,16, 1,1)
)
abline(h=sqrt(mean((err)^2)),lty=2, col=8)
```

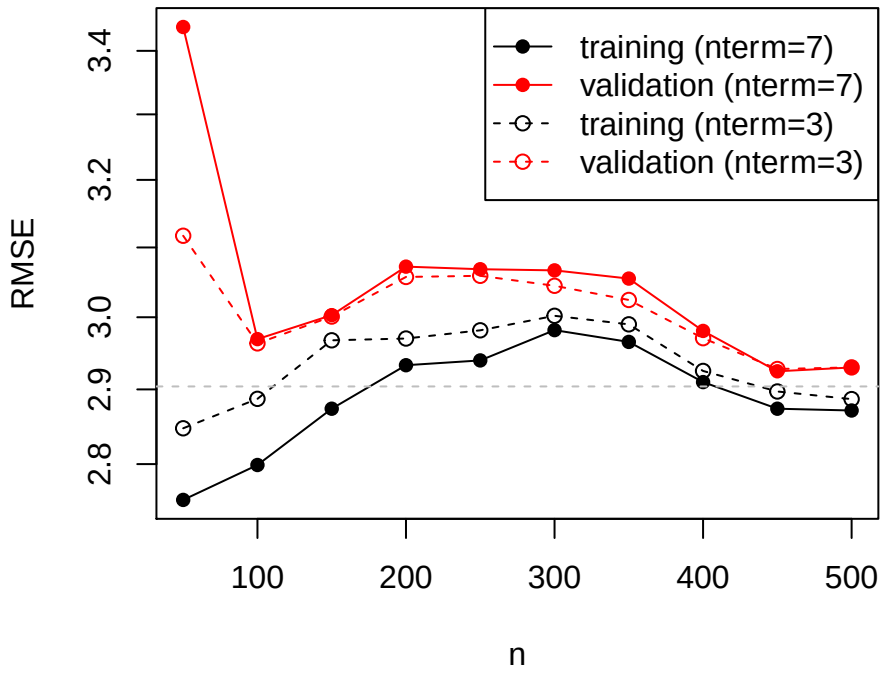

Figure 4: Error (RMSE) as a function of polynomial degree and sample size. The actual observation error of the synthetic dataset is shown by dotted grey line.

4 Further comments

4.0.1 Akaike information criterion for model selection

When comparing nested models or models of similar construction, Akaike information criterion (AIC) is usually an appropriate measure of model performance in that it introduces a penalization based on the degrees of freedom of the model.

```
# AIC (correctly determines best model)
# n=50
trials.sub <- subset(trials, n==50)
plot(AIC ~ nterm, trials.sub, t="o",
  pch=c(1,16)[(trials.sub$AIC == min(trials.sub$AIC))+1])
abline(v=3, lty=2, col=8)
mtext("Sample size (n = 50)")
```

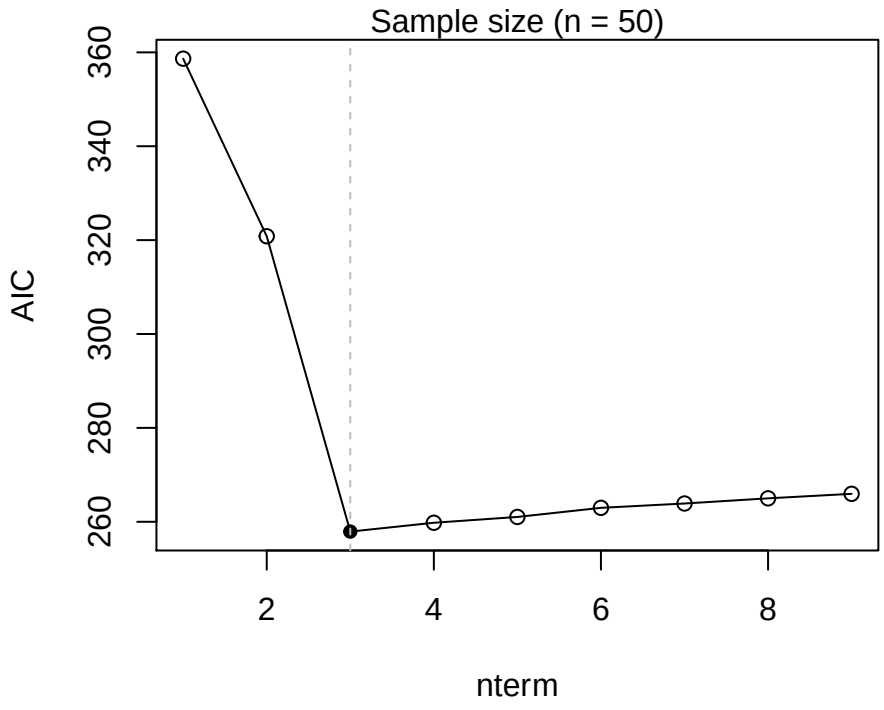

Figure 5: Akaike information criterion (AIC) as a function of polynomial degree (n=50). Minimum error is designated by the filled symbol.