Tree-Based Models

Code ▾

Homework 6

PSTAT 131/231

Tree-Based Models

For this assignment, we will continue working with the file `"pokemon.csv"`, found in `/data`. The file is from Kaggle: https://www.kaggle.com/abcsds/pokemon (https://www.kaggle.com/abcsds/pokemon).

The Pokémon (https://www.pokemon.com/us/) franchise encompasses video games, TV shows, movies, books, and a card game. This data set was drawn from the video game series and contains statistics about 721 Pokémon, or "pocket monsters." In Pokémon games, the user plays as a trainer who collects, trades, and battles Pokémon to (a) collect all the Pokémon and (b) become the champion Pokémon trainer.

Each Pokémon has a primary type (https://bulbapedia.bulbagarden.net/wiki/Type) (some even have secondary types). Based on their type, a Pokémon is strong against some types, and vulnerable to others. (Think rock, paper, scissors.) A Fire-type Pokémon, for example, is vulnerable to Water-type Pokémon, but strong against Grass-type.

Fig 1. Houndoom, a Dark/Fire-type canine Pokémon from Generation II.

The goal of this assignment is to build a statistical learning model that can predict the **primary type** of a Pokémon based on its generation, legendary status, and six battle statistics.

Note: Fitting ensemble tree-based models can take a little while to run. Consider running your models outside of the .Rmd, storing the results, and loading them in your .Rmd to minimize time to knit.

Exercise 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.

Hide

```
#install.packages("rpart.plot")
#install.packages("vip")
#install.packages("randomForest")
#install.packages("xgboost")
#install.packages("ranger")
library(tidymodels)
library(MASS)
library(dplyr)
library(ISLR)
library(ISLR2)
library(tidyverse)
library(janitor)
library(corrplot)
library(glmnet)
library(rpart.plot)
library(vip)
library(randomForest)
library(xgboost)
library(ranger)
tidymodels_prefer()
```

Hide

```
Poke <- read.csv(file="/Users/honchowayne/Desktop/Pokemon.csv")
Poke <- clean_names(Poke)
Poke<- filter(Poke,type_1 %in% c('Bug', 'Fire', 'Grass', 'Normal', 'Water', 'Psychi
c'))
Poke$type_1 <- factor(Poke$type_1)
Poke$legendary <- factor(Poke$legendary)
Poke$generation <- factor(Poke$generation)
```

Hide

```
set.seed(3435)
Poke_split <- initial_split(Poke, prop = 0.70, strata = type_1)
Poke_train <- training(Poke_split)
Poke_test <- testing(Poke_split)
```

Hide

```
Poke_fold <- vfold_cv(Poke_train, v=5, strata = type_1)

Poke_rec <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defe
nse + hp + sp_def, Poke_train) %>%
  step_dummy(c(legendary,generation)) %>%
  step_center(all_predictors()) %>%
  step_normalize(all_predictors())
```

Exercise 2

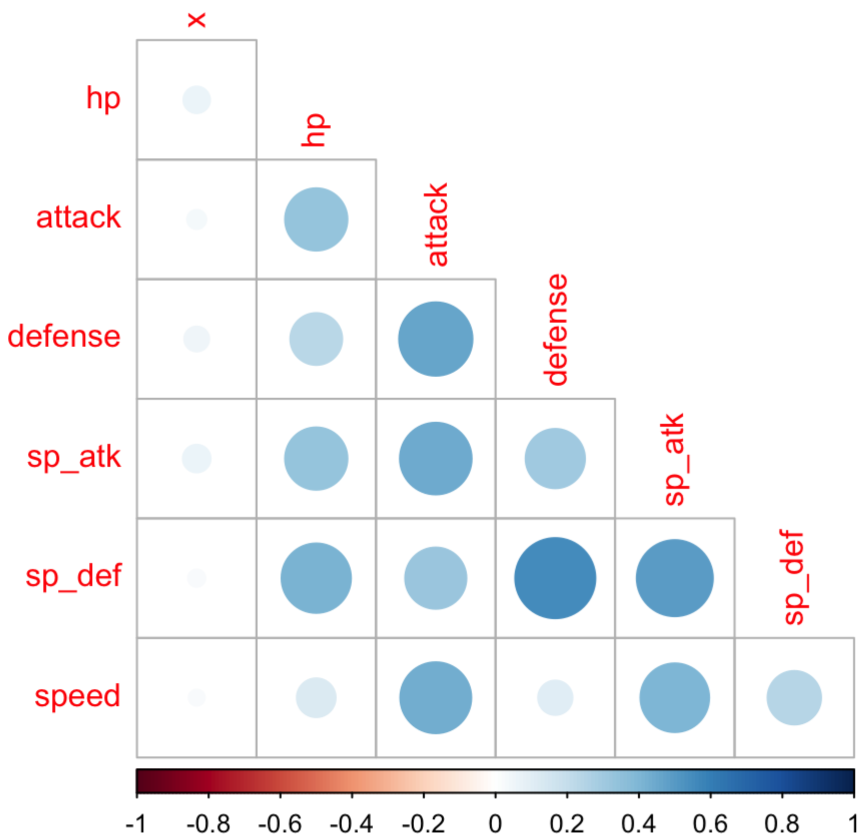
Create a correlation matrix of the training set, using the `corrplot` package. *Note: You can choose how to handle the continuous variables for this plot; justify your decision(s).*

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

I exclude the "total" column because it is the sum of all the other attributes, so of course it has strong correlation with each of the other attributes. I also exclude the ID column because it is like a more of factor here to identity each Pokemon. Other than these two, the strong positive relationships such as special attack & attack ability and special defense & defense ability make the most sense to me. One part that keep me puzzled for a while is the relatively weak relationship between health & defense.

