Pstat131 Hw6

Noe Arambula

2022-05-25

Contents

Loading Packages

```
library(yardstick)
library(tidyverse)
library(tidymodels)
library(ISLR)
library(rpart.plot)
library(vip)
library(janitor)
library(randomForest)
library(xgboost)
library(corrr)
library(corrplot)
library(ranger)

tidymodels_prefer()
```

Set Data and Seed

```
pokemon <- read_csv("Pokemon.csv")
```

```
## Rows: 800 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Name, Type 1, Type 2
## dbl (9): #, Total, HP, Attack, Defense, Sp. Atk, Sp. Def, Speed, Generation
## lgl (1): Legendary
##
## 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.
```

```
set.seed(777)
```

Question 1

Read in the data and set things up as in Homework 5:

- Use `clean_names()`
- Filter out the rarer Pokémon types
- Convert `type_1` and `legendary` to factors

Do an initial split of the data; you can choose the percentage for splitting. Stratify on the outcome variable.

Fold the training set using v-fold cross-validation, with `v = 5`. Stratify on the outcome variable.

Set up a recipe to predict `type_1` with `legendary`, `generation`, `sp_atk`, `attack`, `speed`, `defense`, `hp`, and `sp_def`:

- Dummy-code `legendary` and `generation`;
- Center and scale all predictors.

Setup Like Hw 5

```
clean_pokemon <- clean_names(pokemon)

# Filter entire dataset
filtered_pokemon <- filter(clean_pokemon, type_1 == "Bug" | type_1 == "Fire"
                          | type_1 == "Grass" | type_1== "Normal" |
                          type_1 == "Water" | type_1 == "Psychic")

# Convert type_1 and legendary to factors
filtered_pokemon$type_1 <- factor(filtered_pokemon$type_1)
filtered_pokemon$legendary <- factor(filtered_pokemon$legendary)
```

Initial Split, Vfold, and Recipe Setup

```
set.seed(777)

# Initial split
pokemon_split <- initial_split(filtered_pokemon, strata = type_1, prop = 0.8)

pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)

# Folding Training Data
set.seed(777)
pokemon_fold <- vfold_cv(pokemon_train, v = 5, strata = type_1)

# Recipe
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack
                         + speed + defense + hp + sp_def, data = pokemon_train) %>%
  step_dummy(all_nominal_predictors()) %>%   # creates dummy variables
  step_normalize(all_predictors())    # Centers and Scales all variables

pokemon_recipe
```

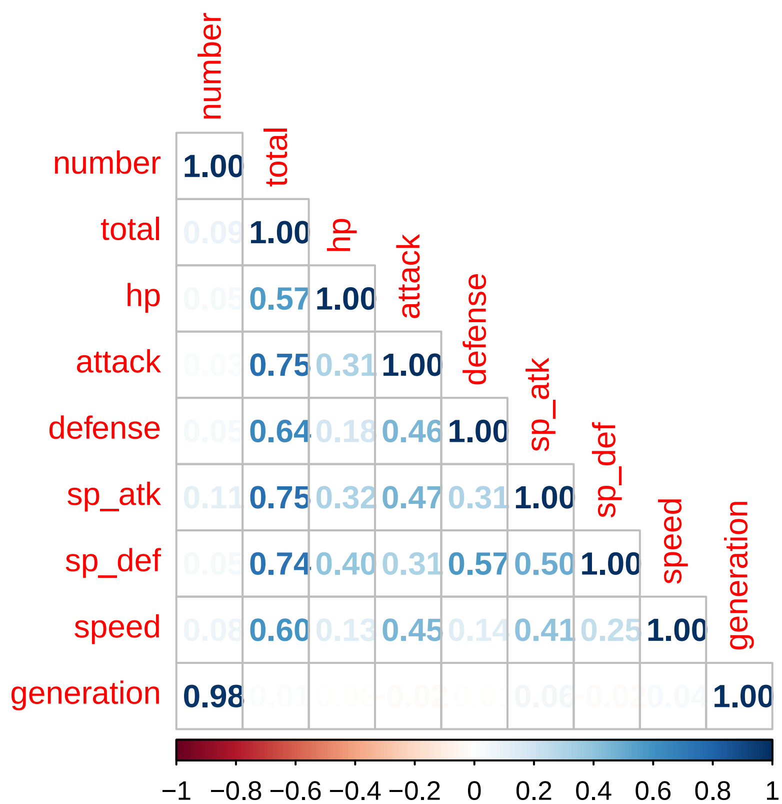
```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          8
##
## Operations:
##
## Dummy variables from all_nominal_predictors()
## Centering and scaling for all_predictors()
```

