Homework 6

PSTAT 131/231

Tree-Based Models

```
library(tidyverse)
library(tidymodels)
library(rpart.plot)
library(vip)
library(janitor)
library(randomForest)
library(xgboost)
library(ranger)
tidymodels_prefer()
```

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.

```
pokemon <- read.csv("Pokemon.csv")
pokemon_clean <- clean_names(pokemon)

pokemon_filter <- pokemon_clean %>%
  filter(type_1 == c("Bug", "Fire", "Grass", "Normal", "Water", "Psychic")) %>%
  mutate_at(vars(type_1, legendary), factor)

set.seed(2022)

pokemon_split <- initial_split(pokemon_filter, prop = 0.70,
                               strata = type_1)
pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)

dim(pokemon_train)
```

```
## [1] 54 13
```

```r
dim(pokemon_test)
```

```
## [1] 28 13
```

```r
pokemon_folds <- vfold_cv(pokemon_train, v = 5, strata = type_1)
pokemon_folds
```

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 x 2
##   splits          id
##   <list>          <chr>
## 1 <split [40/14]> Fold1
## 2 <split [42/12]> Fold2
## 3 <split [43/11]> Fold3
## 4 <split [45/9]>  Fold4
## 5 <split [46/8]>  Fold5
```

```r
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack
                         + speed + defense + hp + sp_def, data = pokemon_train) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_center(all_numeric_predictors()) %>%
  step_scale(all_numeric_predictors())

pokemon_recipe
```

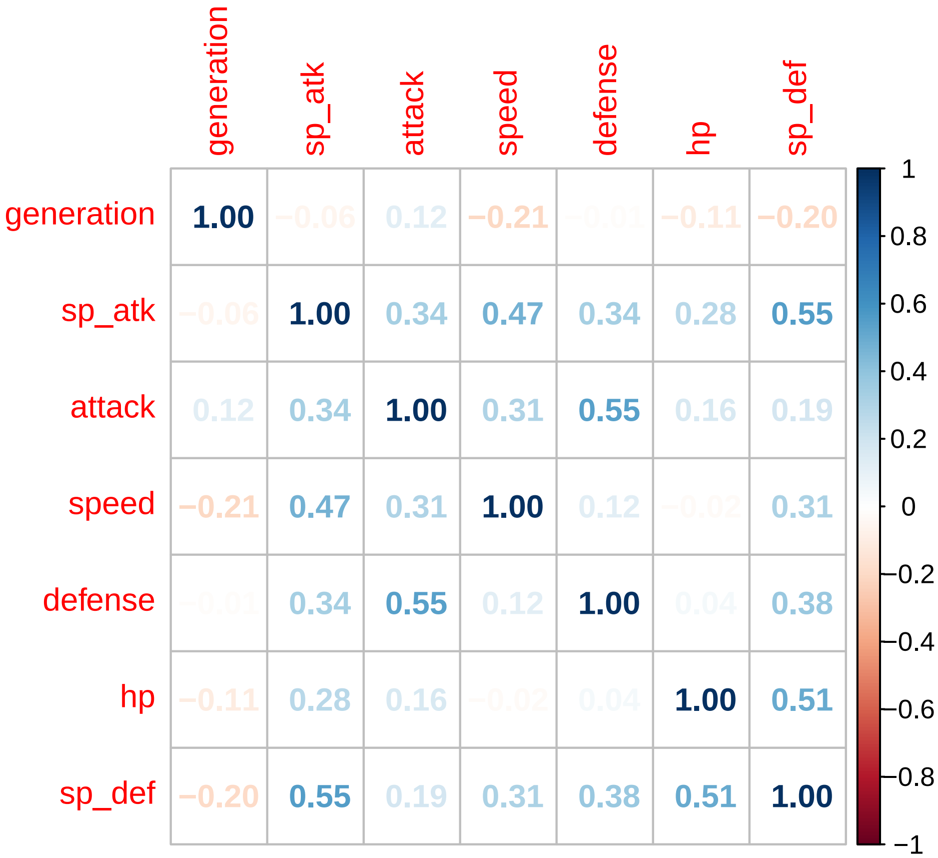
```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          8
##
## Operations:
##
## Dummy variables from all_nominal_predictors()
## Centering for all_numeric_predictors()
## Scaling for all_numeric_predictors()
```

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?