Hide

```
Poke %>%
  select(where(is.numeric),-total) %>%
  cor(use = "complete.obs") %>%
  corrplot(type = "lower", diag = FALSE)
```



Exercise 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?

It perform better with a "above the average" complexity penalty. It starts with a slow upward trend, reaches the top and then drops extremely fast.

Hide

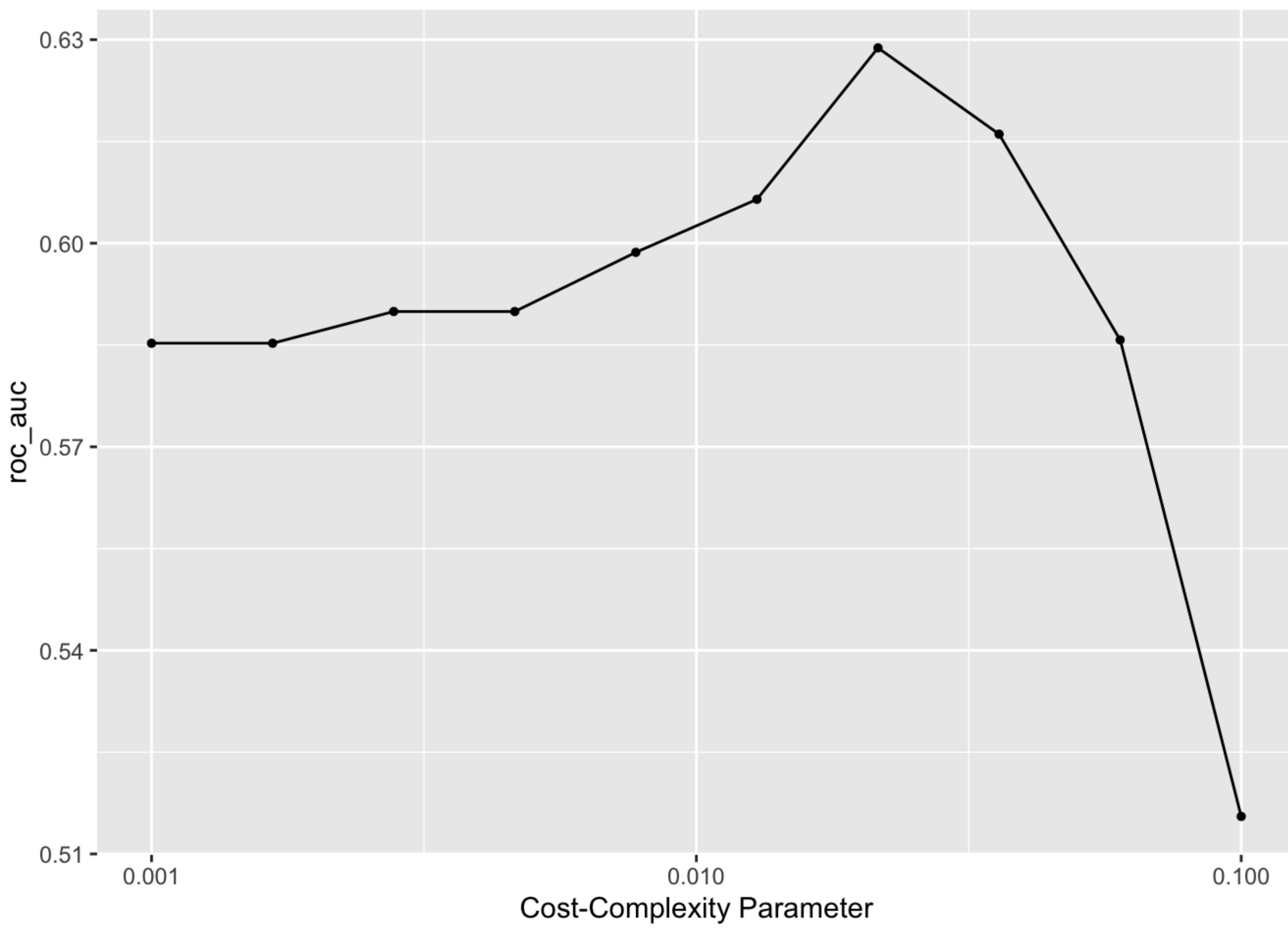
```
class_tree_spec <- decision_tree() %>%
  set_mode("classification") %>%
  set_engine("rpart")

class_tree_wf <- workflow() %>%
  add_model(class_tree_spec %>% set_args(cost_complexity = tune())) %>%
  add_recipe(Poke_rec)

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

tune_res <- tune_grid(
  class_tree_wf,
  resamples = Poke_fold,
  grid = param_grid,
  metrics = metric_set(roc_auc)
)

autoplot(tune_res)
```