Question 2

Create a correlation matrix of the training set, using the `corrplot` package. *Note: You can choose how to handle the continuous variables for this plot; justify your decision(s).*

What relationships, if any, do you notice? Do these relationships make sense to you?

```
# note for a correlation matrix variable must be numeric
pokemon_train %>%
  select(where(is.numeric)) %>%
  cor() %>%
  corrplot(type = 'lower',
           method = 'number')
```



I chose to leave all continuous variables in since I feel it would be important to see all relationships. I see a strong relationship between generation and number which makes complete sense since pokemon that were created later are part of the next generation accordingly. There are also relationships between all of a Pokemon's stats and the total which also makes sense since the stats are what makeup the total number. Special defense and defense seem to be slightly correlated as well.

Question 3

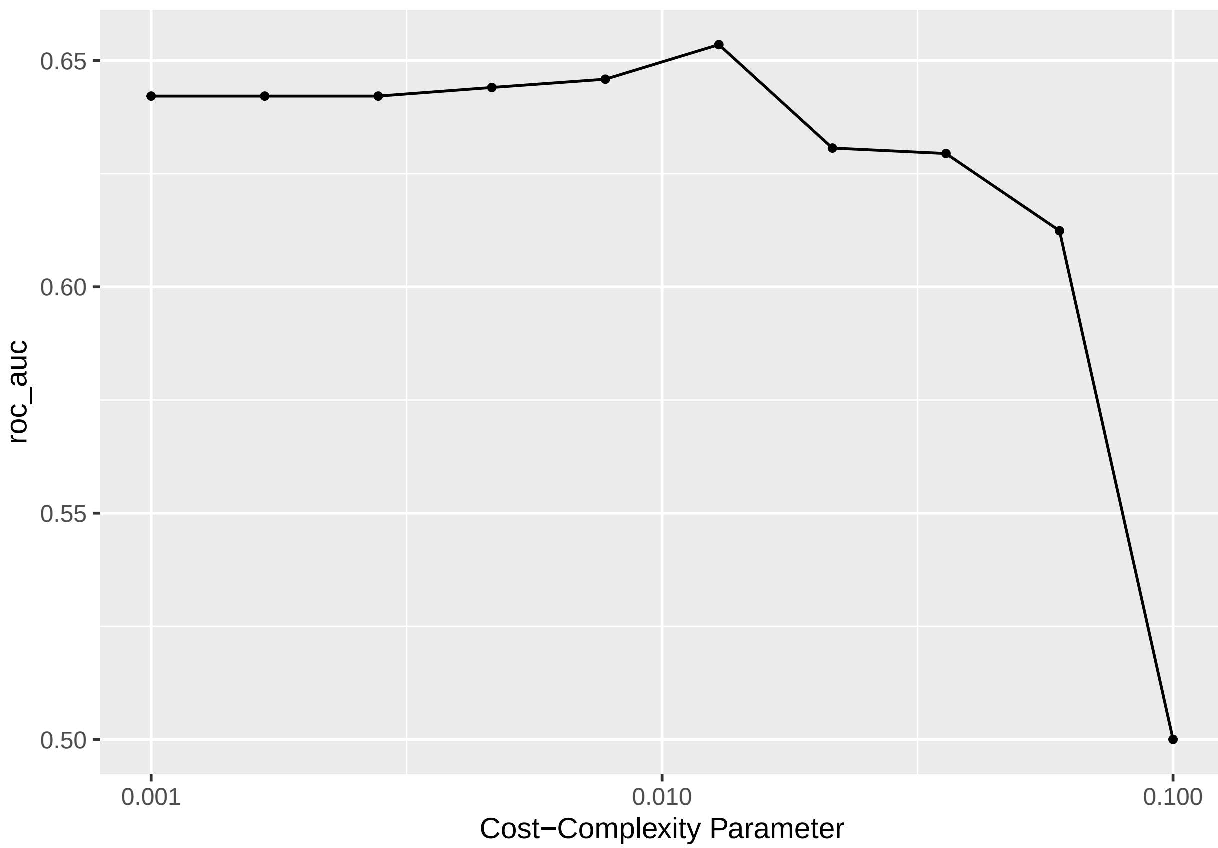
First, set up a decision tree model and workflow. Tune the `cost_complexity` hyperparameter. Use the same levels we used in Lab 7 – that is, `range = c(-3, -1)`. Specify that the metric we want to optimize is `roc_auc`.

