Homework 3: Trees

MACS 30100: Perspectives on Computational Modeling
University of Chicago

Overview

For each of the following prompts, produce responses *with* code in-line. While you are encouraged to stage and draft your problem set solutions using any files, code, and data you'd like within the private repo for the assignment, *only the final, rendered PDF with responses and code in-line will be graded.*

A Conceptual Problem

1. (15 points) Of the Gini index, classification error, and cross-entropy in simple classification settings with two classes, which would be best to use when *growing* a decision tree? Which would be best to use when *pruning* a decision tree? Why?

Classification error rate would be better to use in pruning the tree and can lead to overfitting if used when growing the tree as it does not take node purity into account, which might lead to low performance when applied on new data. Gini and cross entropy both are measures of node purity and they are more sensitive to changes in node probabilities than the misclassification rate. When growing the tree, two different types of split at same depth can result with the same misclassification error rate but different Gini and cross-entropy scores. In this case selecting the purest split would be the better choice, making Gini and cross-entropy more favorable for growing the tree. When pruning the tree, misclassification error rate is preferred if the goal is to make prediction.

An Applied Problem

For the applied portion, your task is to predict attitudes towards racist college professors using the General Social Survey (GSS) survey data. Each respondent was asked *Should a person who believes that Blacks are genetically inferior be allowed to teach in a college or university?* Given the controversy over Richard J. Herrnstein and Charles Murray's *The Bell Curve* and the ostracization of Nobel laureate James Watson over his controversial views on race and intelligence, this applied task will provide additional insight in the public debate over this issue.

To address this problem, use the `gss_*.csv` data sets, which contain a selection of features from the 2012 GSS. The outcome of interest is `colrac`, which is a binary feature coded as either `ALLOWED` or `NOT ALLOWED`, where 1 means the racist professor *should* be allowed to teach, and 0 means the racist professor *should not* be allowed to teach. Full documentation can be found here. I preprocessed the data for you to ease the model-fitting process:

- Missing values have been imputed
- Categorical features with low-frequency classes collapsed into an "other" category
- Nominal features with more than two classes have been converted to dummy features
- Remaining categorical features have been converted to integer values

Your task is to construct a series of models to accurately predict an individual's attitude towards permitting professors who view Blacks to be racially inferior to teach in a college classroom. The learning objectives are:

- Implement a battery of tree-based learners
- Tune hyperparameters
- Substantively interpret models

2. (35 points) Fit the following four tree-based models predicting `colrac` using the training set (`gss_train.csv`) with 10-fold CV. Remember to tune the relevant hyperparameters for each model as necessary. Only use the tuned model with the best performance for the remaining exercises. **Be sure to leave sufficient *time* for hyperparameter tuning, as grid searches can be quite computationally taxing and take a while.**

    - Decision tree (the rpart algorithm)
    - Bagging
    - Random forest
    - Gradient boosting

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
```

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

```r
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 0.1.2 --
```

```
## v broom     0.7.4      v recipes   0.1.15
## v dials     0.0.9      v rsample   0.0.8
## v infer     0.5.4      v tune      0.1.2
## v modeldata 0.1.0      v workflows 0.2.1
## v parsnip   0.1.4      v yardstick 0.0.7
```

```
## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
```

```r
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

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

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(patchwork)
library(rcfss)
```

```
##
## Attaching package: 'rcfss'
```

```
## The following object is masked from 'package:infer':
##
##     gss
```

```r
library(rpart)
```

```
##
## Attaching package: 'rpart'
```

```
## The following object is masked from 'package:dials':
##
##     prune
```

```r
library(rpart.plot)
library(ranger)
```

```
##
## Attaching package: 'ranger'
```

```
## The following object is masked from 'package:randomForest':
##
##     importance
```

```r
library(gbm)
```

```
## Loaded gbm 2.1.8
```

Read data

```r
gss_test <- read_csv('data/gss_test.csv')
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double()
## )
## i Use `spec()` for the full column specifications.
```

