Problem Set 5

Pete Cuppernull

3/1/2020

Load Packages and Data

```r
library(tidyverse)
library(tidymodels)
library(rsample)
library(glmnet)
library(leaps)
library(rcfss)
library(patchwork)
library(caret)
library(h2o)
library(ipred)
library(adabag)
library(randomForest)
library(ranger)
library(gbm)
library(tree)
library(pROC)

gss_test <- na.omit(read_csv("data/gss_test.csv"))
gss_train <- na.omit(read_csv("data/gss_train.csv"))

set.seed(1414)
```

Conceptual: Cost functions for classification trees

When growing a decision tree, either the GINI index or cross-entropy are preferable to misclassification rate because they have a greater capacity to produce pure nodes. The best pick between these two approaches is sometimes context dependent, but on average the GINI index would likely be preferable because it is less computationally expensive than cross-entropy (as it does not require computing logarithms). When pruning a decision tree, typically the misclassification rate is used (see ESL, p. 310).

Application

2. Estimate the Models

Logit

```r
gss_train_logit <- as_tibble(gss_train) %>%
  mutate(colrac = factor(colrac))

# function to generate assessment statistics for titanic model
holdout_results <- function(splits) {
  # Fit the model to the training set
  mod <- glm(colrac ~ ., data = analysis(splits),
             family = binomial)

  # `augment` will save the predictions with the holdout data set
  res <- augment(mod, newdata = assessment(splits)) %>%
    as_tibble() %>%
    mutate(.prob = logit2prob(.fitted),
           .pred = round(.prob))

  # Return the assessment data set with the additional columns
  res
}

gss_logit_cv10 <- vfold_cv(data = gss_train_logit, v = 10) %>%
  mutate(results = map(splits, holdout_results)) %>%
  unnest(results) %>%
  mutate(.pred = factor(.pred)) %>%
  group_by(id) %>%
  accuracy(truth = colrac, estimate = .pred)

logit_error <- 1 - mean(gss_logit_cv10$.estimate, na.rm = TRUE)


##roc
logit_roc <- vfold_cv(data = gss_train_logit, v = 10) %>%
  mutate(results = map(splits, holdout_results)) %>%
  unnest(results) %>%
  group_by(id) %>%
  roc(response = colrac, predictor = .pred)
```

Naive Bayes

```r
gss_train_x <- model.matrix(colrac ~ ., gss_train)[, -1]
gss_train_y <- gss_train$colrac

gss_test_x <- model.matrix(colrac ~ ., gss_test)[, -1]
gss_test_y <- gss_test$colrac

gss_train_caret <- gss_train %>%
  mutate(colrac = make.names(as.character(if_else(colrac == 1, TRUE, FALSE))))

ctrl <- trainControl(method = "cv",     # Cross-validation
                     number = 10,      # 10 folds
                     classProbs = TRUE,                  # For AUC
                     summaryFunction = twoClassSummary)  # For AUC

nb_grid <- expand.grid(
  usekernel = TRUE,
  fL = 1,
  adjust = seq(.5, 2, by = .5)
)


nb_cv <- train(colrac ~ .,
                       data = gss_train_caret,
                       method = 'nb',
                       metric = 'Accuracy',
                       tuneGrid = nb_grid,
                       trControl = ctrl)

#error
nb_error2 <- 1- ((nb_cv$results$Sens[which.max(nb_cv$results$ROC)]*sum(gss_train$colrac)) +

   (nb_cv$results$Spec[which.max(nb_cv$results$ROC)]*(nrow(gss_train)
                     - sum(gss_train$colrac)))) / nrow(gss_train)
```

Elastic net regression

```
lasso    <- glmnet(gss_train_x, gss_train_y, alpha = 1.0)
elastic1 <- glmnet(gss_train_x, gss_train_y, alpha = 0.25)
elastic2 <- glmnet(gss_train_x, gss_train_y, alpha = 0.75)
ridge    <- glmnet(gss_train_x, gss_train_y, alpha = 0.0)

fold_id <- sample(1:10, size = length(gss_train_y), replace = TRUE)

tuning_grid <- tibble::tibble(
  alpha      = seq(0, 1, by = .1),
  mse_min    = NA,
  mse_1se    = NA,
  lambda_min = NA,
  lambda_1se = NA
)

for(i in seq_along(tuning_grid$alpha)) {
  # fit CV model for each alpha value
  fit <- cv.glmnet(gss_train_x,
                   gss_train_y,
                   alpha = tuning_grid$alpha[i],
                   foldid = fold_id)

  # extract MSE and lambda values
```

```r
  tuning_grid$mse_min[i]    <- fit$cvm[fit$lambda == fit$lambda.min]
  tuning_grid$mse_1se[i]    <- fit$cvm[fit$lambda == fit$lambda.1se]
  tuning_grid$lambda_min[i] <- fit$lambda.min
  tuning_grid$lambda_1se[i] <- fit$lambda.1se
}



##now that we have the model, lets generate predictions so we can calculate error rate and AUC
best_elastic_mod <- cv.glmnet(gss_train_x,
                   gss_train_y,
                   alpha = 0.3,
                   foldid = fold_id)

best_elastic_mod_bin <- cv.glmnet(gss_train_x,
                   gss_train_y,
                   alpha = 0.3,
                   foldid = fold_id,
                   family = "binomial")


elastic_error <- mean(best_elastic_mod$cvm)

best_elastic_mod_auc <- cv.glmnet(gss_train_x,
                   gss_train_y,
                   alpha = 0.3,
                   family = "binomial",
                   type.measure = "auc",
                   foldid = fold_id)


elastic_auc <- mean(best_elastic_mod_auc$cvm)
```

