Here are the R packages that I’ll be using:

library(tidyverse)

library(rlang)

library(rsample)

library(yardstick)

library(purrr)

First, duplicate the process of reading in the data and adding two new columns:

stocks <- read.csv('http://blog.mindymallory.com/wp-content/uploads/2018/02/stocks.csv') %>%

as\_tibble() %>%

mutate(

USStockUse = USEndingStocks/USTotalUse,

WorldStockUse = ROWEndingStocks/WorldTotalUse

)

stocks

## # A tibble: 43 x 8

## Year USEndingStocks ROWEndingStocks USTotalUse

##

## 1 1975 633. 36411 5767.

## 2 1976 1136. 39491 5789.

## 3 1977 1436. 40833 6207.

## 4 1978 1710. 47957 6995.

## 5 1979 2034. 59481 7604.

## 6 1980 1392. 67180 7282.

## 7 1981 2537. 62725 6975.

## 8 1982 3523. 60273 7249.

## 9 1983 1006. 63421 6693.

## 10 1984 1648. 76287 7032.

## # ... with 33 more rows, and 4 more variables:

## # WorldTotalUse , PriceRecievedFarmers ,

## # USStockUse , WorldStockUse

The blog post has an excellent description of cross-validation and looked at five different models that encoded the US and World stock-use predictors. Either a log- or inverse-transformation was applied and then polynomial basis functions were used on these features to demonstrate overfitting.

The blog post has some for loops to do the resampling and I volunteered to show how to do it with some tidy modeling packages.

**Tidy Cross-Validation**

First, let’s take the *easy* part. Instead of using for loops, we can use the new infrastructure in the tidyverse to resample the model. The [rsample](https://topepo.github.io/rsample) package has some functions for different types of resampling and we will use the same procedure as the original post:

set.seed(918)

resamp\_info <- vfold\_cv(stocks, v = 5)

resamp\_info

## # 5-fold cross-validation

## # A tibble: 5 x 2

## splits id

##

## 1 Fold1

## 2 Fold2

## 3 Fold3

## 4 Fold4

## 5 Fold5

The first column in the tibble is a set of “rsplit” objects that define how the data are split for each fold of cross-validation. Each one fully and efficiently encapsulates everything what is needed to get the two divisions of the original data. In rsample, to avoid naming confusion, we label the two resulting data sets as:

* The **analysis data** are those that we selected in the resample. For a bootstrap, this is the sample with replacement. For 5-fold cross-validation, this is the 80% of the data. These data are often used to fit a model or calculate a statistic in traditional bootstrapping.
* The **assessment data** are usually the section of the original data not covered by the analysis set. Again, in 5-fold CV, this is the 20% held out. These data are often used to evaluate the performance of a model that was fit to the analysis data.

To get these partitions for the first split there are functions analysis and assessment that return the appropriate data frames when given an rsplit:

# printing just shows the #rows per analysis/assessment/overall

resamp\_info$splits[[1]]

## <34/9/43>

# data used for modeling:

analysis(resamp\_info$splits[[1]])

## # A tibble: 34 x 8

## Year USEndingStocks ROWEndingStocks USTotalUse

##

## 1 1975 633. 36411 5767.

## 2 1976 1136. 39491 5789.

## 3 1977 1436. 40833 6207.

## 4 1979 2034. 59481 7604.

## 5 1980 1392. 67180 7282.

## 6 1981 2537. 62725 6975.

## 7 1982 3523. 60273 7249.

## 8 1983 1006. 63421 6693.

## 9 1984 1648. 76287 7032.

## 10 1985 4040. 75069 6494.

## # ... with 24 more rows, and 4 more variables:

## # WorldTotalUse , PriceRecievedFarmers ,

## # USStockUse , WorldStockUse

The first model for the data contained the US stock-use data with an inverse transformation. Let’s side-step the polynomial model tuning for now and just fit a quadratic model. To make things easier, I’ll define a function that can be used to fit the model when given an rsplit object and return the holdout mean squared error (MSE):

glm\_results <- function(split, ...) {

# Get the data used ot fit the model aka the "analysis" set

# and fit the model with a formula given in the ...

mod <- glm(data = analysis(split), ...)