```r
gss <- read_csv('data/gss_train.csv')
```

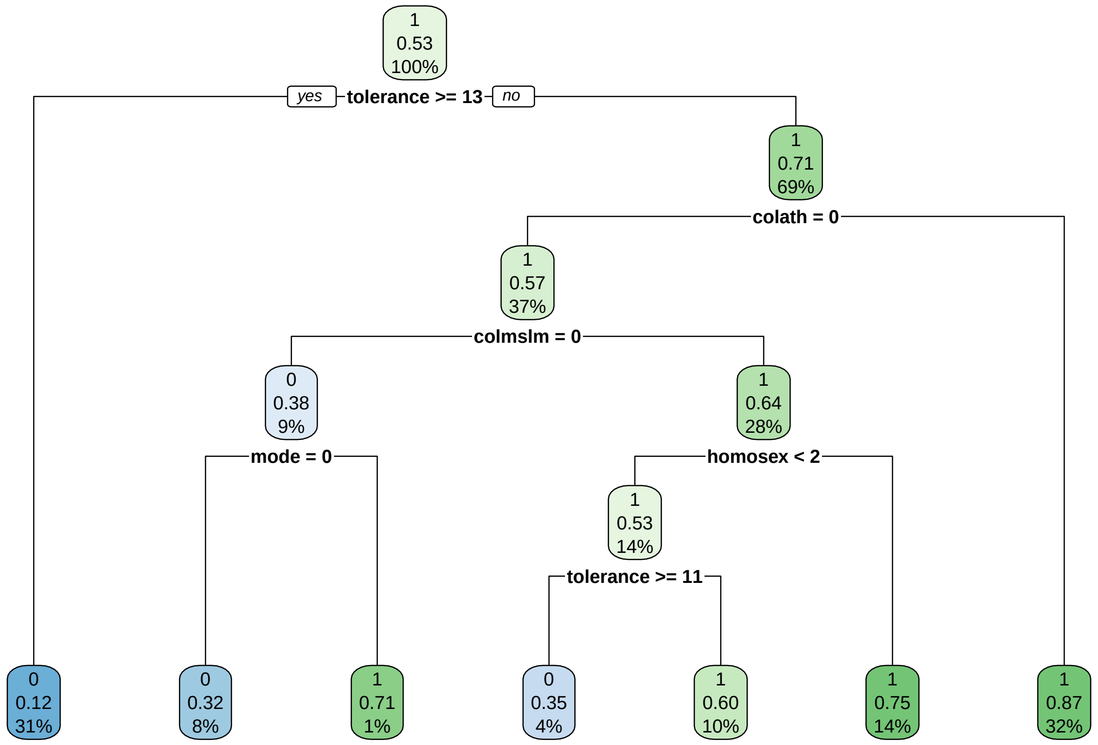
```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double()
## )
## i Use `spec()` for the full column specifications.
```

Decision Tree (rpart algorithm)

```r
hyper_grid <- expand.grid(
  cp = seq(0.001, 0.1, by = 0.01),
  minsplit = seq(5, 20, by = 5),
  minbucket = seq(3, 10, by = 1),
  error_rate = 0)

for(i in 1:nrow(hyper_grid)) {
  # train
  model <- rpart(
    formula = colrac ~.,
    data = gss,
    method = "class",
    control = list(cp = hyper_grid$cp[i],
                   minsplit = hyper_grid$minsplit[i],
                  hyper_grid$minbucket[i]))
  cptable <- as_tibble(model$cptable)
  error <- cptable[length(cptable$CP),]$xerror
  hyper_grid$error_rate[i] <- error
}
```

```r
opt_params <- filter(.data = hyper_grid, hyper_grid$error_rate == min(hyper_grid$error_rate))
opt_tree <- rpart(colrac ~ ., data = gss, method = 'class', control = rpart.control(xval = 10,
                                                                                   cp = opt_params$cp
                                                                                   minbucket = opt_pa
                                                                                   minsplit = opt_par
rpart.plot(opt_tree)
```



```
#RANDOM FOREST
hyper_grid <- expand.grid(
  mtry = seq(20, 30, by = 2),
  node_size = seq(3, 9, by = 2),
  sample_size = c(.55, .632, .70, .80),
  OOB_RMSE = 0
)

for(i in 1:nrow(hyper_grid)) {
  model <- ranger(
    formula = colrac ~.,
    data = gss,
    num.trees = 500,
    mtry = hyper_grid$mtry[i],
    min.node.size = hyper_grid$node_size[i],
    sample.fraction = hyper_grid$sample_size[i],
    classification = T)

  hyper_grid$OOB_RMSE[i] <- sqrt(model$prediction.error)
}
```

```
forest_params <- head(arrange(hyper_grid, OOB_RMSE), 1)


opt_forest <- ranger(
  formula = colrac ~.,
```

```r
  data = gss,
  num.trees = 500,
  mtry = forest_params$mtry,
  min.node.size = forest_params$node_size,
  sample.fraction = forest_params$sample_size,
  classification = T)

