Lasso CV

Tucker Morgan - tlm2152

3/20/2022

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.1.2
## Warning: package 'ggplot2' was built under R version 4.1.2
## Warning: package 'tibble' was built under R version 4.1.2
## Warning: package 'tidyr' was built under R version 4.1.2
## Warning: package 'dplyr' was built under R version 4.1.2
```

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.1.3
```

```
source("./shared_code/setup.R")
```

```
## Warning: package 'beepr' was built under R version 4.1.2
## Warning: package 'gtsummary' was built under R version 4.1.3
```

```
source("./shared_code/partition.R")
source("./shared_code/data_prep.R")
source("./shared_code/logistic_lasso.R")
source("./shared_code/auc_calc_lasso.R")
```

```
## Warning: package 'pROC' was built under R version 4.1.3
```

```
part_bc <- partition(p = 0.8, data = bc)

bc_trn <-
  part_bc %>%
  filter(part_id == "train") %>%
  select(-part_id)

bc_tst <-
  part_bc %>%
  filter(part_id == "test") %>%
  select(-part_id)
```

```
source("./shared_code/cv_folding.R")

bc_trn_folds <-
  cv_sets(training = bc_trn) %>%
  select(-fold_p)
```

```r
set.seed(100)

X <- bc_trn_folds[, -c(1,32)]
X <- as.matrix(X)
Y <- bc_trn_folds$diagnosis
lambda_vec <- seq(0, 0.4, length = 5) # lambda vector for testing

# creating a simple example function for testing
ex_func <- function(x, y, lambda_vec){
  glmnet(x = x, y = y,
         standardize = TRUE,
         alpha = 1,
         lambda = lambda_vec,
         family = "binomial"(link = "logit"))
}

ex_fit <- ex_func(x = X, y = Y, lambda_vec = 0) #%>% coef() # just for example, not stored

cv_function <- function(k = 5, training, func, lambda_vec){

  auc_list = list()
  mean_auc_list = list()
# first, a for loop to iterate over a lambda vector
  for (j in 1:length(lambda_vec)){
    # and now we have a for loop to iterate over each fold, k = 5 here
    for (i in 1:k){
      # this will identify the training set as not i
      trn_set =
        training %>%
        filter(fold_id != i) %>%
        select(-fold_id)
      # and this assigns i to be the test set
      tst_set =
        training %>%
        filter(fold_id == i) %>%
        select(-fold_id)
      # making matrices
      X_trn <- as.matrix(trn_set[,-1])
      X_tst <- as.matrix(tst_set[,-1])
      Y_trn <- trn_set$diagnosis
      # fitting our function based on training set
      trn_fit = func(x = X_trn, y = Y_trn, lambda_vec = lambda_vec[j])
      # calculating AUC
      trn_pred <- predict(trn_fit,
                          newx = X_tst,
                          type = "response")
      trn_roc <- pROC::roc(tst_set$diagnosis, trn_pred)

      auc_list[[i]] = trn_roc$auc
    }
    # calculating mean cv auc for each lambda
    auc_df = data.frame("auc" = do.call(rbind, auc_list))
    mean_auc = mean(auc_df$auc)
```

```
    mean_auc_list[[j]] = data.frame("mean_auc" = mean_auc, "lambda" = lambda_vec[j])
  }
  # creating dataframe to show lambda values and corresponding mean AUC
  res = as.data.frame(do.call(rbind, mean_auc_list))

  return(res)
}

cv_function(training = bc_trn_folds, func = ex_func, lambda_vec = lambda_vec)
```