Exercise 4

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

Hide

```
collect_metrics(tune_res) %>%
  arrange(-mean)
```

```
## # A tibble: 10 × 7
##    cost_complexity .metric .estimator  mean     n std_err .config
##              <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         0.0215  roc_auc hand_till  0.629     5  0.0191 Preprocessor1_Model07
##  2         0.0359  roc_auc hand_till  0.616     5  0.0202 Preprocessor1_Model08
##  3         0.0129  roc_auc hand_till  0.606     5  0.0168 Preprocessor1_Model06
##  4         0.00774 roc_auc hand_till  0.599     5  0.0179 Preprocessor1_Model05
##  5         0.00278 roc_auc hand_till  0.590     5  0.0173 Preprocessor1_Model03
##  6         0.00464 roc_auc hand_till  0.590     5  0.0173 Preprocessor1_Model04
##  7         0.0599  roc_auc hand_till  0.586     5  0.0233 Preprocessor1_Model09
##  8         0.001   roc_auc hand_till  0.585     5  0.0160 Preprocessor1_Model01
##  9         0.00167 roc_auc hand_till  0.585     5  0.0160 Preprocessor1_Model02
## 10         0.1     roc_auc hand_till  0.516     5  0.0155 Preprocessor1_Model10
```

Hide

```
best_class_tree <- select_best(tune_res, metric = "roc_auc")
best_class_tree
```

```
## # A tibble: 1 × 2
##   cost_complexity .config
##             <dbl> <chr>
## 1          0.0215 Preprocessor1_Model07
```

Hide

```
class_roc <- show_best(tune_res, metric = "roc_auc")
class_roc <- class_roc[1,]
class_roc
```

```
## # A tibble: 1 × 7
##   cost_complexity .metric .estimator  mean     n std_err .config
##             <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1          0.0215 roc_auc hand_till  0.629     5  0.0191 Preprocessor1_Model07
```

Exercise 5

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

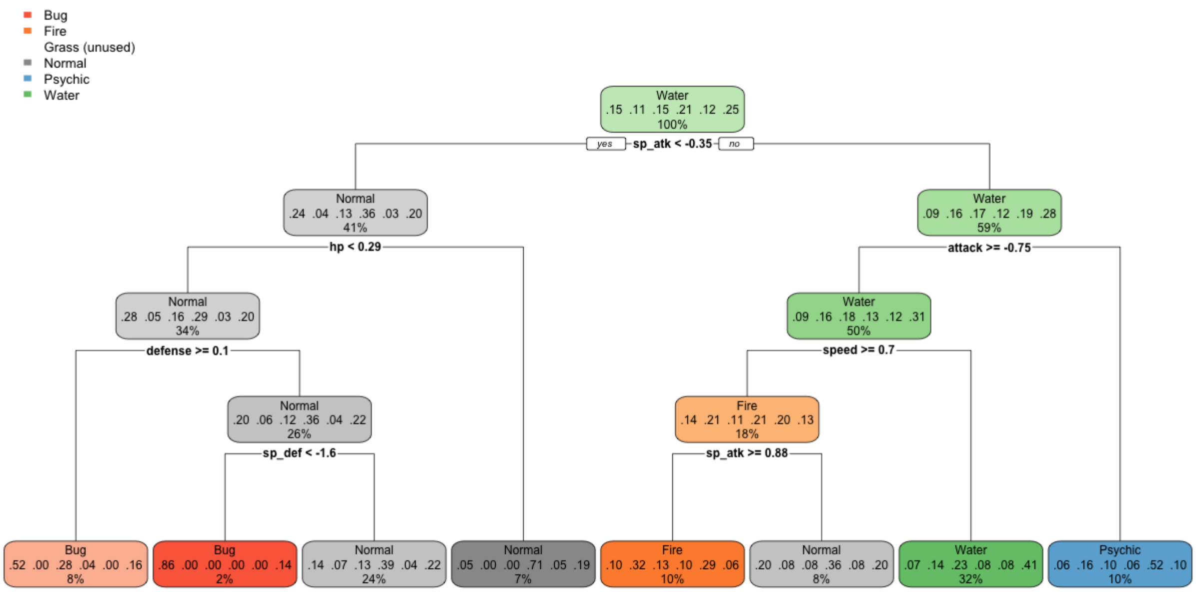
Hide

```
best_complexity <- select_best(tune_res)

class_tree_final <- finalize_workflow(class_tree_wf, best_complexity)

class_tree_final_fit <- fit(class_tree_final, data = Poke_train)

class_tree_final_fit %>%
  extract_fit_engine() %>%
  rpart.plot(roundint=FALSE)
```



Exercise 5

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.

mtry is the number of variables randomly sampled as candidates at each split. trees means the number of trees used in aggregation. min_n is the minimum number of data points in a node that are required for the node to be split further.