#BAGGING
hyper_grid <- expand.grid(
  node_size = seq(3, 9, by = 2),
  sample_size = c(.55, .632, .70, .80),
  OOB_RMSE = 0
)

for(i in 1:nrow(hyper_grid)) {
  model <- ranger(
    formula = colrac ~.,
    data = gss,
    num.trees = 500,
    mtry = length(select(gss, -colrac)),
    min.node.size = hyper_grid$node_size[i],
    sample.fraction = hyper_grid$sample_size[i],
    classification = T)

  hyper_grid$OOB_RMSE[i] <- sqrt(model$prediction.error)
}
```

```r
bagging_params <- head(arrange(hyper_grid, OOB_RMSE), 1)


opt_bagging <- ranger(
  formula = colrac ~.,
  data = gss,
  num.trees = 500,
  mtry = bagging_params$mtry,
  min.node.size = bagging_params$node_size,
  sample.fraction = bagging_params$sample_size,
  classification = T)

hyper_grid_boosting <- expand.grid(
  lambda_range = c(0.3, 0.1, 0.05, 0.01, 0.005),
  RMSE = 0
)
for(lmi in 1:length(hyper_grid_boosting$lambda_range)){
  lm <- hyper_grid_boosting$lambda_range[lmi]

  boost <- gbm(
    formula = colrac ~ .,
    data = gss,
    distribution = "bernoulli",  # SSE loss function
    n.trees = 500,
    shrinkage = lm,
    interaction.depth = 3,
    n.minobsinnode = 10,
    cv.folds = 10)
  hyper_grid_boosting$RMSE[lmi] = sqrt(min(boost$cv.error))
}
```

```
optimal_lrate <- head(arrange(hyper_grid_boosting, RMSE), 1)$lambda_range

opt_boost <- gbm(
    formula = colrac ~ .,
    data = gss,
    distribution = "bernoulli",  # SSE loss function
    n.trees = 500,
    shrinkage = optimal_lrate,
    interaction.depth = 3,
    n.minobsinnode = 10,
    cv.folds = 10)
```

```
library(ROCR)
```

3. (20 points) Compare and present each model's (training) performance based on:

    - Cross-validated error rate
    - ROC/AUC

```
dt_pred <- prediction(predict(opt_tree, type = "prob")[, 2], gss$colrac)
dt_auc <- performance(dt_pred, measure = "auc")
dt_auc <- dt_auc@y.values[[1]]

rf_pred <- prediction(predictions(opt_forest), gss$colrac)
rf_auc <- performance(rf_pred, measure = "auc")
rf_auc <- rf_auc@y.values[[1]]

bag_pred <- prediction(predictions(opt_bagging), gss$colrac)
bag_auc <- performance(bag_pred, measure = "auc")
bag_auc <- bag_auc@y.values[[1]]

boost_pred <- prediction(predict.gbm(opt_boost, type = 'response'), gss$colrac)
```

```
## Using 255 trees...
```

```
boost_auc <- performance(boost_pred, measure = "auc")
boost_auc <- boost_auc@y.values[[1]]

tree_preds <- xpred.rpart(opt_tree, xval = 10)
class.pred <- table(tree_preds[, ncol(tree_preds)] - 1, gss$colrac)
tree_error <- 1-sum(diag(class.pred))/sum(class.pred)