```r
library(corrplot)

cor_pokemon <- pokemon_train %>%
  select(c(generation, sp_atk, attack, speed, defense, hp, sp_def)) %>%
  cor()

corrplot(cor_pokemon, method = "number")
```



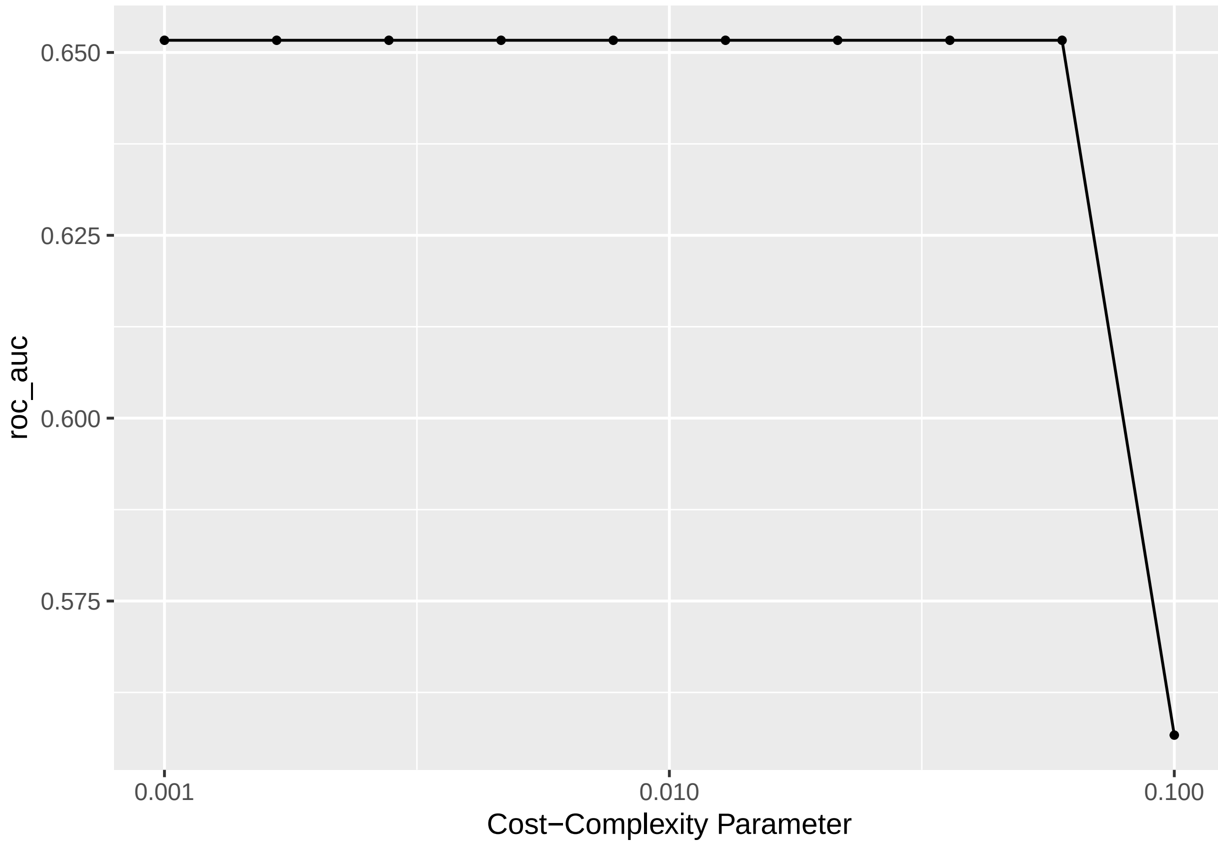
Exercise 3

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?

```r
tree_spec <- decision_tree() %>%
  set_engine("rpart")

class_tree_spec <- tree_spec %>%
  set_mode("classification")

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

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

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

```
)

autoplot(tune_res)
```



Solution: A single decision tree performs better with a smaller complexity penalty.