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?**

mtry cannot go larger than 8 because we only have 8 predictors here. if set mtry = 8, then it is a bagging model

Hide

```
rf_spec <- rand_forest() %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")

rf_wf <- workflow() %>%
  add_model(rf_spec %>% set_args(mtry = tune(), trees = tune(), min_n = tune())) %
>%
  add_recipe(Poke_rec)

rf_grid <- grid_regular(mtry(range = c(1,8)), trees(range = c(1,100)), min_n(range
 = c(1,8)), levels = 8)
```

Exercise 6

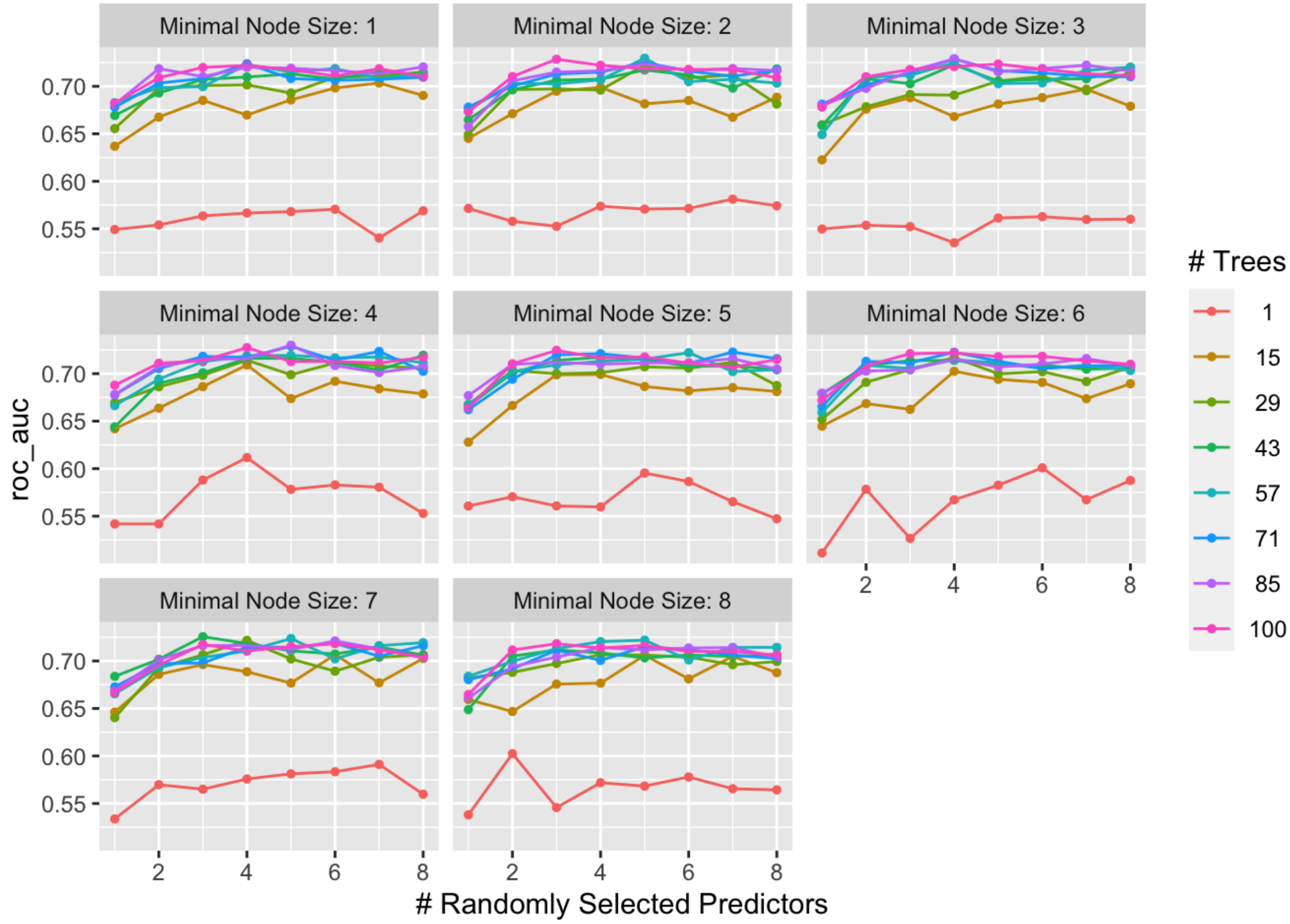
Specify `roc_auc` as a metric. Tune the model and print an `autoplot()` of the results. What do you observe? What values of the hyperparameters seem to yield the best performance?

it seems having more decisions in the tree improves the model performance.

Hide

```
rf_tune_res <- tune_grid(
  rf_wf,
  resamples = Poke_fold,
  grid = rf_grid,
  metrics = metric_set(roc_auc)
)

autoplot(rf_tune_res)
```



Hide

```
show_best(rf_tune_res, metric = "roc_auc")
```

```
## # A tibble: 5 × 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     5    85     4 roc_auc hand_till  0.730     5  0.0134 Preprocessor1_Model2…
## 2     5    57     2 roc_auc hand_till  0.730     5  0.0102 Preprocessor1_Model1…
## 3     4    85     3 roc_auc hand_till  0.729     5  0.0162 Preprocessor1_Model1…
## 4     5    71     4 roc_auc hand_till  0.729     5  0.0172 Preprocessor1_Model2…
## 5     4    71     3 roc_auc hand_till  0.729     5  0.0132 Preprocessor1_Model1…
```

