gbm

Laura Cosgrove

5/7/2019

```
knitr::opts_chunk$set(eval = FALSE)

library(tidyverse)
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
## Registered S3 method overwritten by 'rvest':
##   method            from
##   read_xml.response xml2
```

```
## -- Attaching packages ---------------------------------------------------------------------------------
```

```
## v ggplot2 3.1.1       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------------------------------- t
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(ranger)
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(xgboost)
```

```
##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
# Using caret
ctrl1 <- trainControl(method = "repeatedcv",
                      repeats = 5,
                      summaryFunction = twoClassSummary, #because we're in the two-class setting
                      classProbs = TRUE) #because need predicted class probabilities to get ROC curve

#Read RDS
cog_train <- readRDS("./data/cog_train_preproc.RDS")
cog_test <- readRDS("./data/cog_test_preproc.RDS")
```

```r
set.seed(1)

#tuning
gbm_grid <- expand.grid(n.trees = c(2000,3000),
                        interaction.depth = 2:10,
                        shrinkage = c(0.01, 0.03, 0.05),
                        n.minobsinnode = 1)

gbm_fit <- train(x = cog_train[3:10],
                 y = cog_train$cdr,
                 method = "gbm",
                 tuneGrid = gbm_grid,
                 trControl = ctrl1,
                 verbose = FALSE)
```

```r
#Save and reload
#saveRDS(gbm_fit, file = "./data/gbm_fit_1.RDS")
gbm_fit_1 = readRDS("./data/gbm_fit_1.RDS")

ggplot(gbm_fit_1, highlight = TRUE)

gbm_fit_1$results[which.max(gbm_fit_1$results$ROC),]
```

```r
set.seed(12)

gbm_grid_2 <- expand.grid(n.trees = 2000,
                          interaction.depth = 2:8,
                          shrinkage = c(0.0008, 0.001, 0.004),
                          n.minobsinnode = 1)

gbm_fit_2 <- train(x = cog_train[3:10],
                   y = cog_train$cdr,
                   method = "gbm",
                   tuneGrid = gbm_grid_2,
                   trControl = ctrl1,
                   verbose = FALSE)
```

```r
#Save and reload
#saveRDS(gbm_fit_2, file = "./data/gbm_fit_3.RDS")
```

```r
gbm_fit_2 = readRDS("./data/gbm_fit_2.RDS")

ggplot(gbm_fit_2, highlight = TRUE)

gbm_fit_2$results[which.max(gbm_fit_2$results$ROC),]
```

```r
set.seed(12)

gbm_grid_3 <- expand.grid(n.trees = c(2000, 5000),
                          interaction.depth = 4:10,
                          shrinkage = 0.001,
                          n.minobsinnode = 1)

gbm_fit_3 <- train(x = cog_train[3:10],
                   y = cog_train$cdr,
                   distribution = "bernoulli",
                   method = "gbm",
                   tuneGrid = gbm_grid_3,
                   trControl = ctrl1,
                   verbose = FALSE)
```

```r
#Save and reload
#saveRDS(gbm_fit_3, file = "./data/gbm_fit_3.RDS")
gbm_fit_3 = readRDS("./data/gbm_fit_3.RDS")

ggplot(gbm_fit_3, highlight = TRUE)


gbm_fit_3$results[which.max(gbm_fit_3$results$ROC),]

## variable importance
summary.gbm(gbm_fit_3$finalModel)
```

