Exercise 20

Marc Dotson

One last time, return to the data from the previous three exercises.

1. Build a logistic regression, decision tree, random forest, boosted tree, and a neural network as well as a stacking ensemble with all five of the model types as candidates. Use the same random training and testing split.
2. Create a table comparing all six models. Which is the best-fitting model? Why do you think its the best-fitting model for this specific problem?
3. Render the Quarto document into Word and upload to Canvas.

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

* **Fitting a logistic regression, tuned decision tree, tuned random forest, tuned boosted tree, and tune neural network using the same resampled training data.**
* **Fitting a stacked ensemble using all of the previous models as candidates.**
* **Creating a table to compare all of the models on the same testing data.**
* **An explanation as to why they think the best-fitting model is best for this specific problem.**
* **Submitting a rendered Word document.**

## Data Prep and Feature Engineering

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

# Load packages and functions.  
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 1.2.0   
✔ infer 1.0.5 ✔ workflows 1.1.4   
✔ modeldata 1.2.0 ✔ workflowsets 1.0.1   
✔ parsnip 1.2.1 ✔ yardstick 1.3.1   
✔ recipes 1.0.10   
── 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()  
• Use suppressPackageStartupMessages() to eliminate package startup messages

library(stacks)  
  
fit\_accuracy <- function(fit, testing\_data, truth) {  
 fit |>   
 predict(new\_data = testing\_data) |>  
 bind\_cols(testing\_data) |>  
 accuracy(truth = {{truth}}, estimate = .pred\_class)  
}  
  
# Set a seed.  
set.seed(97)  
  
# 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)  
  
# Use v-fold cross-validation based on segment.  
roomba\_cv <- vfold\_cv(training(roomba\_split), v = 10, 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\_predictors()) |>   
 step\_normalize(all\_predictors())

## Fit Models

Let’s fit all of the competing models on the same resampled training data.

# Logistic regression.  
roomba\_lr <- logistic\_reg() |>   
 set\_engine(engine = "glm")  
  
roomba\_wf\_lr <- workflow() |>   
 add\_recipe(roomba\_recipe) |>   
 add\_model(roomba\_lr)  
  
resample\_lr <- roomba\_wf\_lr |>   
 fit\_resamples(  
 resamples = roomba\_cv,  
 control = control\_stack\_resamples()  
 )  
  
fit\_lr <- roomba\_wf\_lr |>   
 fit(data = training(roomba\_split))  
  
# Decision tree.  
roomba\_dt <- decision\_tree(tree\_depth = tune(), min\_n = tune()) |>   
 set\_engine(engine = "rpart") |>   
 set\_mode("classification")  
  
roomba\_wf\_dt <- roomba\_wf\_lr |>   
 update\_model(roomba\_dt)  
  
tune\_dt <- roomba\_wf\_dt |>   
 tune\_grid(  
 resamples = roomba\_cv,  
 control = control\_stack\_grid()  
 )  
  
fit\_dt <- roomba\_wf\_dt |>   
 finalize\_workflow(select\_best(tune\_dt, metric = "accuracy")) |>   
 fit(data = training(roomba\_split))  
  
# Random forest.  
roomba\_rf <- rand\_forest(mtry = tune(), trees = tune(), min\_n = tune()) |>   
 set\_engine(engine = "randomForest") |>   
 set\_mode("classification")  
  
roomba\_wf\_rf <- roomba\_wf\_dt |>   
 update\_model(roomba\_rf)  
  
tune\_rf <- roomba\_wf\_rf |>   
 tune\_grid(  
 resamples = roomba\_cv,  
 control = control\_stack\_grid()  
 )

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

fit\_rf <- roomba\_wf\_rf |>   
 finalize\_workflow(select\_best(tune\_rf, metric = "accuracy")) |>   
 fit(data = training(roomba\_split))  
  
# Boosted tree.  
roomba\_bt <- boost\_tree(  
 tree\_depth = tune(), trees = tune(), learn\_rate = tune(),   
 mtry = tune(), min\_n = tune(), sample\_size = tune()  
) |>  
 set\_engine("xgboost") |>   
 set\_mode("classification")  
  
roomba\_wf\_bt <- roomba\_wf\_dt |>   
 update\_model(roomba\_bt)  
  
tune\_bt <- roomba\_wf\_bt |>   
 tune\_grid(  
 resamples = roomba\_cv,  
 control = control\_stack\_grid()  
 )

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

fit\_bt <- roomba\_wf\_bt |>   
 finalize\_workflow(select\_best(tune\_bt, metric = "accuracy")) |>  
 fit(data = training(roomba\_split))  
  
# Neural network.  
roomba\_nn <- mlp(hidden\_units = tune(), epochs = tune(), penalty = tune()) |>   
 set\_engine(engine = "nnet") |>   
 set\_mode("classification")  
  
roomba\_wf\_nn <- roomba\_wf\_dt |>  
 update\_model(roomba\_nn)  
  
tune\_nn <- roomba\_wf\_nn |>   
 tune\_grid(  
 resamples = roomba\_cv,  
 control = control\_stack\_grid()  
 )  
  
fit\_nn <- roomba\_wf\_nn |>   
 finalize\_workflow(select\_best(tune\_nn, metric = "accuracy")) |>   
 fit(data = training(roomba\_split))  
  
# Stack of candidate ensemble members.  
roomba\_stack <- stacks() |>   
 add\_candidates(resample\_lr) |>   
 add\_candidates(tune\_dt) |>   
 add\_candidates(tune\_rf) |>   
 add\_candidates(tune\_bt) |>  
 add\_candidates(tune\_nn)

Warning: Predictions from 2 candidates were identical to those from existing candidates  
and were removed from the data stack.

Warning: Predictions from 3 candidates were identical to those from existing candidates  
and were removed from the data stack.

# Aggregate predictions.  
fit\_stack <- roomba\_stack |>   
 blend\_predictions() |>   
 fit\_members()

## Compare Predictive Fit

Let’s create a table to compare predictive fit using a function.

# Create a table of predictive fit.  
bind\_cols(  
 model = c("lr", "dt", "rf", "bt", "nn", "stack"),  
 bind\_rows(  
 fit\_accuracy(fit\_lr, testing(roomba\_split), segment),  
 fit\_accuracy(fit\_dt, testing(roomba\_split), segment),  
 fit\_accuracy(fit\_rf, testing(roomba\_split), segment),  
 fit\_accuracy(fit\_bt, testing(roomba\_split), segment),  
 fit\_accuracy(fit\_nn, testing(roomba\_split), segment),  
 fit\_accuracy(fit\_stack, testing(roomba\_split), segment)  
 )  
)

# A tibble: 6 × 4  
 model .metric .estimator .estimate  
 <chr> <chr> <chr> <dbl>  
1 lr accuracy binary 0.583  
2 dt accuracy binary 0.643  
3 rf accuracy binary 0.679  
4 bt accuracy binary 0.690  
5 nn accuracy binary 0.655  
6 stack accuracy binary 0.655

The random forest and the boosted tree tie for best predictive accuracy. These are hard to beat! It allows the flexibility of simple models with the benefit of an ensemble. It’s interesting that both approaches arrive with the same predictive fit. Clearly the added complexity of the neural net, and the ensemble of ensembles that is stacking, wasn’t needed for this specific application.