In-Class Programming Activity 13

Math 253: Statistical Computing & Machine Learning

k-1 cross-validation

Today you are going to write a function to carry out k-fold cross validation. Recall that this involves dividing the available data set into k equal-sized, independent sets. For each of the k sets, reserve that set as testing data, fit your model to the remaining data, and evaluate the fitted model on the reserved set. In doing this, you will accumulate k different estimates of the model error. Average those to produce an overall estimate of model error.

A function framework

Call your function `k_fold1()`. It should have this interface:

```
k_fold1 <- function(formula, method = lm, data = mtcars, predfun = predict, k = 10) {
  # do the calculations,
  # producing an estimate of error
  return(error_estimate)
}
```

As written above, the function does nothing. You're going to fix that. But use exactly the interface (function name, argument names and order, default values) given above.

The k sets

In k-fold cross validation, you'll divide the cases in your data into k sets. To help do that, you're going to create a vector named `sets` that has one element for each case in the data. The value of that element will be 1 if the corresponding case is to be in set 1 of the k sets, 2 if the case is to be in set 2, and so on. It doesn't matter how you assign cases to sets, so long as there are k sets of roughly equal size.

There are many ways to construct the vector `sets`. Here are two operations, `%%` and `rep(, , each=)` that might be the basis for two different ways of creating the vector Here are two operations that may give you a hint. The example assumes that there are 51 rows in the `data`.

```
k <- 10
sets <- (1:51 %% k) + 1
# or, alternatively, ...
sets <- rep(1:k, each = 51/k,
 length.out = 51)
```

The loop

Your function will perform the same operation k times. Doing this will produce k numbers, the mean square prediction error for each of the k test data sets. In writing a loop, you . . .

1. Set up the state or "accumulator" that will be updated as you go through the loop
2. Construct the outline of the loop
3. Fill in the operations to be carried out inside the loop.
4. Tidy up the accumulator to produce the final result.

Here's a code fragment that handles (1), (2), and (4).

```
mspe <- numeric(k)
for (i in 1:k) {
# Your statements go here
}
mean(mspe)
```

Make sure that you understand what `numeric(k)` is doing.

Inside the loop

As you can see, the statements inside the loop will be evaluated k times. The object `i` can be used inside the loop to indicate which of these k passes through the loop is currently being performed.

Here's what to do inside the loop:

1. Compare `i` to `sets` to create a logical (that is, `TRUE/FALSE` vector) of the rows to be used for the **test** set. Create a new data frame, `For_Testing` that has just these rows.
2. Using logical negation (that is, `!`) to create another data frame, fill a data frame `For_Training` with the remaining rows.
3. Fit your model using `For_Training`. For simplicity in developing your function, you can use a particular model appropriate for the `mtcars` data set: `mod <- lm(mpg ~ hp + wt + am)`. Note that `mpg` is the response variable. Later on, you'll replace this with a more general statement.
4. Evaluate the model on the `For_Testing` data. You can do this with `pred_vals <- predict(mod, newdata = For_Testing)`
5. Calculate the mean square prediction error (MSPE). This will be `mean((For_Testing[["mpg"]] - pred_vals)^2 )`
6. Save the MSPE in the appropriate slot of `mspe`.

Trying out your function

At this point, you should have a function `k_fold1()` that, when evaluated with the default settings for the arguments, returns a number. That number should be very roughly similar in size to this one, which is called the "in-sample" error.

```
## [1] 5.634096
```

In-sample prediction errors are biased to be lower than cross-validated prediction errors.

Generalizing the function

When you are satisfied that your `k_fold1()` function is working, *copy* the code to create another function which we'll call `k_fold()`. This is going to be the generalization of the function to other data sets and other model formulas.

Modify `k_fold()` in the following ways:

- Replace `mpg ~ hp + wt + am` with `formula`. This will allow you to specify the formula on as the first argument.
- Replace `lm()` with `method()`. This will allow your function to use modeling types other than `lm()`
- Replace `predict()` with `predfun()`
- Replace `[["mpg"]]` with `[[as.character(formula[[2]])]]`.

Indexing formula objects enables extraction of parts of the formula. For those with an interest in computer languages, a formula can be used as a parse tree, with the operator (e.g. ~ or +, etc.) at the top and the arguments underneath.

Test out your function using the same data and formula as before, that is:

```
k_fold(formula = mpg ~ hp + wt + am, data=mtcars)
```

You should get the same answer as with `k_fold1()`.