Exercise 7

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

Hide

```
collect_metrics(rf_tune_res) %>%
  arrange(-mean)
```

```
## # A tibble: 512 × 9
##     mtry trees min_n .metric .estimator  mean     n std_err .config
##    <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     5    85     4 roc_auc hand_till  0.730     5  0.0134 Preprocessor1_Model…
##  2     5    57     2 roc_auc hand_till  0.730     5  0.0102 Preprocessor1_Model…
##  3     4    85     3 roc_auc hand_till  0.729     5  0.0162 Preprocessor1_Model…
##  4     5    71     4 roc_auc hand_till  0.729     5  0.0172 Preprocessor1_Model…
##  5     4    71     3 roc_auc hand_till  0.729     5  0.0132 Preprocessor1_Model…
##  6     3   100     2 roc_auc hand_till  0.728     5  0.0203 Preprocessor1_Model…
##  7     4   100     4 roc_auc hand_till  0.727     5  0.0146 Preprocessor1_Model…
##  8     3    43     7 roc_auc hand_till  0.726     5  0.0134 Preprocessor1_Model…
##  9     4    57     3 roc_auc hand_till  0.725     5  0.0156 Preprocessor1_Model…
## 10     5    71     2 roc_auc hand_till  0.725     5  0.0116 Preprocessor1_Model…
## # … with 502 more rows
```

Hide

```
best_rf <- select_best(rf_tune_res, metric = "roc_auc")
best_rf
```

```
## # A tibble: 1 × 4
##    mtry trees min_n .config
##   <int> <int> <int> <chr>
## 1     5    85     4 Preprocessor1_Model245
```

Hide

```
rf_roc <- show_best(rf_tune_res, metric = "roc_auc")
rf_roc <- rf_roc[1,]
rf_roc
```

```
## # A tibble: 1 × 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     5    85     4 roc_auc hand_till  0.730     5  0.0134 Preprocessor1_Model2…
```

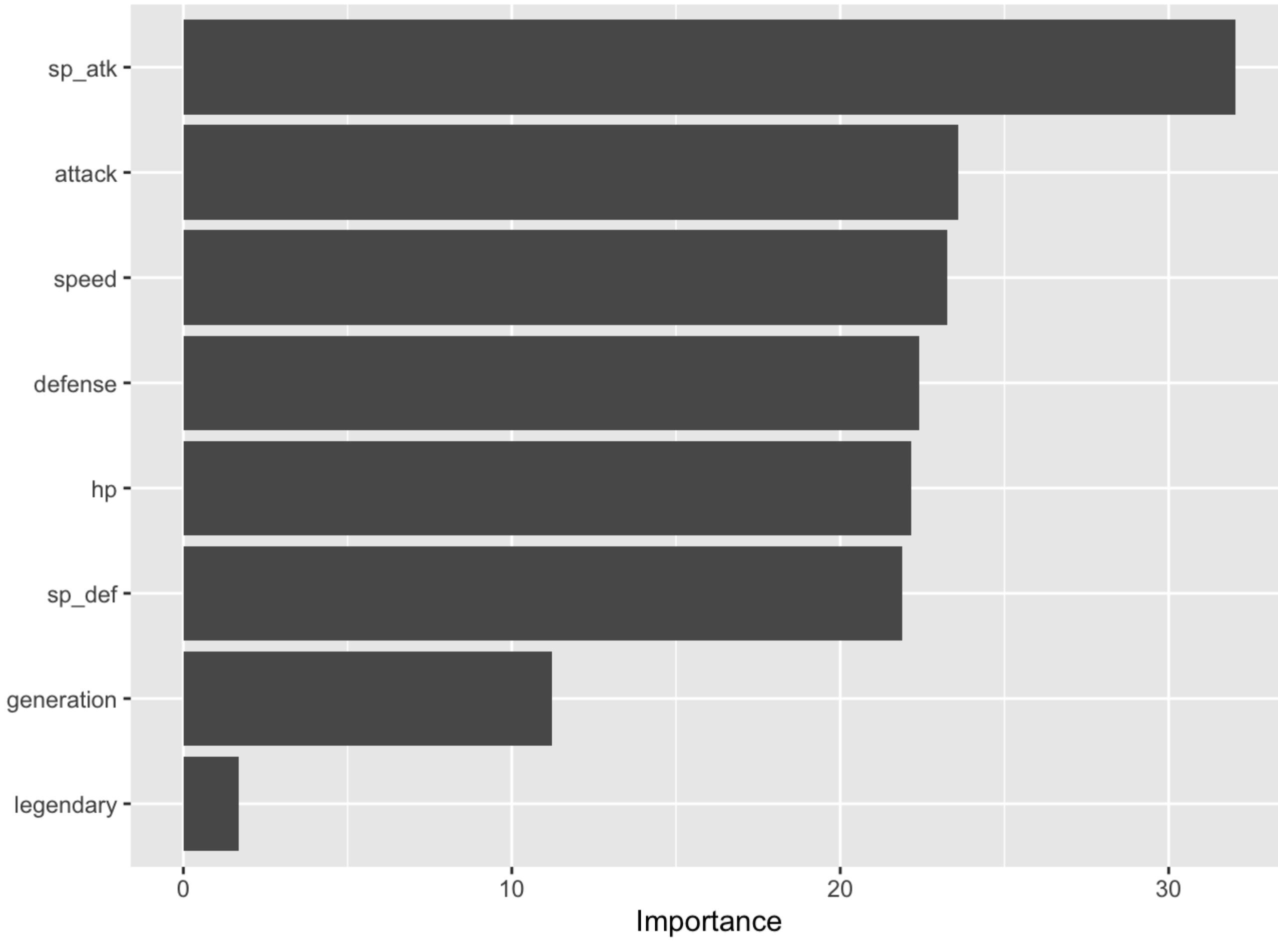
Exercise 8

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?

