Lasso CV

Tucker Morgan - tlm2152

3/20/2022

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

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

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

Folding Data

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

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

Helper Functions

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

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

Cross Validation Function

```
#################
#
# This function takes in the following parameters:
# > k: number of folds
# > training: training dataset, first column is outcome, remaining columns are predictors.
# > lambda_list: list containing the following:
#     - lambda_start: starting value of lambda vector, without lam_start_stop_func applied
#     - lambda_stop: stopping value of lambda vector, without lam_start_stop_func applied
#     - lambda_step: step size between numbers in linear sequence
#     - lambda_func: transformation of initial sequence of numbers
#     - lambda_start_stop_func: function applied to lambda_start/lambda_stop
#
#
```

```r
# This function returns a list with two elements:
# > A matrix containing the results of the last CV:
#     - lambda: a column of lambdas
#     - mean_auc: mean AuC
#     - selected_vars: number of selected variables column (excluding intercept)
#     - num_dropped_Vars: number of variables deleted from previous lambda
# > A matrix containing the lambda vectors that were attempted.
#     - lambda_count: id of lambda vector/iteration number
#     - lambda_start: starting value of lambda vector, *without* the lam_start_stop_func
#       applied (i.e., in linear/untransformed units)
#     - lambda_stop: stopping value of lambda vector, *without* the lam_start_stop_func
#       applied (i.e., in linear/untransformed units)
#     - lambda_step: step size between lambdas
#       if lambda_start > lambda_stop, this should be negative.
#
#################

cv_jt <- function(k = 5, training, func, lam_start_stop_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) {
     # print("working")

     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()

     pb <- progress_bar$new(
        format = " lasso-ing [:bar] :percent eta: :eta",
        total = 5, clear = FALSE, width = 60)
```

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

      pb$tick()

      # 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

      #print("about to lasso")

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

      #print("done with lasso")

      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)

    # del_too_many_var <- 0
```

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

  # 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)

# identify minimum lambda value for which all coefficients are zero
#
# set lambda_min based on scaled data
# lambda_min <- 0.0001
# define vector of lambdas
# lambda_seq <- exp(seq(log(lambda_max), log(lambda_min), length.out = 100))

lambda_max <- max(t(scale(as.matrix(bc_trn[,-1]))) %*% bc_trn[,1]) / nrow(bc_trn[,-1])

lambda_list <- list(log(lambda_max), log(0.0001), -(log(lambda_max) - log(0.0001))/100, exp)

cv_res <- cv_jt(k = 5, training = bc_trn_folds, func = logistic_lasso, lam_start_stop_func = log, lambda

cv_res[[1]] %>% knitr::kable()
```

lambda	mean_auc	selected_num	num_dropped_vars
0.0481050	0.9800836	6	NA
0.0482621	0.9800414	6	0
0.0484198	0.9800414	6	0
0.0485779	0.9800414	6	0
0.0487366	0.9800414	6	0
0.0488958	0.9801609	6	0
0.0490556	0.9801609	6	0
0.0492158	0.9801609	6	0
0.0493766	0.9802859	6	0
0.0495379	0.9802859	6	0
0.0496997	0.9802859	6	0
0.0498620	0.9802859	6	0
0.0500249	0.9802859	6	0
0.0501883	0.9802859	6	0
0.0503523	0.9802859	6	0
0.0505167	0.9802859	6	0
0.0506817	0.9804762	6	0
0.0508473	0.9804762	6	0
0.0510134	0.9804762	6	0