Test predictions, using gbm model 3:

```r
##Test Predictions##

pred_gbm_raw <- predict(gbm_fit_3, newdata = cog_test,
                    n.trees = 5000,
                    type = "raw")

confusionMatrix(data = pred_gbm_raw,
                reference = cog_test$cdr,
                positive = "Dementia")

pred_gbm_prob <- predict(gbm_fit_3, newdata = cog_test,
                    n.trees = 5000,
                    type = "prob")

roc_gbm_test <- roc(cog_test$cdr, pred_gbm_prob$Dementia)

plot(roc_gbm_test, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc_gbm_test), col = 4, add = TRUE)
```

xgboost

```
library(xgboost)
```

4.1 Step 1: Number of Iterations and the Learning Rate

```
set.seed(1)

nrounds <- 1000

tune_grid <- expand.grid(
  nrounds = seq(from = 200, to = nrounds, by = 50),
  eta = c(0.025, 0.05, 0.1, 0.3),
  max_depth = c(2, 3, 4, 5, 6),
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = 1,
  subsample = 1
)

tune_control <- caret::trainControl(
  method = "cv", # cross-validation
  number = 3, # with n folds
  #index = createFolds(tr_treated$Id_clean), # fix the folds
  verboseIter = FALSE, # no training log
  allowParallel = TRUE, # FALSE for reproducible results
  summaryFunction = twoClassSummary, #because we're in the two-class setting
                     classProbs = TRUE
)

xgb_tune <- caret::train(
  cog_train[3:10],
  y = cog_train$cdr,
  metric = "ROC",
  trControl = tune_control,
  tuneGrid = tune_grid,
  method = "xgbTree",
  verbose = TRUE
)

# helper function for the plots
tuneplot <- function(x, probs = .30) {
  ggplot(x) +
    coord_cartesian(ylim = c(quantile(x$results$ROC, probs = probs), max(x$results$ROC))) +
    theme_bw()
}

tuneplot(xgb_tune)

xgb_tune$bestTune

xgb_tune$results[which.max(xgb_tune$results$ROC),]
```

4.2 Step 2: Maximum Depth and Minimum Child Weight After fixing the learning rate to 0.05 and we'll also set maximum depth to 3 +-1 (or +2 if max_depth == 2) to experiment a bit around the suggested best tune in previous step. Then, well fix maximum depth and minimum child weight:

```
set.seed(1)

tune_grid2 <- expand.grid(
  nrounds = seq(from = 50, to = nrounds, by = 50),
  eta = xgb_tune$bestTune$eta,
  max_depth = ifelse(xgb_tune$bestTune$max_depth == 2,
    c(xgb_tune$bestTune$max_depth:4),
    xgb_tune$bestTune$max_depth - 1:xgb_tune$bestTune$max_depth + 1),
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = c(1, 2, 3),
  subsample = 1
)

xgb_tune2 <- caret::train(
  x = cog_train[3:10],
  y = cog_train$cdr,
  metric = "ROC",
  trControl = tune_control,
  tuneGrid = tune_grid2,
  method = "xgbTree",
  verbose = TRUE
)

tuneplot(xgb_tune2)

xgb_tune2$bestTune

xgb_tune2$results[which.max(xgb_tune2$results$ROC),]
```

4.3 Step 3: Column and Row Sampling Based on this, we can fix minimum child weight to 2 and maximum depth to 2. Next, we'll try different values for row and column sampling:

```
set.seed(1)

tune_grid3 <- expand.grid(
  nrounds = seq(from = 50, to = nrounds, by = 50),
  eta = xgb_tune$bestTune$eta,
  max_depth = xgb_tune2$bestTune$max_depth,
  gamma = 0,
  colsample_bytree = c(0.4, 0.6, 0.8, 1.0),
  min_child_weight = xgb_tune2$bestTune$min_child_weight,
  subsample = c(0.5, 0.75, 1.0)
)

xgb_tune3 <- caret::train(
  x = cog_train[3:10],
  y = cog_train$cdr,
  metric = "ROC",
  trControl = tune_control,
  tuneGrid = tune_grid3,
  method = "xgbTree",
```

```
  verbose = TRUE
)

tuneplot(xgb_tune3, probs = .5)

xgb_tune3$bestTune

xgb_tune3$results[which.max(xgb_tune3$results$ROC),]
```