Print an `autoplot()` of the results. What do you observe? Does a single decision tree perform better with a smaller or larger complexity penalty?

```r
# general decision tree specification
tree_spec <- decision_tree() %>%
  set_engine("rpart")

# classificaition decision tree engine/model
class_tree_spec <- tree_spec %>%
  set_mode("classification")

# Workflow tuning cost complexity
class_tree_wf <- workflow() %>%
  add_model(class_tree_spec %>% set_args(cost_complexity = tune())) %>%
  add_recipe(pokemon_recipe)

# setup grid
set.seed(777)

param_grid <- grid_regular(cost_complexity(range = c(-3, -1)), levels = 10)
```

```r
# will load in the model later instead of running code to save time
tune_res_tree <- tune_grid(
  class_tree_wf,
  resamples = pokemon_fold,
  grid = param_grid,
  metrics = metric_set(roc_auc)
)
```

```r
load(file = "tunedmodels.rda")
autoplot(tune_res_tree)
```



The lower the complexity parameter the higher and better roc_auc we get. It seems that the 6th model performs the best with cost complexity parameter about 0.02

Question 4

What is the `roc_auc` of your best-performing pruned decision tree on the folds? *Hint: Use `collect_metrics()` and `arrange()`.*

```
set.seed(777)
# this will also show the best model based on roc_auc but does not show the actual roc_auc number
best_model <- select_best(tune_res_tree, metric = "roc_auc")
best_model
```

```
## # A tibble: 1 x 2
##   cost_complexity .config
##             <dbl> <chr>
## 1          0.0129 Preprocessor1_Model06
```

```
# shows all models and organizes from worst to best roc_auc
collect_metrics(tune_res_tree) %>%
  arrange(mean)
```

```
## # A tibble: 10 x 7
```

```
##    cost_complexity .metric .estimator  mean     n std_err .config
##              <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         0.1     roc_auc hand_till  0.5       5  0      Preprocessor1_Model10
##  2         0.0599  roc_auc hand_till  0.612     5  0.0155 Preprocessor1_Model09
##  3         0.0359  roc_auc hand_till  0.629     5  0.0286 Preprocessor1_Model08
##  4         0.0215  roc_auc hand_till  0.631     5  0.0287 Preprocessor1_Model07
##  5         0.001   roc_auc hand_till  0.642     5  0.0123 Preprocessor1_Model01
##  6         0.00167 roc_auc hand_till  0.642     5  0.0123 Preprocessor1_Model02
##  7         0.00278 roc_auc hand_till  0.642     5  0.0123 Preprocessor1_Model03
##  8         0.00464 roc_auc hand_till  0.644     5  0.0197 Preprocessor1_Model04
##  9         0.00774 roc_auc hand_till  0.646     5  0.0190 Preprocessor1_Model05
## 10         0.0129  roc_auc hand_till  0.654     5  0.0213 Preprocessor1_Model06
```