yhat <- round(predict.gbm(opt_boost, type = 'response'))
```

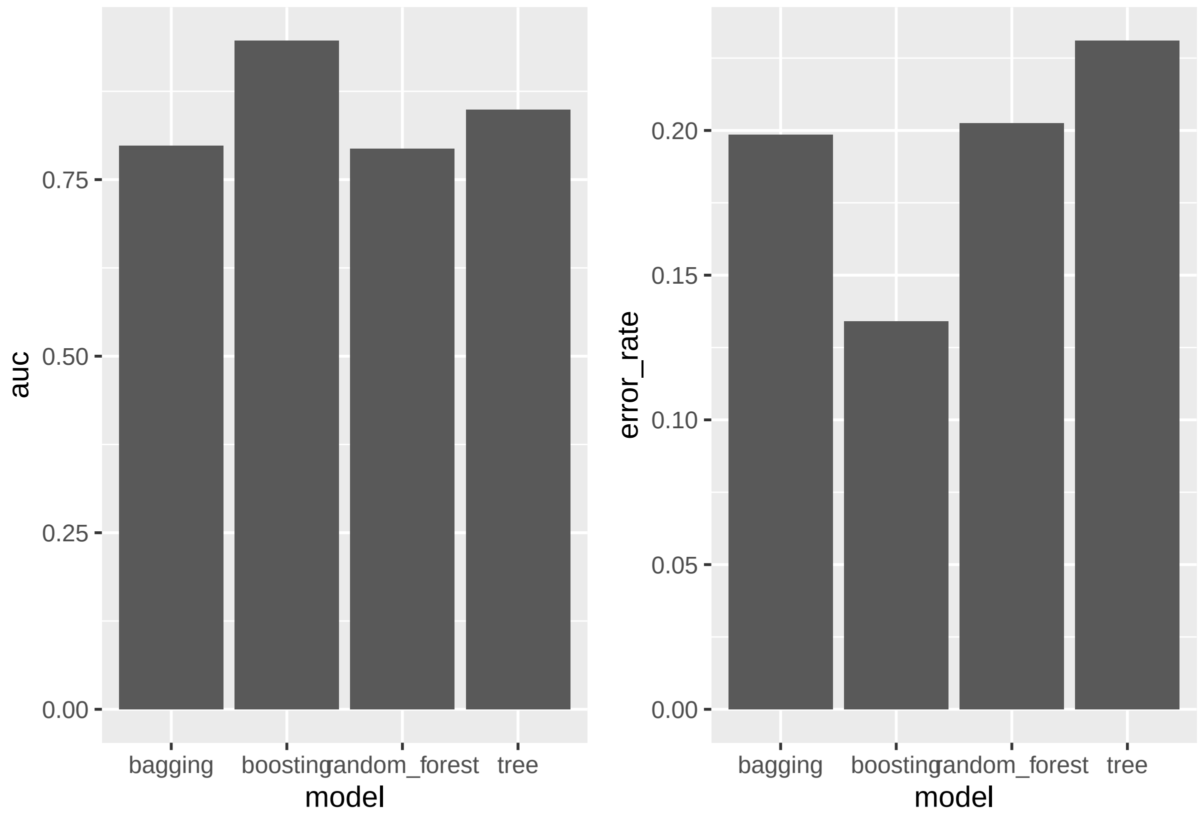
```
## Using 255 trees...
```

```r
boost_error <- mean((yhat-gss$colrac)^2)

errors <- tibble(boosting = boost_error,
                bagging = opt_bagging$prediction.error,
                random_forest = opt_forest$prediction.error,
                tree = tree_error)
errors <- pivot_longer(errors, cols = boosting:tree, names_to = 'model', values_to = 'error_rate')

aucs <- tibble(model = 'random_forest', auc = rf_auc) %>%
  add_row(model = 'tree', auc = dt_auc) %>%
  add_row(model = 'bagging', auc = bag_auc) %>%
  add_row(model = 'boosting', auc = boost_auc)

ggplot(data = aucs) +
  geom_col(aes(x = model, y = auc)) +
  ggplot(data = errors) +
  geom_col(aes(x = model, y = error_rate))
```



4. (15 points) Which is the best model? Defend your choice.

Random forest and bagging perform very similarly and decision tree is by far the worst performer among the four models. I would choose to go with the Boosting model because it has both the lowest error rate and highest AUC value. One issue with the boosting model could be overfitting as the number of trees increase (I used 500). But the difference in metrics compared to other models would justify the boosting model which has relatively higher variance. Every tree in the boosting model learns from the mistakes of the previous trees by taking the residuals into account, and the tuned hyperparameter lambda allowed us to select the optimal learning rate, which in turn lead to higher performance. One of the problems with boosting could be interpretation when compared to the decision tree model, but since our goal here is prediction, increased complexity is justified and we can sacrifice interpretability.

5. (15 points) Evaluate the performance of the best model selected in the previous question using the test set (`gss_test.csv`) by calculating and presenting the classification error rate and AUC of this model. Compared to the fit evaluated on the training set, does this "best" model generalize well? Why or why not? How do you know?

```
pred <- round(predict.gbm(opt_boost, gss_test, type = 'response'))
```

```
## Using 255 trees...
```

```
paste('classification error rate is:', mean((gss_test$colrac - pred)^2))
```

```
## [1] "classification error rate is: 0.202839756592292"
```

```
best_pred <- prediction(pred, gss_test$colrac)
best_auc <- performance(best_pred, measure = "auc")
best_auc <- best_auc@y.values[[1]]
paste('AUC of best model is:', best_auc)
```

```
## [1] "AUC of best model is: 0.79019364448858"
```

In general, the model generalized okay and did not lose much predictive power when confronted with reserved testing data. The AUC and error rates are expectedly worse than the ones we saw in the training split. This is because the boosted trees are learning from the mistakes revealed by the training data, and the newly introduced data often will not have identical properties with the training set. It can be argued that the difference between training and testing performance for some other models such as the random forest could be smaller, especially since RF shuffles the predictor variables in each iteration of the tree, avoiding overfitting. However, even when there is considerable difference between training and testing error rates for Boosting model, the resulting test error rates are still lower than some of the training error rates of the other models, which justifies our decision to go with the Boosting model.