Process Coefficients

Matt Multach

2021-10-05

Introduction

This document demo's the processing of model coefficients from a list of multiple model objects produced by the previous *hqmsa.sim.funct()* function. Note that this document walks through the processing of coefficient results from *all* lasso and elastic net models applied to the 100 data splits. There are therefore a few more steps involved compared to *Process_Coefficients_SingleModel.Rmd*. Furthermore, the final results will be saved for used in future demos.

Preamble

First, let's load the necessary packages. I've also set this code chunk to not print warnings or messages, as they are largely not helpful for the current example.

```
# This chunk loads the packages used in this workbook
library(xaringan)   # Allows active preview of report in RStudio
library(mvtnorm)    # Generates multivariate-normal data
library(magrittr)   # Used for piping
library(purrr)      # Used for mapping functions efficiently
library(data.table) # For more streamlined data structures
library(glmnet)     # For general lasso/elastic net functionality
library(hqreg)      # For LAD/Huber-loss regularization with SNCD algorithm
library(rlang)      # For parse_expr() to parse data name in k-fold subsetting functions
library(msaenet)    # For the multi-step adaptive elastic net
```

Let's also load our model data.

```
models_all <- readRDS("/Users/Matt/Desktop/GitHub_Dissertation_Scripts/Robust_Lasso_ElasticNet/Datasets/
```

Note that this data has already been processed to remove the Results vs. Error list level.

Extracting Coefficient Information

Coefficient frequency function

There are many pieces of information that might be extracted from such a rich collection of model results. However, the combined model that I am working towards is particularly interested in selection frequency for each of the potential predictors.

I'd like to add a list element that contains information about the adaptive LAD lasso models across all data splits, particularly with respect to coefficient inclusion frequency. Note that this does *not* include intercepts, given that some models do not include them.

Also note that this function relies on a list with a comparable structure to my processed models: * Level 1 of list corresponds with each data split * Level 2 of list corresponds with different model result objects from a given data split * One list element at Level 2 contains the model summary object *model.info* *which contains a list element *coefs*, a numeric vector of coefficient values (and intercept, if applicable)

```
coefs_freq <- function(models) {
       # initialize vector of predictor selection frequency, with all values set to 0
       coefs.freq <- numeric(length = 8)

       # label frequency vector with predictor names, accounting for
       names(coefs.freq) <- names(models[[1]][["model.info"]][["coefs"]][-1])

       # update each predictors' selection frequency if the corresponding coefficient value
       # # in each data split is nonzero (aka, was selected into the model)
       for(i in 1:length(models)) {
              for(j in 1:length(coefs.freq)) {
                     if(models[[i]][["model.info"]][["coefs"]][(j + 1)] != 0) {
                            coefs.freq[j] <- coefs.freq[j] + 1
                     }
              }
       }

       return(coefs.freq)
}
```

Minor functions

I'd also like to count the number of variables with 100% or 0% selection for each adaptation. I'm going to create two basic functions that I can apply during piping.

```
all <- function(data) {
        return(length(which(data == 100)
                      )
               )
}
```

```
none <- function(data) {
        return(length(which(data == 0)
                      )
        )
}
```

Coefficient processing: Initialize list

Coefficient frequency processing: By Adaptation

Now let's map our coefficient frequency function across all adaptations and model results.

```
models_all[["Frequencies"]] <- models_all[1:7] %>%
         map(coefs_freq) %>%
         setDT %>%
         t %>%
         data.frame %>%
         setDT %>%
```

```
        .[ , "all" := apply(. , 1 , all)] %>%
        .[ , "none" := apply(. , 1 , none)] %>%
        .[ , "method" := names(models_all[-8])] %>%
        setkey(. , method)
```

Let's walk through what's going on in the last code chunk. * First, establish reference to the *models_all* object for the subsequent pipe operations * Second, *map()* the *coefs_freq()* function across elements of *models_all* * Third, make the resulting object a data.table using *setDT()* function * Fourth, transpose the result using *t()*, since the result will be an $8x7$ object with rows corresponding with each possible predictor and columns corresponding with each model/adaptation * As a reminder, our desired object is a $7x8$ object, with model/adaptation in each row and possible predictor frequency in each column * Fifth, make the result a dataframe * Sixth, make the result a data.table, *again* * I can't quite explain why we need to conduct the steps above in the exact order (and repetition) in the pipes above. There's a sequence of transformations on the unstored/transitory object that impact what the result will be that mean *t()* has to go in the middle, and *data.frame()* at the end. But then we have to reset the object as a data.table after *data.frame()*. * Seventh, create a column of 100%-selection variables by mapping the *all()* function over the rows of our intermediate data.table * Eighth, create a column of 0%-selection variables by mapping the *none()* function over the rows of our intermediate data.table * Ninth, make a column indicating the adaptation for each row

Because I'm currently learning about keys, the last operation of the pipe sets a key for the *"Frequences"* data.table based on the method/adaptation. (If you don't know what this means, don't worry too much about it. It won't generally come up for the purposes of demonstrating my combined model. Right now its only practical significance is alphabetically sorting *"Frequencies"* by method/adaptation).

Finally, let's take a quick look at the structure of our coefficient frequencies object. Note that if we had not set the key, the final line would not be present. This tells us that the data.table is sorted by the character values in *"method"*.

```
str(models_all[["Frequencies"]])
```

```
## Classes 'data.table' and 'data.frame':   7 obs. of  11 variables:
##  $ X1    : num  100 100 100 100 100 100 99
##  $ X2    : num  100 100 100 100 100 100 100
##  $ X3    : num  100 100 100 100 100 100 100
##  $ X4    : num  100 100 100 100 100 100 100
##  $ X5    : num  5 4 3 4 5 4 3
##  $ X6    : num  0 2 0 0 0 0 0
##  $ X7    : num  7 3 9 9 2 4 4
##  $ X8    : num  46 23 36 12 6 15 6
##  $ all   : int  4 4 4 4 4 4 3
##  $ none  : int  1 0 1 1 1 1 1
##  $ method: chr  "huberelnet" "huberlasso" "ladelnet" "ladlasso" ...
##  - attr(*, ".internal.selfref")=<externalptr>
##  - attr(*, "sorted")= chr "method"
```

Save the Last ~~Dance~~ File

Let's save the model results file which also includes our new "Frequencies" data.table so that we can load it during the full walkthrough.

```
saveRDS(models_all , "/Users/Matt/Desktop/GitHub_Dissertation_Scripts/Robust_Lasso_ElasticNet/Datasets/m
```