Exercise 18

Marc Dotson

Return to the previous exercise.

1. Try and improve on the predictive fit by using a random forest.
2. After fitting a random forest, use cross-validation and hyperparameter tuning on the random forest. Finalize the best-fitting model and compute predictive fit. Have you improved on the best-fitting model?
3. Render the Quarto document into Word and upload to Canvas.

**Five points total, one point each for:**

* **Fitting one or more random forests on the same data as the previous exercise.**
* **Using cross-validation.**
* **Using hyperparameter tuning for the random forest.**
* **Discuss whether or not they’ve been able to improve predictive fit.**
* **Submitting a rendered Word document.**

## Data Prep and Feature Engineering

We are encoding segment as two categories and using the same predictors as before.

# Load packages.  
library(tidyverse)

── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
✔ dplyr 1.1.4 ✔ readr 2.1.4  
✔ forcats 1.0.0 ✔ stringr 1.5.1  
✔ ggplot2 3.4.4 ✔ tibble 3.2.1  
✔ lubridate 1.9.3 ✔ tidyr 1.3.0  
✔ purrr 1.0.2   
── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
✖ dplyr::filter() masks stats::filter()  
✖ dplyr::lag() masks stats::lag()  
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(tidymodels)

── Attaching packages ────────────────────────────────────── tidymodels 1.1.1 ──  
✔ broom 1.0.5 ✔ rsample 1.2.0  
✔ dials 1.2.0 ✔ tune 1.1.2  
✔ infer 1.0.5 ✔ workflows 1.1.3  
✔ modeldata 1.2.0 ✔ workflowsets 1.0.1  
✔ parsnip 1.1.1 ✔ yardstick 1.2.0  
✔ recipes 1.0.9   
── Conflicts ───────────────────────────────────────── tidymodels\_conflicts() ──  
✖ scales::discard() masks purrr::discard()  
✖ dplyr::filter() masks stats::filter()  
✖ recipes::fixed() masks stringr::fixed()  
✖ dplyr::lag() masks stats::lag()  
✖ yardstick::spec() masks readr::spec()  
✖ recipes::step() masks stats::step()  
• Learn how to get started at https://www.tidymodels.org/start/

# Set the randomization seed.  
set.seed(42)  
  
# Import data and wrangle S1 into segment.  
roomba\_survey <- read\_csv(here::here("Data", "roomba\_survey.csv")) |>   
 rename(segment = S1) |>   
 mutate(  
 segment = case\_when(  
 segment == 1 ~ "own or shopping",  
 segment == 3 ~ "own or shopping",  
 segment == 4 ~ "considering"  
 ),  
 segment = factor(segment)  
 )

Rows: 332 Columns: 128  
── Column specification ────────────────────────────────────────────────────────  
Delimiter: ","  
chr (9): sys\_LastQuestion, sys\_CBC\_CBC1\_design, sys\_CBC\_CBC1\_design\_info, ...  
dbl (119): sys\_RespNum, sys\_StartTime, sys\_EndTime, S1, S1A, S1B, S1C, S2, S...  
  
ℹ Use `spec()` to retrieve the full column specification for this data.  
ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

# Split data based on segment.  
roomba\_split <- initial\_split(roomba\_survey, prop = 0.75, strata = segment)  
  
# Feature engineering.  
roomba\_recipe <- training(roomba\_split) |>  
 recipe(  
 segment ~ CleaningAttitudes\_1 + CleaningAttitudes\_2 + CleaningAttitudes\_3 +   
 CleaningAttitudes\_4 + CleaningAttitudes\_5 + CleaningAttitudes\_6 +   
 CleaningAttitudes\_7 + CleaningAttitudes\_8 + CleaningAttitudes\_9 +   
 CleaningAttitudes\_10 + CleaningAttitudes\_11 +  
 D1Gender + D2HomeType + D3Neighborhood + D4MaritalStatus  
 ) |>  
 step\_dummy(all\_nominal(), -all\_outcomes())

## Random Forest

Let’s create a workflow for random then update it for subsequent models.

# Random forest.  
roomba\_rf <- rand\_forest() |>   
 set\_engine(engine = "randomForest") |>   
 set\_mode("classification")  
  
# Create a workflow for lr.  
roomba\_wf\_rf <- workflow() |>   
 add\_recipe(roomba\_recipe) |>   
 add\_model(roomba\_rf)  
  
# Fit the model.  
fit\_wf\_rf <- fit(roomba\_wf\_rf, data = training(roomba\_split))  
  
# Compute model accuracy.  
fit\_wf\_rf |>   
 predict(new\_data = testing(roomba\_split)) |>  
 bind\_cols(testing(roomba\_split)) |>  
 accuracy(truth = segment, estimate = .pred\_class)

# A tibble: 1 × 3  
 .metric .estimator .estimate  
 <chr> <chr> <dbl>  
1 accuracy binary 0.726

The logistic regression and decision tree previously had a tied predictive fit accuracy of 0.711. Without any hyperparameter tuning, the random forest already has an improved predictive fit accuracy of 0.726.

## Hyperparameter Tuning

Let’s see if we can improve the prediction by tuning the random forest hyperparameters.

# Use v-fold cross-validation based on segment.  
roomba\_cv <- vfold\_cv(training(roomba\_split), v = 10, strata = segment)  
  
# Random forest with hyperparameters to tune.  
roomba\_rf\_tune <- rand\_forest(mtry = tune(), trees = tune(), min\_n = tune()) |>   
 set\_engine(engine = "randomForest") |>   
 set\_mode("classification")  
  
# Update the workflow.  
roomba\_wf\_rf <- roomba\_wf\_rf |>   
 update\_model(roomba\_rf\_tune)  
  
# Tune the hyperparameters by using the cross-validation.  
fit\_wf\_rf <- roomba\_wf\_rf |>   
 tune\_grid(resamples = roomba\_cv)

i Creating pre-processing data to finalize unknown parameter: mtry

# Select the best fitting model.  
rf\_best\_fit <- fit\_wf\_rf |>   
 select\_best(metric = "accuracy")  
  
# Finalize the workflow.  
roomba\_wf\_rf <- roomba\_wf\_rf |>   
 finalize\_workflow(rf\_best\_fit)  
  
# Fit the tuned workflow to the whole dataset.  
fit\_wf\_rf <- fit(roomba\_wf\_rf, data = training(roomba\_split))  
  
# Compute model accuracy.  
fit\_wf\_rf |>   
 predict(new\_data = testing(roomba\_split)) |>  
 bind\_cols(testing(roomba\_split)) |>  
 accuracy(truth = segment, estimate = .pred\_class)

# A tibble: 1 × 3  
 .metric .estimator .estimate  
 <chr> <chr> <dbl>  
1 accuracy binary 0.690

By tuning the hyperparameters, we only get a predictive fit accuracy of 0.690. It looks like the defaults for the hyperparameters are sufficient for the best predictive fit.