# Get predictions on the other data (aka the "assessment" set

# and compute some metrics

holdout <- assessment(split)

# Compute performance using the yardstick package

rmse <- holdout %>%

mutate(pred = predict(mod, holdout)) %>%

rmse(truth = PriceRecievedFarmers, estimate = pred)

rmse^2

}

We can use this with any formula since it is just passed to glm using the ellipses. For example to get the holdout MSE estimate for the first fold:

glm\_results(resamp\_info$splits[[1]], formula = PriceRecievedFarmers ~ poly(1 / USStockUse, 2))

## [1] 1.67

To get these statistics for all folds, purrr::map\_dbl is used to add another column:

resamp\_info <- resamp\_info %>%

mutate(

model\_1\_deg\_2 =

map\_dbl(

splits,

glm\_results,

PriceRecievedFarmers ~ poly(1 / USStockUse, 2)

)

)

resamp\_info

## # 5-fold cross-validation

## # A tibble: 5 x 3

## splits id model\_1\_deg\_2

## \*

## 1 Fold1 1.67

## 2 Fold2 0.558

## 3 Fold3 0.596

## 4 Fold4 9.58

## 5 Fold5 0.248

That’s a lot of variation in the outcome! The mean value is fairly consistent with the blog post though:

resamp\_info %>% select(model\_1\_deg\_2) %>% colMeans()

## model\_1\_deg\_2

## 2.53

# MSE = 2.675 in the blog post

*K*-fold cross-validation is one of the noisiest resampling methods so this difference isn’t too surprising.

This same process could be repeated for each polynomial degree to get new columns for this model (we’ll discuss this below). The good things about doing things this way:

* It is a lot cleaner (so far) than doing for loops.
* Other tidyverse infrastructure can be used. For example, [tidyposterior](https://topepo.github.io/tidyposterior) is a great way to do model comparisons with resampling and Bayesian analysis.
* It is simple to change resampling methods. Suppose you wanted to change to a larger number of bootstrap resamples (given the variance shown above). The same infrastructure can be easily exchanged; resample::bootstraps is used in place of rsample::vfold\_cv.

**Tidy Model Specification (maybe)**

The model specification part is, for me, a lot more difficult to tidy. It would be good to be able to state what predictors that we want, specify the polynomial degree, and have a function to generate the appropriate formula. The original post sensibly just types the terms out.

I spent some time thinking about how we could use [expressions](https://adv-r.hadley.nz/expressions.html) and tidy evaluation ([video from Hadley](https://t.co/xEhCKne2mM)) to make it a little less script-like. The problem is that the solution took me a while to write and, arguably, it doesn’t really buy you much more than the original code (apart from the potential copy/paste duplication errors).

In any case, the function uses rlang to manipulate expressions to make the formula. The inputs are expressions of *how* the predictors are used (i.e. log, inverse, etc.) and the degree. It captures the expression (without evaluating it), substitutes it into the polynomial function, then creates the formula:

make\_formula <- function(..., degree = 1) {

# Capture the expression so it is not evaluated

var\_expr <- exprs(...)

# Create a template expression

inv\_poly <- quote(poly(x = x, degree = degree))

# Use a wrapper around rlang::call\_modify to reverse the

# order of the arguments so that we can map over the

# predictor expressions

add\_args <- function(arg, call, ...)

call\_modify(call, x = arg, ...)

# Add the variables and the degree into the template

poly\_expr <- map(var\_expr, add\_args, call = inv\_poly, degree = degree)

# Convert to character

poly\_char <- map\_chr(poly\_expr, deparse)

# Convert to a formula

poly\_char <- paste(poly\_char, collapse = " + ")

as.formula(paste("PriceRecievedFarmers ~", poly\_char))

}

# For example:

make\_formula(1/USStockUse, log(WorldStockUse), degree = 3)

## PriceRecievedFarmers ~ poly(x = 1/USStockUse, degree = 3) + poly(x = log(WorldStockUse),

## degree = 3)

##

From here, we could use a bunch of mutate commands like the one shown above or write a slightly smaller for loop to work across the polynomial degrees. While the function above works well, the overall approach to working across models isn’t particularly satisfying.