The special attach are the most useful variable. Besides, attck, defense, health, speed and special defense are also useful. Legendary is the least useful. The result fits my expectation because type does determines a Pokemon’s weakness/resistance to attacks.

Hide

```
rf_sepc_final <- finalize_model(rf_spec, best_rf)
rf_fit <- fit(rf_spec, type_1 ~ sp_atk + attack + speed + hp + defense + sp_def + g
eneration + legendary, data = Poke_train)
vip(rf_fit)
```



Exercise 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?

The image rises in a precipitous manner, reaches the top and then returns to a stable and decreasing trend.

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

Hide

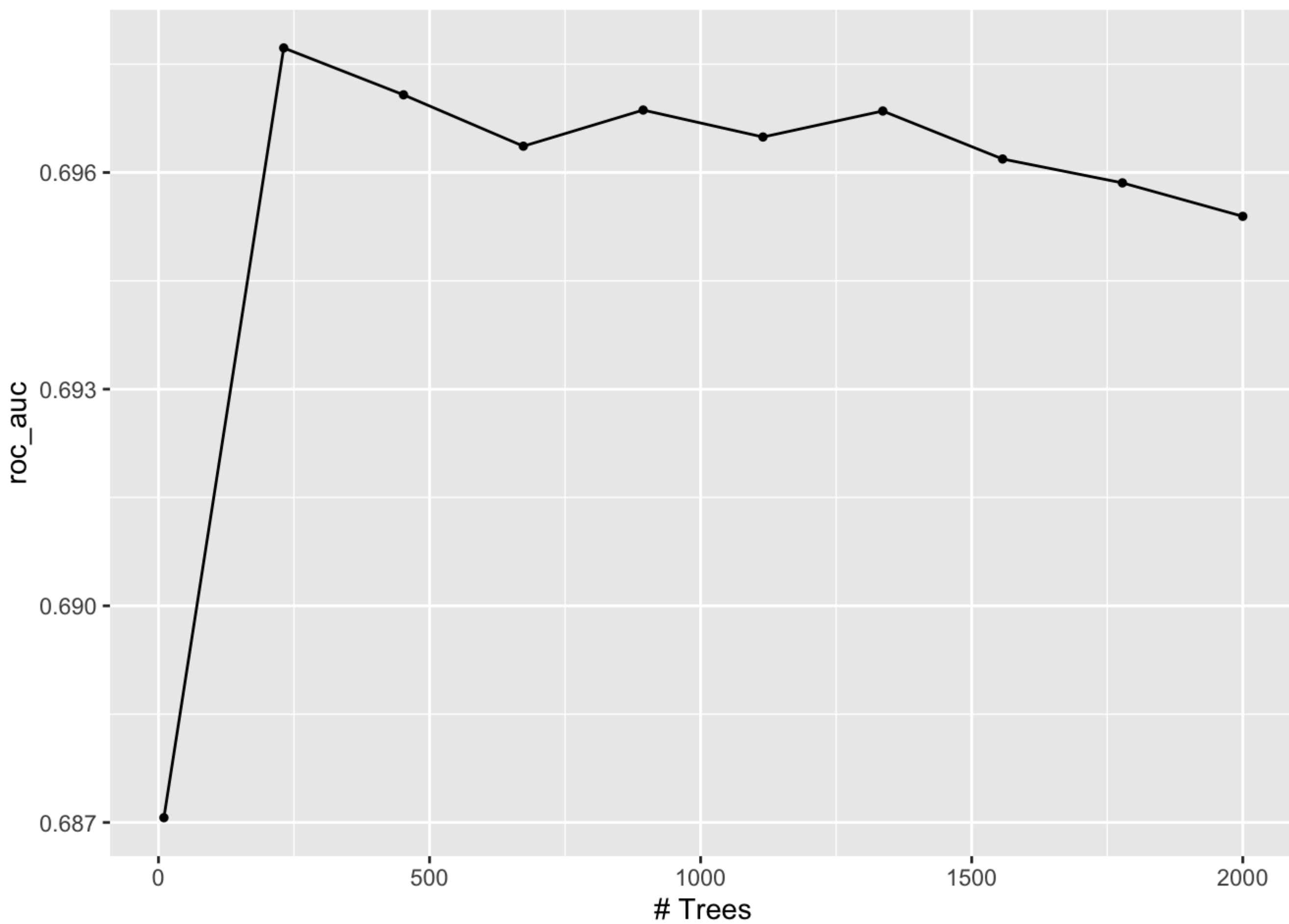
```
boost_spec <- boost_tree() %>%
  set_engine("xgboost") %>%
  set_mode("classification")

boost_wf <- workflow() %>%
  add_model(boost_spec %>% set_args(trees = tune())) %>%
  add_recipe(Poke_rec)

boost_grid <- grid_regular(trees(range = c(10, 2000)), levels = 10)

boost_tune_res <- tune_grid(
  boost_wf,
  resamples = Poke_fold,
  grid = boost_grid,
  metrics = metric_set(roc_auc)
)

autoplot(boost_tune_res)
```



