Lasso CV

Tucker Morgan - tlm2152

3/20/2022

```
set.seed(100)
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)

  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
0.0375753	0.9781980	5	NA
0.0376981	0.9763913	5	0
0.0378212	0.9782700	5	0
0.0379447	0.9808587	5	0
0.0380687	0.9782082	5	0
0.0381930	0.9765992	5	0
0.0383178	0.9787449	5	0
0.0384430	0.9776039	5	0
0.0385685	0.9817535	5	0
0.0386945	0.9800282	5	0
0.0388209	0.9815476	5	0
0.0389477	0.9800001	5	0
0.0390750	0.9799716	5	0
0.0392026	0.9811053	5	0
0.0393307	0.9808587	5	0
0.0394591	0.9808790	5	0
0.0395880	0.9814296	5	0

lambda	mean_auc	selected_num	num_dropped_vars
0.0397174	0.9813628	5	0
0.0398471	0.9816042	5	0
0.0399773	0.9812286	5	0
0.0401078	0.9821401	5	0
0.0402389	0.9822585	5	0
0.0403703	0.9824331	5	0
0.0405022	0.9811453	5	0
0.0406345	0.9824306	5	0
0.0407672	0.9825438	5	0
0.0409004	0.9813175	5	0
0.0410340	0.9824306	5	0
0.0411680	0.9824148	5	0
0.0413025	0.9823867	5	0
0.0414374	0.9823379	5	0
0.0415728	0.9824974	5	0
0.0417086	0.9824872	5	0
0.0418448	0.9824974	5	0
0.0419815	0.9824872	5	0
0.0421186	0.9824510	5	0
0.0422562	0.9824408	5	0
0.0423943	0.9824587	5	0
0.0425327	0.9826080	5	0
0.0426717	0.9825051	5	0
0.0428111	0.9825617	5	0
0.0429509	0.9825153	5	0
0.0430912	0.9825617	5	0
0.0432320	0.9825617	5	0
0.0433732	0.9826182	5	0
0.0435149	0.9825617	5	0
0.0436570	0.9825718	5	0
0.0437996	0.9826748	5	0
0.0439427	0.9826182	5	0
0.0440862	0.9826748	5	0
0.0442303	0.9826182	5	0
0.0443747	0.9826748	5	0
0.0445197	0.9826748	5	0
0.0446651	0.9826207	5	0
0.0448110	0.9825100	5	0
0.0449574	0.9825666	5	0
0.0451043	0.9825100	5	0
0.0452516	0.9825666	5	0
0.0453994	0.9825202	5	0
0.0455477	0.9825666	5	0
0.0456965	0.9825202	5	0
0.0458458	0.9825666	5	0
0.0459955	0.9826231	5	0
0.0461458	0.9825666	5	0
0.0462965	0.9824113	5	0
0.0464477	0.9823649	5	0
0.0465995	0.9824678	5	0
0.0467517	0.9824215	5	0
0.0469044	0.9824215	5	0

lambda	mean_auc	selected_num	num_dropped_vars
0.0470576	0.9824215	6	1
0.0472113	0.9821531	6	0
0.0473655	0.9821531	6	0
0.0475203	0.9821531	6	0
0.0476755	0.9822096	6	0
0.0478312	0.9822662	6	0
0.0479875	0.9821734	6	0
0.0481442	0.9821734	6	0
0.0483015	0.9820806	6	0
0.0484593	0.9820806	6	0
0.0486176	0.9820806	6	0
0.0487764	0.9820806	6	0
0.0489357	0.9820806	6	0
0.0490956	0.9819254	6	0
0.0492559	0.9819254	6	0
0.0494168	0.9817701	6	0
0.0495783	0.9817701	6	0
0.0497402	0.9817237	6	0
0.0499027	0.9817237	6	0
0.0500657	0.9816773	6	0
0.0502292	0.9816773	6	0
0.0503933	0.9816773	6	0
0.0505579	0.9816773	6	0
0.0507231	0.9816773	6	0
0.0508888	0.9816773	6	0
0.0510550	0.9815846	6	0
0.0512218	0.9815846	6	0
0.0513891	0.9815846	6	0
0.0515570	0.9815846	6	0
0.0517254	0.9815846	6	0
0.0518943	0.9814918	6	0
0.0520639	0.9814918	6	0
0.0522339	0.9814918	6	0

```
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	-2.9520230	-3.281408	-0.0032612

Final Lasso-Logistic Model with Selected Lambda

```
# selected lambda: 0.05118004
selected_lambda <- cv_res[[1]] %>% filter(mean_auc == max(mean_auc)) %>% pull(lambda) %>% mean()
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 0.9650075.