Exercise 4

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

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

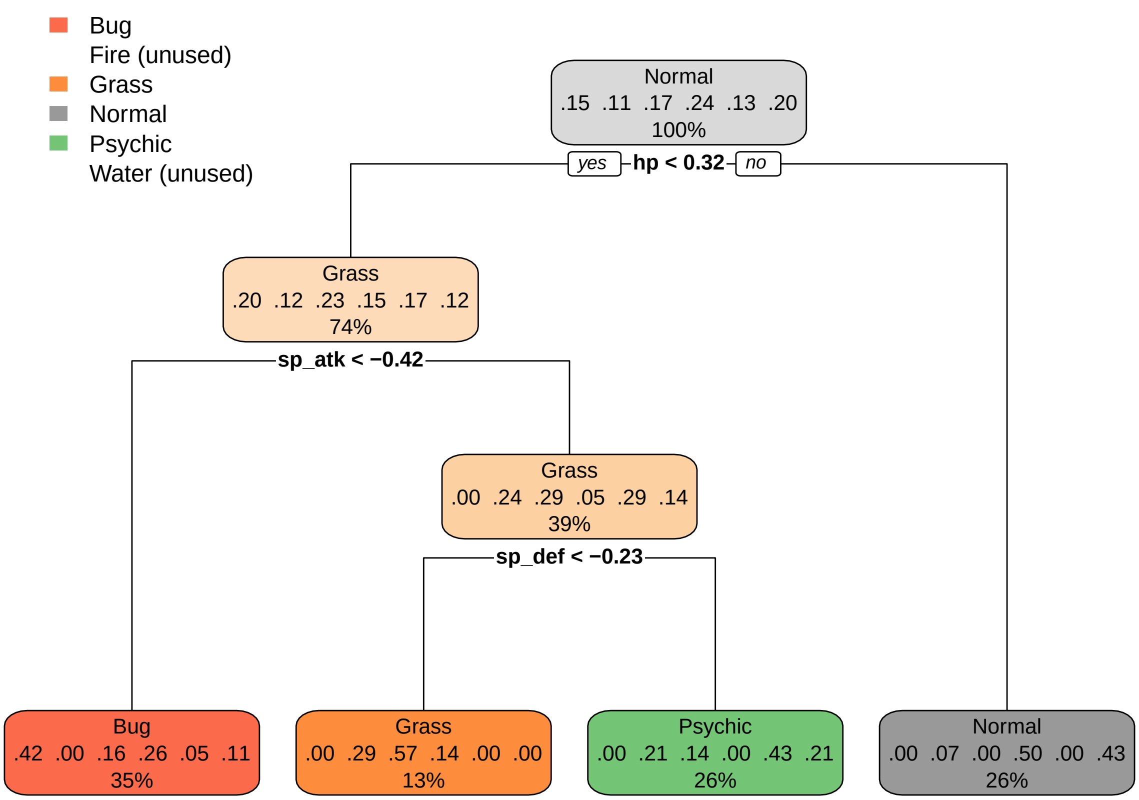
```
## # A tibble: 10 x 7
##    cost_complexity .metric .estimator  mean     n std_err .config
##              <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         0.001   roc_auc hand_till  0.652     5  0.0465 Preprocessor1_Model01
##  2         0.00167 roc_auc hand_till  0.652     5  0.0465 Preprocessor1_Model02
##  3         0.00278 roc_auc hand_till  0.652     5  0.0465 Preprocessor1_Model03
##  4         0.00464 roc_auc hand_till  0.652     5  0.0465 Preprocessor1_Model04
##  5         0.00774 roc_auc hand_till  0.652     5  0.0465 Preprocessor1_Model05
##  6         0.0129  roc_auc hand_till  0.652     5  0.0465 Preprocessor1_Model06
##  7         0.0215  roc_auc hand_till  0.652     5  0.0465 Preprocessor1_Model07
##  8         0.0359  roc_auc hand_till  0.652     5  0.0465 Preprocessor1_Model08
##  9         0.0599  roc_auc hand_till  0.652     5  0.0465 Preprocessor1_Model09
## 10         0.1     roc_auc hand_till  0.557     5  0.0314 Preprocessor1_Model10
```

Solution: The roc_auc of best-performing pruned decision tree on the folds is 0.651667.