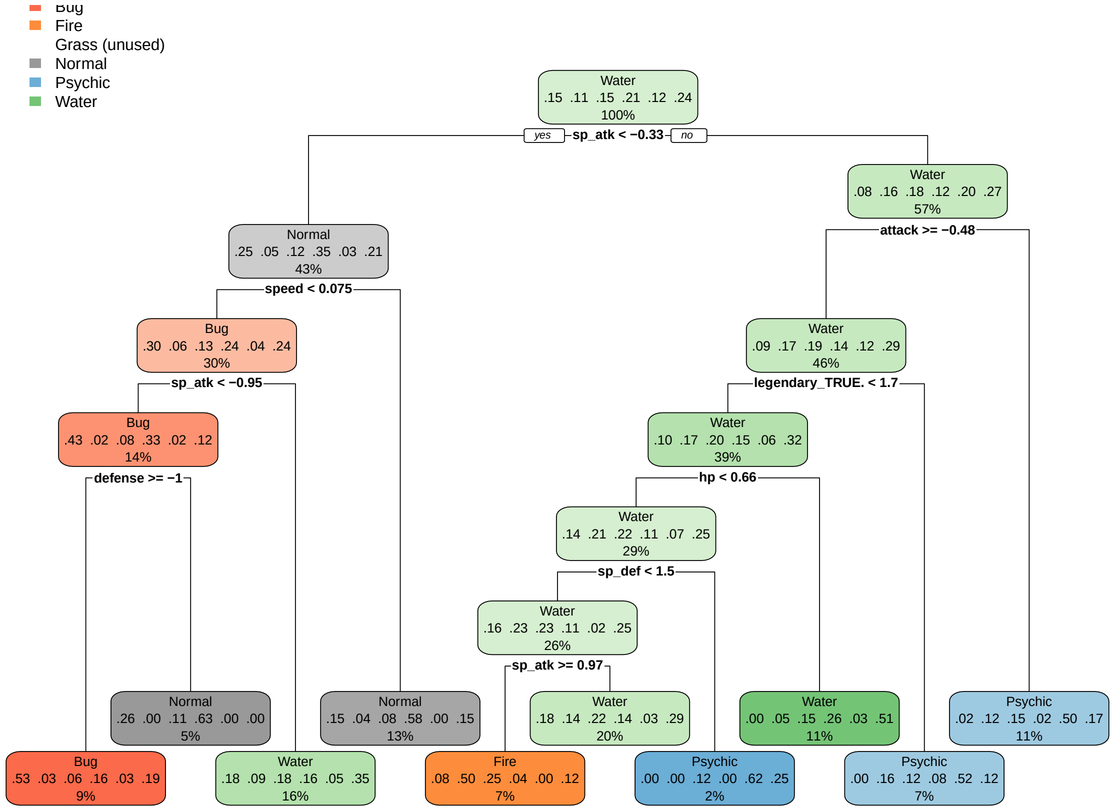
The roc_auc of my best-performing pruned decision tree on the folds is 0.6535274

Question 5

Using `rpart.plot`, fit and visualize your best-performing pruned decision tree with the *training* set.

```r
#fit model
best_model <- select_best(tune_res_tree)

class_tree_final <- finalize_workflow(class_tree_wf, best_model)

class_tree_final_fit <- fit(class_tree_final, data = pokemon_train)

# visualize
class_tree_final_fit %>%
  extract_fit_engine() %>%
  rpart.plot()
```

```
## Warning: Cannot retrieve the data used to build the model (so cannot determine roundint and is.binary
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.
```



Question 5 (continued)

Now set up a random forest model and workflow. Use the `ranger` engine and set `importance = "impurity"`. Tune `mtry`, `trees`, and `min_n`. Using the documentation for `rand_forest()`, explain in your own words what each of these hyperparameters represent.

Create a regular grid with 8 levels each. You can choose plausible ranges for each hyperparameter. Note that `mtry` should not be smaller than 1 or larger than 8. **Explain why not. What type of model would `mtry = 8` represent?**

```
set.seed(777)
# Random forest model and workflow
rf_mod <- rand_forest(mtry = tune(), trees = tune(), min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")

rf_wf <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(rf_mod)

# grid
param_grid <- grid_regular(mtry(range = c(2, 6)),
                           trees(range = c(2, 5)),
                           min_n(), levels = c(8,8,8)) # I am not sure what a good range for min_n would be
```

Mtry is the number of predictors that will be randomly chosen at each split of the tree models thus it cannot be lower than 1 since then no predictors would be chosen or greater than 8 since we only have 8 predictors. If mtry = 8 that would represent a bagging model.