Hide

```
best_boost <- select_best(boost_tune_res, metric = "roc_auc")
best_boost
```

```
## # A tibble: 1 × 2
##   trees .config
##   <int> <chr>
## 1   231 Preprocessor1_Model02
```

Hide

```
boost_roc <-collect_metrics(boost_tune_res) %>%
  arrange
boost_roc <- boost_roc[2,]
boost_roc
```

```
## # A tibble: 1 × 7
##   trees .metric .estimator  mean     n std_err .config
##   <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   231 roc_auc hand_till  0.698     5  0.0139 Preprocessor1_Model02
```

Exercise 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?

By inspection, Normal class is my model most accurate at predicting bacause it has the biggest AUC value and Psychic is the worst due to the least AUC value

Hide

```
all_three <- bind_rows(class_roc, rf_roc, boost_roc)
all_three
```

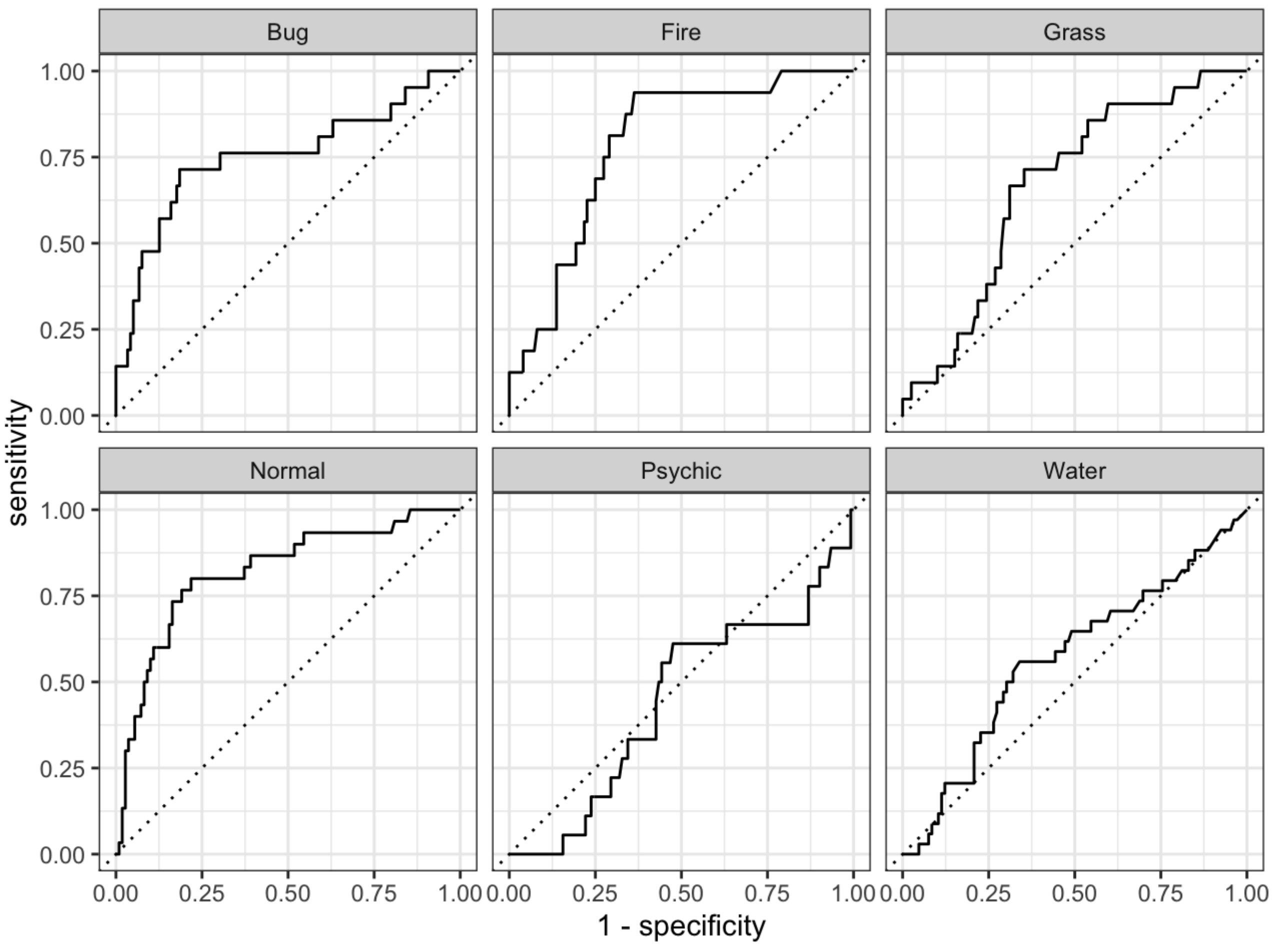
```
## # A tibble: 3 × 10
##   cost_complexity .metric .estimator  mean     n std_err .config      mtry trees
##             <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>       <int> <int>
## 1          0.0215 roc_auc hand_till  0.629     5  0.0191 Preprocess…    NA    NA
## 2         NA      roc_auc hand_till  0.730     5  0.0134 Preprocess…     5    85
## 3         NA      roc_auc hand_till  0.698     5  0.0139 Preprocess…    NA   231
## # … with 1 more variable: min_n <int>
```