The cross-validation function seems to work as intended, let's try updating to work with Jimmy's lasso function.

```
identity <- function(x){
  return(x)
}

lambda_init <- function(start, stop, step, func = identity){
  lambda_vec <- seq(start, stop, step) %>% func()
  return(lambda_vec)
}

cv_jt <- function(k = 5, training, func, lambda_list){

  lam_start <- lambda_list[[1]]
  lam_stop <- lambda_list[[2]]
  lam_step <- lambda_list[[3]]
  lam_func <- lambda_list[[4]]

  lam_list <- tibble(
    lam_count = 0,
    lam_start = 0,
    lam_stop = 0,
    lam_step = 0
  )

  lam_count <- 0
  del_too_many_var <- 1
  out_res <- list()

 while (del_too_many_var > 0) {

 lam_count <- lam_count + 1

  # saving lambda vector parameters
 cur_lam_list <- tibble(lam_count, lam_start, lam_stop, lam_step)


 lam_list <- bind_rows(lam_list, cur_lam_list)

 new_lambda_vec <- lambda_init(lam_start, lam_stop, lam_step, lam_func)
 lasso_list <- list()
 auc_list <- list()
```

```
  for (i in 1:k) {

    # this will identify the training set as not i
    trn_set =
      training %>%
      filter(fold_id != i) %>%
      select(-fold_id)

    # and this assigns i to be the test set
    tst_set =
      training %>%
      filter(fold_id == i) %>%
      select(-fold_id) %>%
      rename(y = diagnosis)

    # making matrices
    X_trn <- trn_set[,-1]
    Y_trn <- trn_set$diagnosis

    # lasso_list
    lasso_list <- func(inputs = X_trn, output = Y_trn, lambda_vec = new_lambda_vec)

    lasso_lambda <- lasso_list[[1]]
    lasso_beta <- lasso_list[[2]]
    lasso_selected <- tibble(selected_num = lasso_list[[3]])

    lasso_lam_bet <- cbind(lasso_lambda, lasso_beta) %>% as.matrix()

    trn_roc <- auc_calc_lasso(lasso_lam_bet, tst_set)

    auc_list <- bind_rows(auc_list, trn_roc)

  }

  auc_res <-
    auc_list %>%
    group_by(lambda) %>%
    summarise(mean_auc = mean(auc_vals))

  res <- bind_cols(auc_res, lasso_selected)

  res <- res %>%
    mutate(num_dropped_vars = selected_num - lag(selected_num, 1))

  del_too_many_var <- sum(na.omit(res$num_dropped_vars) > 1)

    if (del_too_many_var > 0) {
      max_auc_lam <- res %>% filter(mean_auc == max(mean_auc)) %>% pull(lambda)
      lam_start <- max_auc_lam + 2*abs(lam_step)
      lam_stop <- max_auc_lam - 2*abs(lam_step)
      lam_step <- sign(lam_step)*(lam_start - lam_stop)/length(new_lambda_vec)
    }
}
```

```r
  # creating dataframe to show lambda values and corresponding mean AUC
  out_res[[1]] <- res
  out_res[[2]] <- lam_list

  return(out_res)
}

# test lambda list
lambda_list <- list(0.4, 0.20, -0.01, identity)
cv_res <- cv_jt(k = 5, training = bc_trn_folds, func = logistic_lasso, lambda_list = lambda_list)

cv_res[[1]]
```

```
## # A tibble: 22 x 4
##    lambda mean_auc selected_num num_dropped_vars
##     <dbl>    <dbl>        <dbl>            <dbl>
##  1  0.31     0.970            2               NA
##  2  0.312    0.970            2                0
##  3  0.314    0.970            2                0
##  4  0.316    0.970            2                0
##  5  0.318    0.970            2                0
##  6  0.320    0.970            2                0
##  7  0.321    0.970            2                0
##  8  0.323    0.970            2                0
##  9  0.325    0.970            2                0
## 10  0.327    0.970            2                0
## # ... with 12 more rows
```

```r
cv_res[[2]]
```

```
## # A tibble: 3 x 4
##   lam_count lam_start lam_stop lam_step
##       <dbl>     <dbl>    <dbl>    <dbl>
## 1         0      0         0     0
## 2         1      0.4       0.2  -0.01
## 3         2      0.35      0.31 -0.00190
```