Trees represents the number of trees that will be used in each model

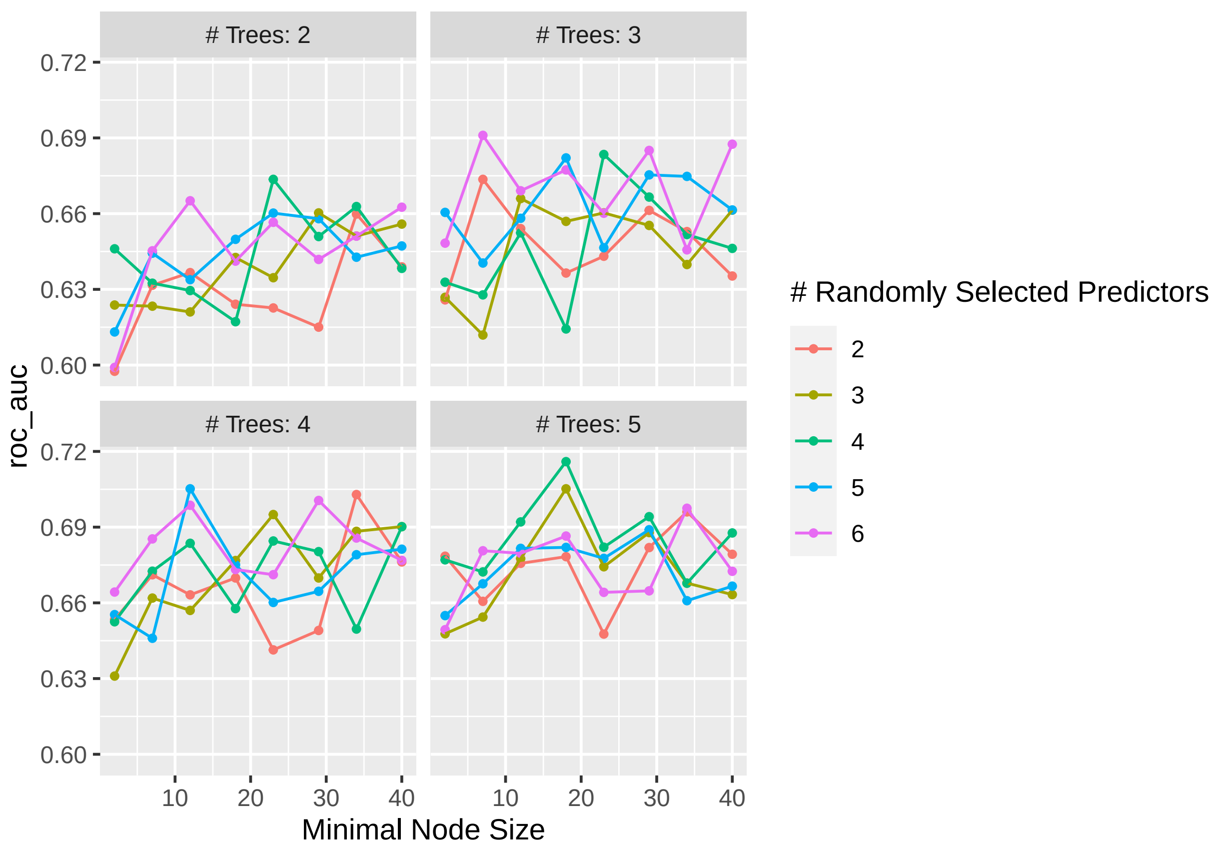
min_n represents the minimum number of of data points needed at each node/tree end that is needed in order to split it further into another tree

Question 6

Specify `roc_auc` as a metric. Tune the model and print an `autoplot()` of the results. What do you observe? What values of the hyperparameters seem to yield the best performance?

```
# will load in the model later instead of running code to save time
tune_res_rf <- tune_grid(
  rf_wf,
  resamples = pokemon_fold,
  grid = param_grid,
  metrics = metric_set(roc_auc)
)
```

```
load(file = "tunedmodels.rda")
set.seed(777)
autoplot(tune_res_rf)
```