Decision tree (CART)

```r
# generate 10-fold CV trees
gss_tree_cv <- gss_train_logit %>%
  vfold_cv(v = 10) %>%
  mutate(tree = map(splits, ~ tree(colrac ~ ., data = analysis(.x),
                                   control = tree.control(nobs = nrow(gss_train_logit),
                                                          mindev = .001))))

# calculate each possible prune result for each fold
gss_cv_tree_prune <- expand.grid(gss_tree_cv$id, 2:30) %>%
  as_tibble() %>%
  mutate(Var2 = as.numeric(Var2)) %>%
  rename(id = Var1,
         k = Var2) %>%
  left_join(gss_tree_cv) %>%
  mutate(prune = map2(tree, k, ~ prune.tree(.x, best = .y)),
         estimate = map2(prune, splits, ~ predict(.x, newdata = assessment(.y))[,2]),
         truth = map(splits, ~ assessment(.x)$colrac)) %>%
  unnest(estimate, truth) %>%
  group_by(k) %>%
  accuracy(truth = truth, estimate = factor(round(estimate), levels = 0:1, labels = c("0", "1")))

#find auc
tree_auc <- expand.grid(gss_tree_cv$id, 2:30) %>%
  as_tibble() %>%
  mutate(Var2 = as.numeric(Var2)) %>%
  rename(id = Var1,
         k = Var2) %>%
  left_join(gss_tree_cv) %>%
  mutate(prune = map2(tree, k, ~ prune.tree(.x, best = .y)),
         estimate = map2(prune, splits, ~ predict(.x, newdata = assessment(.y))[,2]),
         truth = map(splits, ~ assessment(.x)$colrac)) %>%
  unnest(estimate, truth) %>%
  mutate(estimate = round(estimate)) %>%
  group_by(k) %>%
  roc(response = truth, predictor = estimate)
```

Bagging

```r
gss_bag <- gss_train %>%
  mutate(colrac = make.names(as.character(if_else(colrac == 1, TRUE, FALSE))))


gss_bag_cv <- train(colrac ~ .,
                            data = gss_bag,
                            method = "treebag",
                            metric = "ROC",
                            trControl = ctrl)



bag_error2 <-  1- ((gss_bag_cv$results$Sens[which.max(gss_bag_cv$results$ROC)]*sum(gss_train$colrac)) +

   (gss_bag_cv$results$Spec[which.max(gss_bag_cv$results$ROC)]*(nrow(gss_train)
                                  - sum(gss_train$colrac)))) / nrow(gss_train)
```

Random Forest

```r
testgrid2 <- expand.grid(
  mtry = seq(14, 24, by = 3),
  min.node.size = seq(3, 9, by = 3),
  splitrule = "gini")


rf_cv <- train(colrac ~ .,
                       data = gss_bag,
                       method = 'ranger',
                       metric = 'Accuracy',
                       tuneGrid = testgrid2,
                       trControl = ctrl)


#ROC
rf_roc <- rf_cv$results$ROC[which.max(rf_cv$results$ROC)]


#new error


rf_error2 <- 1- ((rf_cv$results$Sens[which.max(rf_cv$results$ROC)]*sum(gss_train$colrac)) +

   (rf_cv$results$Spec[which.max(rf_cv$results$ROC)]*(nrow(gss_train)
                          - sum(gss_train$colrac)))) / nrow(gss_train)
```