lambda	mean_auc	selected_num	num_dropped_vars
0.0511800	0.9806012	6	0
0.0513472	0.9803895	6	0
0.0515149	0.9803895	6	0
0.0516832	0.9803895	6	0
0.0518521	0.9802837	6	0
0.0520214	0.9801779	6	0
0.0521914	0.9802623	6	0
0.0523618	0.9802623	6	0
0.0525329	0.9802623	6	0
0.0527045	0.9802623	6	0
0.0528767	0.9802623	6	0
0.0530494	0.9802623	6	0
0.0532227	0.9802623	6	0
0.0533965	0.9803873	6	0
0.0535709	0.9803873	6	0
0.0537459	0.9803029	6	0
0.0539215	0.9803029	6	0
0.0540976	0.9803029	6	0
0.0542744	0.9803029	6	0
0.0544516	0.9803029	6	0
0.0546295	0.9804279	6	0
0.0548080	0.9804279	6	0
0.0549870	0.9805337	6	0
0.0551666	0.9805337	6	0
0.0553468	0.9805337	6	0
0.0555276	0.9804279	6	0
0.0557090	0.9804279	6	0
0.0558910	0.9804279	6	0
0.0560735	0.9804279	6	0
0.0562567	0.9804279	6	0
0.0564405	0.9804279	6	0
0.0566248	0.9804279	6	0
0.0568098	0.9804279	6	0
0.0569954	0.9804279	6	0
0.0571816	0.9804279	6	0
0.0573684	0.9804279	6	0
0.0575558	0.9804279	6	0
0.0577438	0.9804279	6	0
0.0579324	0.9803221	6	0
0.0581216	0.9802798	6	0
0.0583115	0.9802376	6	0
0.0585020	0.9802376	6	0
0.0586931	0.9801181	6	0
0.0588848	0.9801181	6	0
0.0590771	0.9801181	6	0
0.0592701	0.9799065	6	0
0.0594637	0.9799065	6	0
0.0596580	0.9799065	6	0
0.0598528	0.9799065	6	0
0.0600484	0.9798007	6	0
0.0602445	0.9798007	6	0
0.0604413	0.9798007	6	0

lambda	mean_auc	selected_num	num_dropped_vars
0.0606387	0.9799257	6	0
0.0608368	0.9799257	6	0
0.0610355	0.9798412	6	0
0.0612349	0.9798412	6	0
0.0614350	0.9794909	6	0
0.0616356	0.9794909	6	0
0.0618370	0.9793765	6	0
0.0620390	0.9794823	6	0
0.0622416	0.9794823	6	0
0.0624449	0.9795881	6	0
0.0626489	0.9795881	6	0
0.0628536	0.9795881	6	0
0.0630589	0.9796073	6	0
0.0632649	0.9796073	6	0
0.0634715	0.9796918	6	0
0.0636789	0.9796918	6	0
0.0638869	0.9796918	6	0
0.0640956	0.9796918	6	0
0.0643049	0.9796918	6	0
0.0645150	0.9796918	6	0
0.0647257	0.9795860	6	0
0.0649372	0.9795860	6	0
0.0651493	0.9795860	6	0
0.0653621	0.9795860	6	0
0.0655756	0.9794610	6	0
0.0657898	0.9794610	6	0
0.0660047	0.9794746	6	0
0.0662203	0.9794746	6	0
0.0664366	0.9793688	6	0
0.0666537	0.9793338	6	0
0.0668714	0.9793338	6	0

```r
cv_res[[2]] %>% knitr::kable()
```

lam_count	lam_start	lam_stop	lam_step
0	0.0000000	0.000000	0.0000000
1	-0.9757123	-9.210340	-0.0823463
2	-2.7049842	-3.034369	-0.0032612

Final Lasso-Logistic Model with Selected Lambda

```r
# selected lambda: 0.05118004
selected_lambda <- cv_res[[1]] %>% filter(mean_auc == max(mean_auc)) %>% pull(lambda)
lasso_final_model <- logistic_lasso(inputs = bc_trn[,-1], output = bc_trn[,1], lambda_vec = selected_lar
save(lasso_final_model, file = "lasso_results_wq.Rdata")
lasso_betas <- lasso_final_model[[2]] %>% t()
```

Lasso AUC

```
auc_calc_full <- function(beta_est, test_data){
  # pulling out the terms used in the full model (should be all)

  # we have this flexible in case we want to test fewer variables
  # terms <- beta_est %>% pull(term)
  # col.num <- which(colnames(test_data) %in% terms)
  # select the desired x values
  test_data = bc_tst
  xvals <- test_data[,-1] %>%
     mutate(
      intercept = 1 # create a intercept variable
    ) %>%
    relocate(intercept) # move it to the front
  pred <- as.matrix(xvals) %*% beta_est # get the cross product of the linear model
  logit_pred <- exp(pred) / (1 + exp(pred)) # link function to get probabilities

  auc_val <- auc(test_data[,1], as.vector(logit_pred)) # calculating the AUC

  # roc(tst_data$y, as.vector(logit_pred)) %>% plot( legacy.axes=TRUE) # graphs AUC
  return(auc_val)

}

lasso_auc <- auc_calc_full(lasso_betas, bc_tst)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

The AUC of the logistic-lasso model is 0.9659125.