I notice that generally, the higher number of trees tend to do better. Results for values of mtry vary differently with each number of trees. Values of min_n around 20 tend to do best across all models

Question 7

What is the `roc_auc` of your best-performing random forest model on the folds? *Hint: Use `collect_metrics()` and `arrange()`.*

```
collect_metrics(tune_res_rf) %>%
  arrange(mean)
```

```
## # A tibble: 160 x 9
##     mtry trees min_n .metric .estimator  mean     n std_err .config
##    <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     2     2     2 roc_auc hand_till  0.598     5 0.00999 Preprocessor1_Model~
##  2     6     2     2 roc_auc hand_till  0.599     5 0.0305  Preprocessor1_Model~
##  3     3     3     7 roc_auc hand_till  0.612     5 0.0176  Preprocessor1_Model~
##  4     5     2     2 roc_auc hand_till  0.613     5 0.0158  Preprocessor1_Model~
##  5     4     3    18 roc_auc hand_till  0.614     5 0.0148  Preprocessor1_Model~
##  6     2     2    29 roc_auc hand_till  0.615     5 0.0122  Preprocessor1_Model~
##  7     4     2    18 roc_auc hand_till  0.617     5 0.0128  Preprocessor1_Model~
##  8     3     2    12 roc_auc hand_till  0.621     5 0.0155  Preprocessor1_Model~
##  9     2     2    23 roc_auc hand_till  0.623     5 0.0238  Preprocessor1_Model~
## 10     3     2     7 roc_auc hand_till  0.623     5 0.0314  Preprocessor1_Model~
## # ... with 150 more rows
```

The roc_auc of my best performing random forest model on the folds is 0.7159568

Question 8

Create a variable importance plot, using `vip()`, with your best-performing random forest model fit on the *training* set.

Which variables were most useful? Which were least useful? Are these results what you expected, or not?

Question 9

Finally, set up a boosted tree model and workflow. Use the `xgboost` engine. Tune `trees`. Create a regular grid with 10 levels; let `trees` range from 10 to 2000. Specify `roc_auc` and again print an `autoplot()` of the results.

What do you observe?

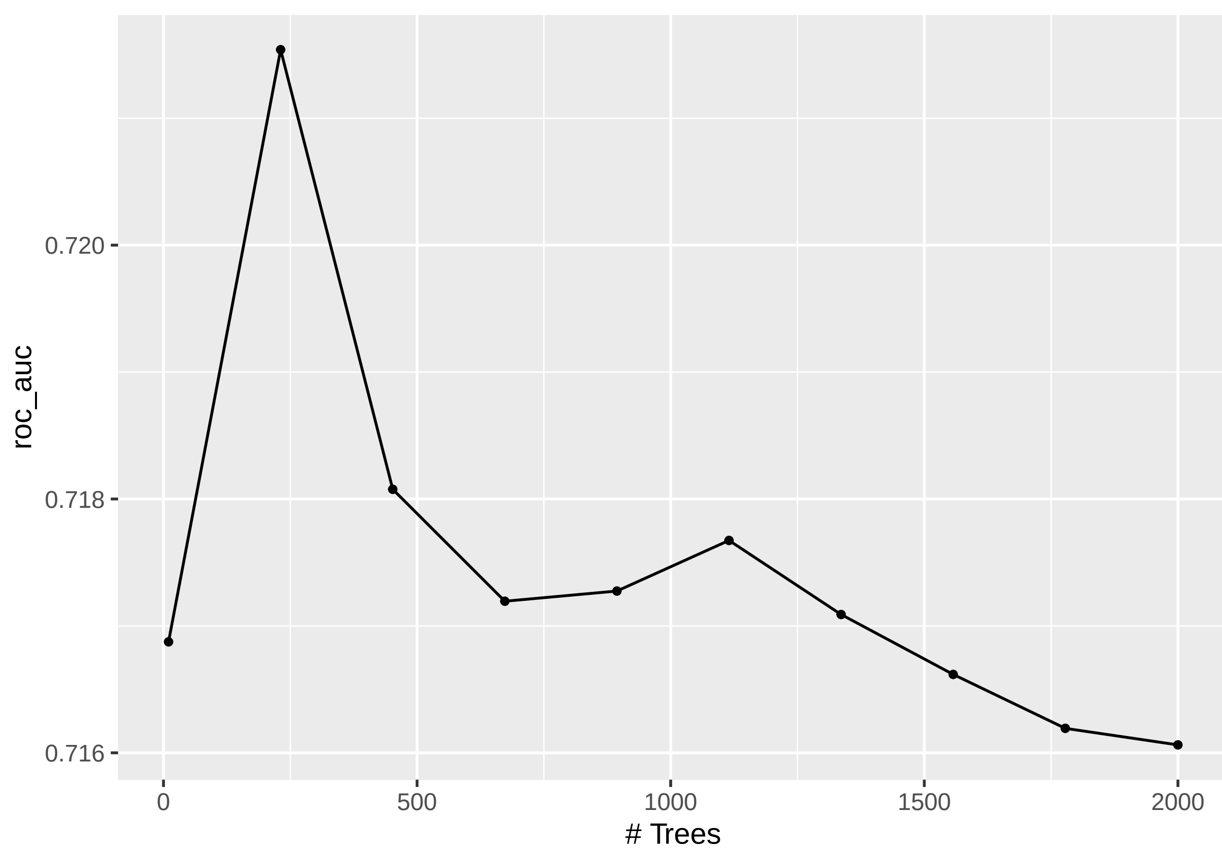
What is the `roc_auc` of your best-performing boosted tree model on the folds? *Hint: Use `collect_metrics()` and `arrange()`.*

```
boost_spec <- boost_tree(trees = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