Boosting

```
boost_grid <- expand.grid(
 shrinkage = seq(0.0001, 0.04, by = 0.01),
 n.trees = seq(50, 150, by = 50),
 interaction.depth = 1:3,
 n.minobsinnode = 10
)

gss_boost_cv <- train(colrac ~ .,
                            data = gss_bag,
                            method = "gbm",
                            metric = "ROC",
                            trControl = ctrl,
                      tuneGrid = boost_grid)


boost_roc <- gss_boost_cv$results$ROC[which.max(gss_boost_cv$results$ROC)]


#new error
boost_error2 <- 1- ((gss_boost_cv$results$Sens[which.max(gss_boost_cv$results$ROC)]*sum(gss_train$colrac

   (gss_boost_cv$results$Spec[which.max(gss_boost_cv$results$ROC)]*(nrow(gss_train)
                                      - sum(gss_train$colrac)))) / nrow(gss_train)
```

3. Evaluate the Models

Model performances:

- Cross-validated error rate
    - Logistic regression: 0.2059248
    - Naive Bayes: 0.2688408
    - Elastic net regression: 0.1595012

  - Decision tree (CART): 0.2276423
  - Bagging: 0.2226739
  - Random forest: 0.1991272
  - Boosting: 0.2095093
- ROC/AUC
  - Logistic regression: 0.7983075
  - Naive Bayes: 0.8141113
  - Elastic net regression: 0.869265
  - Decision tree (CART): 0.758432
  - Bagging: 0.8649713
  - Random forest: 0.89236
  - Boosting: 0.878489

4. Which is the best model? Defend your choice.

Considering the classification error rates and the AUC results, I believe the best model is the one produced by the random forest method. This model exhibits the second lowest classification error rate at 19.91%, second only to elastic net at 15.95% error. However, its AUC performed the best of all methods at .892, considerably better than elastic net at .869. Considering that it was a high performer in both categories, I argue that the random forest method is the best modeling procedure.

5. Evaluate the best model

```r
#Generate Preds
test_preds <- predict(rf_cv, as.matrix(gss_test_x))

test_preds <- as.numeric(test_preds) - 1
#Classification error rate
test_error <- sum(abs(test_preds-gss_test_y))/493

test_auc <- roc(response = gss_test_y, predictor = test_preds)
```

The test classification error rate is 0.2028398 and the test AUC is 0.7892751. We can compare this to the training error rate of 0.1991272 and training AUC of 0.89236. From these results, we can see that the model generalized reasonably well – the test error rates are considerably close, with only an approximate 0.4% drop in accuracy between the training and test error rates. The AUC did not generalize as well, with a drop of about 0.1. This would indicate that while the error rate remained nearly the same, the ratio of true positives to false positives decreased. In review, while I am happy with the well-generalized error rate, it is possible that other methods may have generalized better in terms of AUC.

Extra Credit

```r
#Plot Ice curve
library(ICEbox)
```

```r
#Tolerance
gss_test_ice <- gss_test
X <- as.data.frame(gss_test_ice)

gss.ice <- ice(object = rf_cv, X = X, predictor = "tolerance", logodds = FALSE,
predictfcn = function(object, newdata){
predict(object, newdata, type = "prob")[, 2]
}
)
```

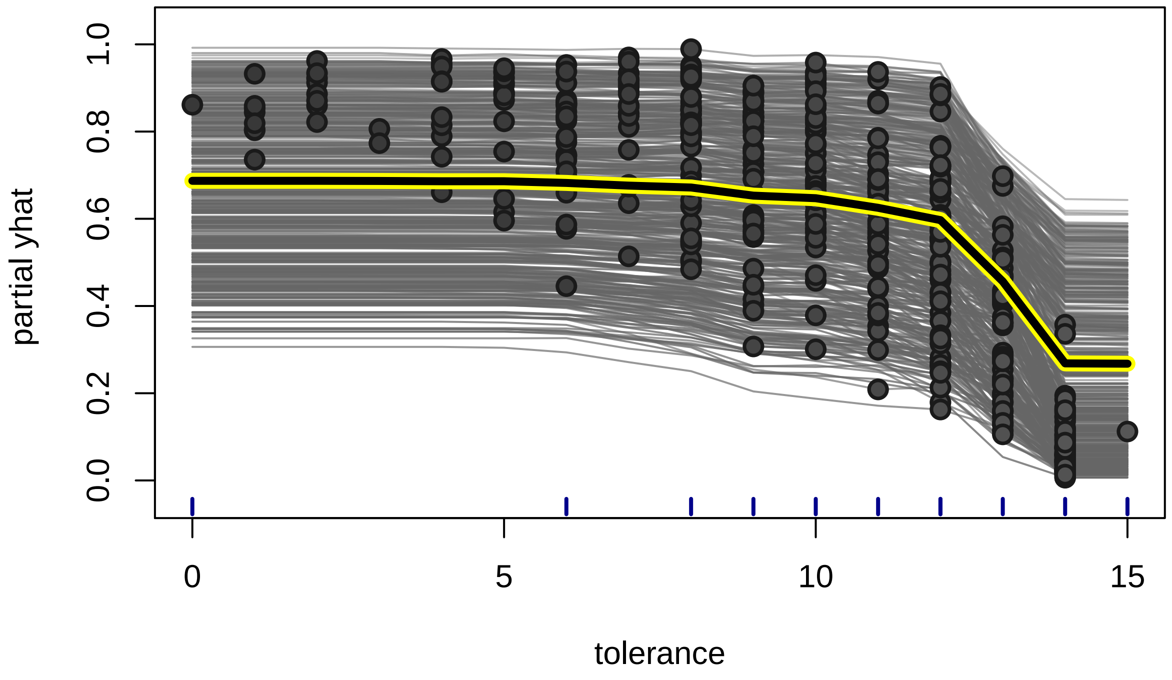
```
## ................
## y not passed, so range_y is range of ice curves and sd_y is sd of predictions on real observations
```

```r
#Age
gss.ice.age <- ice(object = rf_cv, X = X, predictor = "age", logodds = FALSE,
predictfcn = function(object, newdata){
predict(object, newdata, type = "prob")[, 2]
}
)
```

```
## ...................................................................
## y not passed, so range_y is range of ice curves and sd_y is sd of predictions on real observations
```