Hide

```
# from these three, random forest's roc_auc performs the best
```

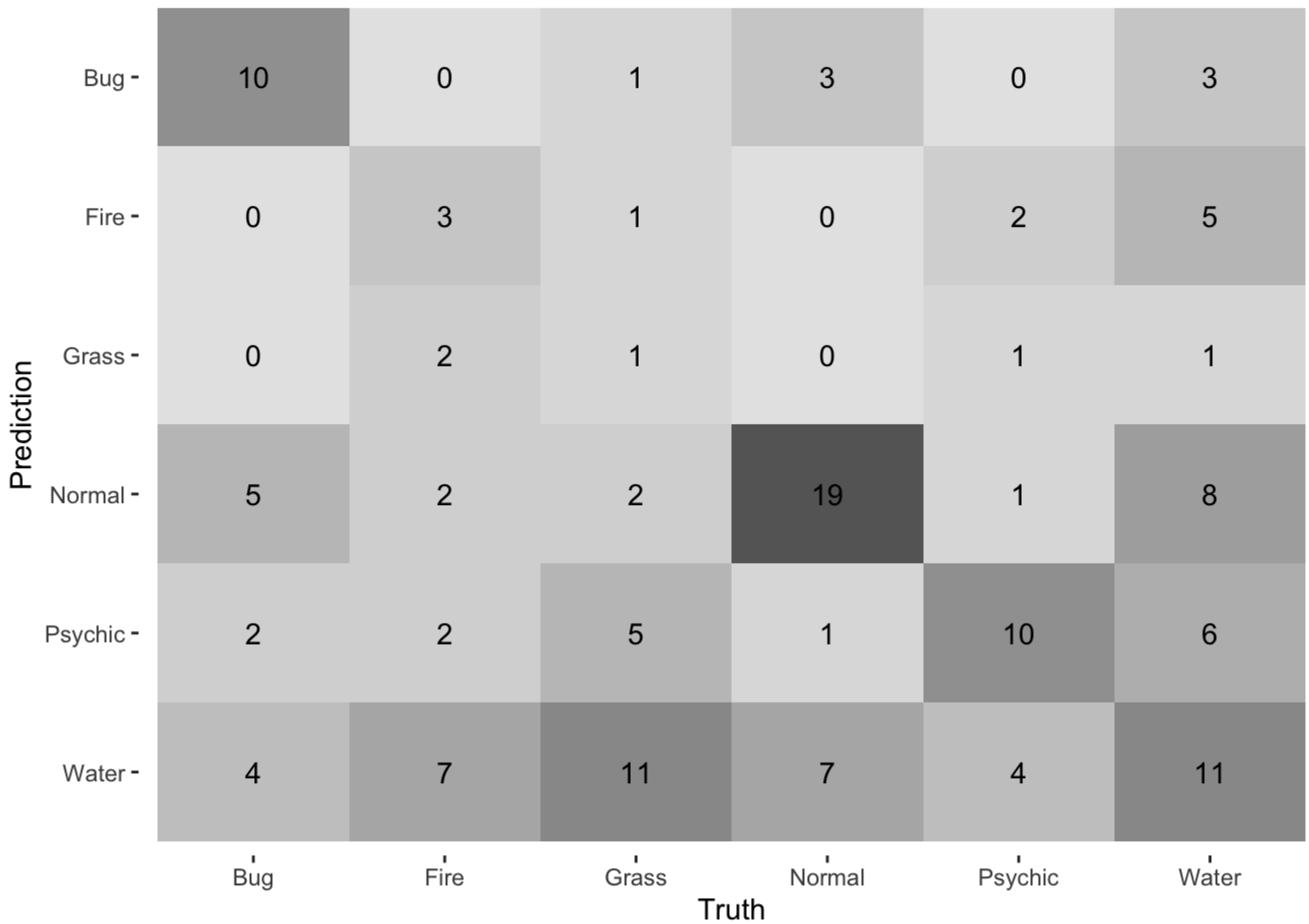
Hide

```
best_rf <- select_best(rf_tune_res, metric = "roc_auc")
final_wf<-finalize_workflow(rf_wf, best_rf)
final_fit <- fit(final_wf, data = Poke_train)
rf_roc_plot <- augment(final_fit, new_data = Poke_test) %>%
  roc_curve(type_1, estimate = c(.pred_Bug, .pred_Fire, .pred_Grass,
                                  .pred_Normal, .pred_Water, .pred_Psychic)) %>%
  autoplot()
rf_roc_plot
```



Hide

```
rf_roc_plot <- augment(final_fit, new_data = Poke_test)
rf_roc_plot %>%
  conf_mat(type_1, .pred_class) %>%
  autoplot(type = 'heatmap')
```



For 231 Students

Exercise 11

Using the `abalone.txt` data from previous assignments, fit and tune a random forest model to predict `age`. Use stratified cross-validation and select ranges for `mtry`, `min_n`, and `trees`. Present your results. What was the model's RMSE on your testing set?