```r
boost_wf <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(boost_spec)

# grid
param_grid <- grid_regular(trees(range = c(10, 2000)), levels = 10)
```

```r
# will load in the model later instead of running code to save time
tune_res_boost <- tune_grid(
  boost_wf,
  resamples = pokemon_fold,
  grid = param_grid,
  metrics = metric_set(roc_auc)
)
```

```r
load(file = "tunedmodels.rda")
set.seed(777)
autoplot(tune_res_boost)
```



I see that thee best number of trees is at about 240 (right before 250)

```r
collect_metrics(tune_res_boost) %>%
  arrange(mean)
```

```
## # A tibble: 10 x 7
```

```
##    trees .metric .estimator  mean     n std_err .config
##    <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1  2000 roc_auc hand_till  0.716     5  0.0148 Preprocessor1_Model10
##  2  1778 roc_auc hand_till  0.716     5  0.0147 Preprocessor1_Model09
##  3  1557 roc_auc hand_till  0.717     5  0.0146 Preprocessor1_Model08
##  4    10 roc_auc hand_till  0.717     5  0.0113 Preprocessor1_Model01
##  5  1336 roc_auc hand_till  0.717     5  0.0146 Preprocessor1_Model07
##  6   673 roc_auc hand_till  0.717     5  0.0130 Preprocessor1_Model04
##  7   894 roc_auc hand_till  0.717     5  0.0129 Preprocessor1_Model05
##  8  1115 roc_auc hand_till  0.718     5  0.0140 Preprocessor1_Model06
##  9   452 roc_auc hand_till  0.718     5  0.0132 Preprocessor1_Model03
## 10   231 roc_auc hand_till  0.722     5  0.0115 Preprocessor1_Model02
```

The best roc_auc of my best-performing boosted tree model on the folds is 0.7215399

Question 10

Display a table of the three ROC AUC values for your best-performing pruned tree, random forest, and boosted tree models. Which performed best on the folds? Select the best of the three and use `select_best()`, `finalize_workflow()`, and `fit()` to fit it to the *testing* set.

Print the AUC value of your best-performing model on the testing set. Print the ROC curves. Finally, create and visualize a confusion matrix heat map.

Which classes was your model most accurate at predicting? Which was it worst at?