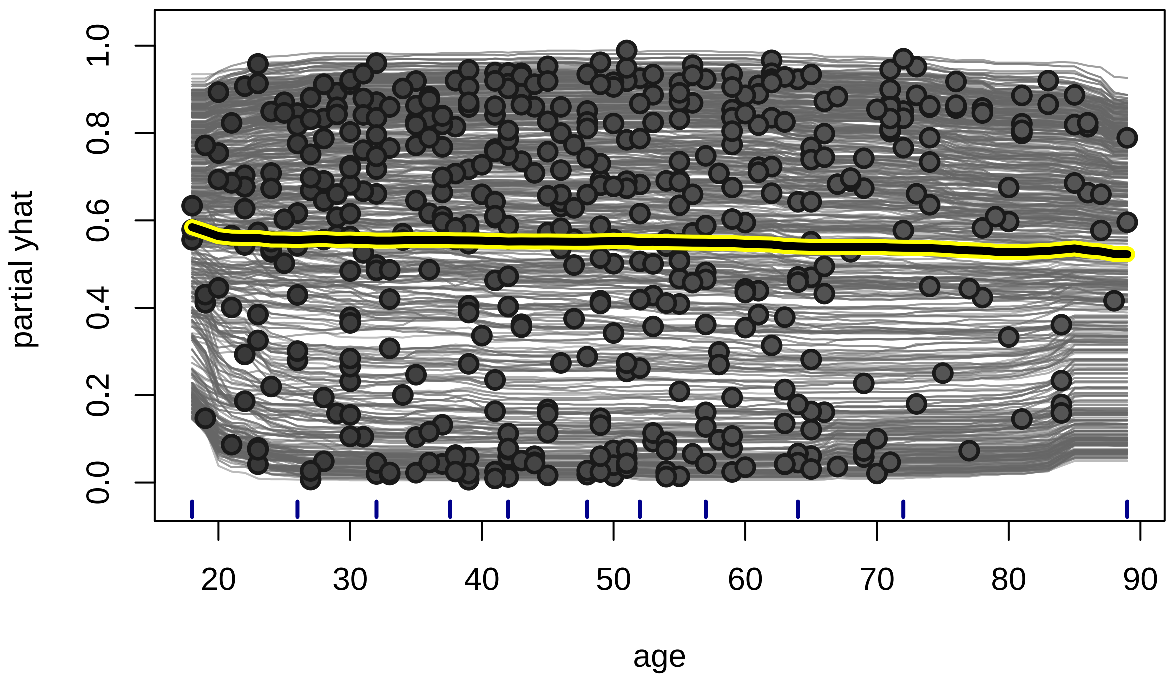
ICE Curve of Tolerance

```r
plot(gss.ice)
```



ICE Curve of Age

```r
plot(gss.ice.age)
```



The ICE curves of tolerance and age show us that tolerance levels are a strong predictor of an individual's answer to the question: "Should a person who believes that Blacks are genetically inferior be allowed to teach in a college or university?" The average probability of a Yes response to this question hovers at appproximately 0.7 for tolerance levels below 7 (out of 15) – it then begins to drop, reaching a minimum probability of just over 0.2 for the most tolerant respondents. For the other ICE curve, we can infer that age is not a strong predictor of an individual's response to the same question since the predicted probability of a Yes response remains between 0.5 and 0.6 across then entire age spectrum. While there might be a slight inverse relationship between age and response, tolerance levels are a much more informative predictor variable.