4.4 Step 4: Gamma Next, we again pick the best values from previous step, and now will see whether changing the gamma has any effect on the model fit:

```
set.seed(1)
tune_grid4 <- expand.grid(
  nrounds = seq(from = 50, to = nrounds, by = 50),
  eta = xgb_tune$bestTune$eta,
  max_depth = xgb_tune2$bestTune$max_depth,
  gamma = c(0, 0.05, 0.1, 0.5, 0.7, 0.9, 1.0),
  colsample_bytree = xgb_tune3$bestTune$colsample_bytree,
  min_child_weight = xgb_tune2$bestTune$min_child_weight,
  subsample = xgb_tune3$bestTune$subsample
)

xgb_tune4 <- caret::train(
  x = cog_train[3:10],
  y = cog_train$cdr,
  metric = "ROC",
  trControl = tune_control,
  tuneGrid = tune_grid4,
  method = "xgbTree",
  verbose = TRUE
)

tuneplot(xgb_tune4)

xgb_tune4$results[which.max(xgb_tune4$results$ROC),]
```

4.5 Step 5: Reducing the Learning Rate

```
set.seed(1)

tune_grid5 <- expand.grid(
  nrounds = seq(from = 100, to = 10000, by = 100),
  eta = c(0.01, 0.015, 0.025, 0.05, 0.1),
  max_depth = xgb_tune2$bestTune$max_depth,
  gamma = xgb_tune4$bestTune$gamma,
  colsample_bytree = xgb_tune3$bestTune$colsample_bytree,
  min_child_weight = xgb_tune2$bestTune$min_child_weight,
  subsample = xgb_tune3$bestTune$subsample
)

xgb_tune5 <- caret::train(
  x = cog_train[3:10],
  y = cog_train$cdr,
  metric = "ROC",
```

```
  trControl = tune_control,
  tuneGrid = tune_grid5,
  method = "xgbTree",
  verbose = TRUE
)

tuneplot(xgb_tune5)

xgb_tune5$results[which.max(xgb_tune5$results$ROC),]
```

Final grid:

```
ctrl1 <- trainControl(method = "repeatedcv",
                      repeats = 5,
                      summaryFunction = twoClassSummary,
                      classProbs = TRUE) #because need predicted class probabilities to get ROC curve

final_grid <- expand.grid(
  nrounds = xgb_tune5$bestTune$nrounds,
  eta = xgb_tune5$bestTune$eta,
  max_depth = xgb_tune5$bestTune$max_depth,
  gamma = xgb_tune5$bestTune$gamma,
  colsample_bytree = xgb_tune5$bestTune$colsample_bytree,
  min_child_weight = xgb_tune5$bestTune$min_child_weight,
  subsample = xgb_tune5$bestTune$subsample)

final_xbg = caret::train(
 x = cog_train[3:10],
  y = cog_train$cdr,
   metric = "ROC",
  trControl = ctrl1,
  tuneGrid = final_grid,
  method = "xgbTree",
  verbose = TRUE
)

final_xbg

saveRDS(final_xbg, "./data/xgboost.RDS")
```

#xg boost test pred

```
final_xbg = readRDS("./data/xgboost.RDS")

#model prediction on test data
xgbpred <- predict(final_xbg, cog_test)

confusionMatrix(xgbpred, reference = cog_test$cdr)

#roc test
xgbpred_prob <- predict(final_xbg, cog_test, type = "prob")

roc_xboost_test <- roc(cog_test$cdr, xgbpred_prob$Dementia)

plot(roc_xboost_test, legacy.axes = TRUE, print.auc = TRUE)
```

```
plot(smooth(roc_xboost_test), col = 4, add = TRUE)

#roc train
xgbpred_train_prob <- predict(final_xbg, cog_train, type = "prob")
roc_xboost_train <- roc(cog_train$cdr, xgbpred_train_prob$Dementia)
plot(roc_xboost_train, legacy.axes = TRUE, print.auc = TRUE)
plot(smooth(roc_xboost_train), col = 4, add = TRUE)

#importance
mat <- xgb.importance(feature_names = colnames(cog_train[3:10]),
                      model = final_xbg$finalModel)
xgb.plot.importance(importance_matrix = mat)
```