```r
# Table of roc_auc values
auc_table <- matrix(c(0.6535274, 0.7159568, 0.7215399), ncol=3)
colnames(auc_table ) <- c('Pruned Tree Model','Random Forest Model','Boosted Tree Model')
rownames(auc_table ) <- c('roc_auc value')

auc_table
```

```
##               Pruned Tree Model Random Forest Model Boosted Tree Model
## roc_auc value         0.6535274           0.7159568          0.7215399
```

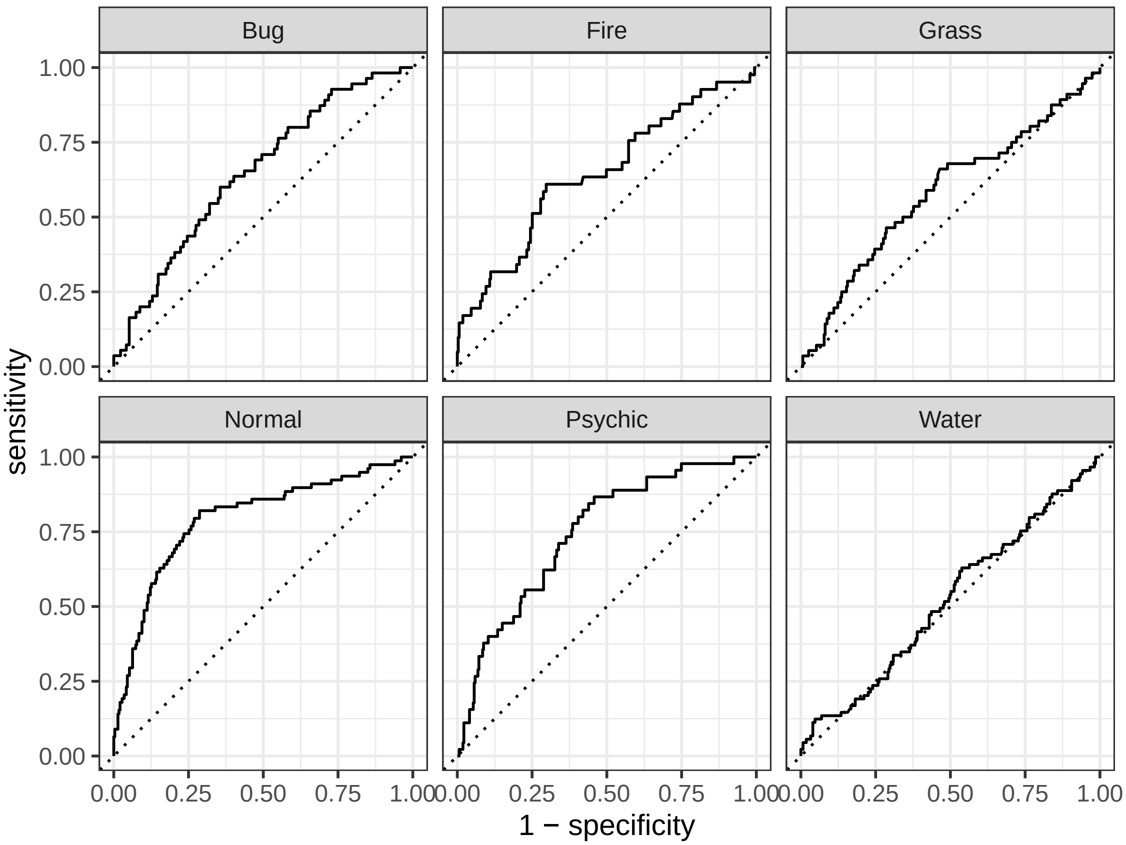
The boosted tree model performed best on the folds and so I will only use select_best() on the boosted tree model

```r
# Select best and fit to testing data

best_boosted <- select_best(tune_res_boost, metric = 'roc_auc')

boosted_final <- finalize_workflow(boost_wf, best_boosted)

overrall_final_fit <- fit(boosted_final, data = pokemon_test)
```

```r
# UC value of your best-performing model on the testing set
predicted_data <- augment(overrall_final_fit, new_data = pokemon_train) %>%
  select(type_1, starts_with(".pred"))

predicted_data  %>% roc_auc(type_1, .pred_Bug:.pred_Water)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.654
```

```
# Print the ROC curves
predicted_data %>% roc_curve(type_1, .pred_Bug:.pred_Water) %>%
  autoplot()
```



Model best predicted normal and psychic types and was worst at predicting water types