Exercise 5

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

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

class_tree_final <- finalize_workflow(class_tree_wf, best_complexity)

class_tree_final_fit <- fit(class_tree_final, data = pokemon_train)

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



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.

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?**

```
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(pokemon_recipe)

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

Solution: mtry: the number of predictors that will be randomly sampled.

trees: the number of trees contained in the ensemble.

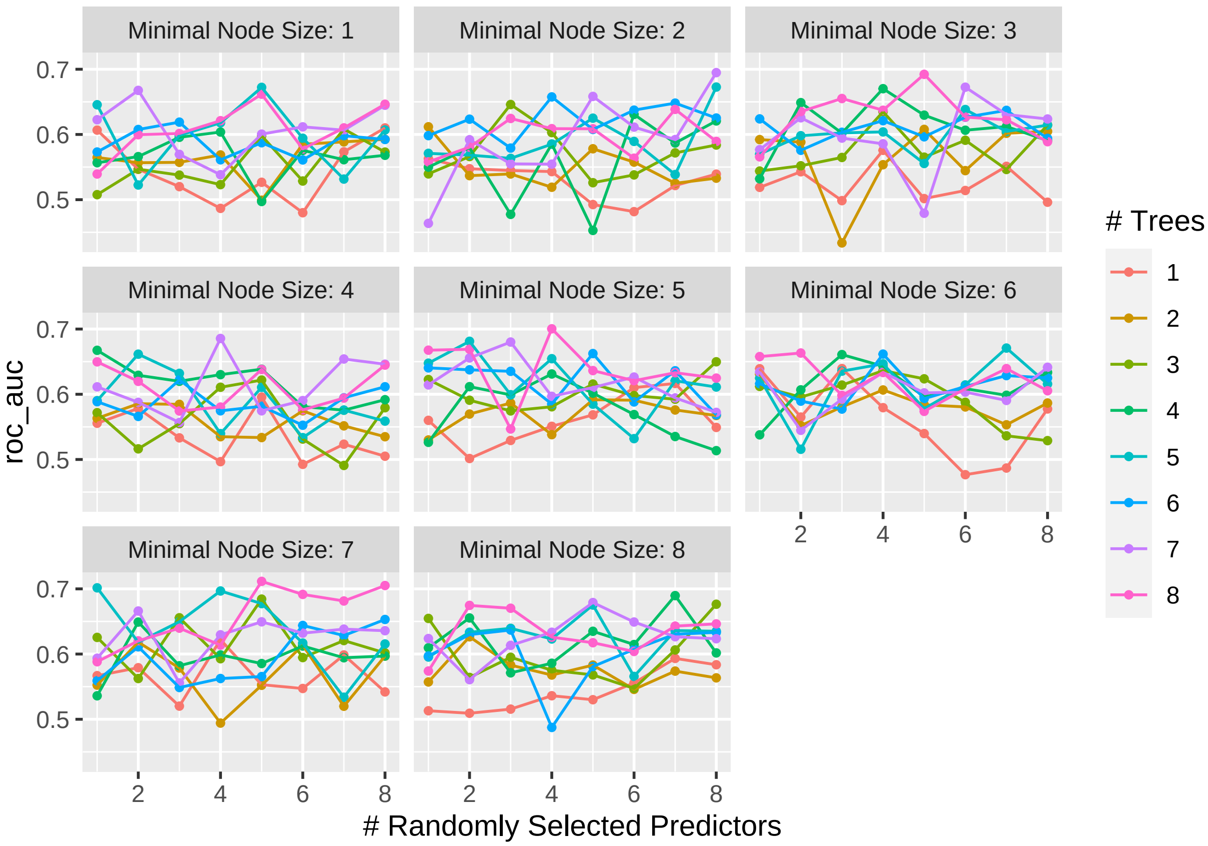
min_n: the minimum number of data points in a node that are required for the node to be split further.

Because there are only 8 predictors thus 1 <= mtry <= 8. mtry = 8 means we use all the predictors to be randomly sampled at each split when creating the tree models.

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?

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

autoplot(rf_tune_res)
```



Solution: The relationship is complex. trees = 8, mtys = 5, min_n = 7 seems to yield the best performance.

Exercise 7

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

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

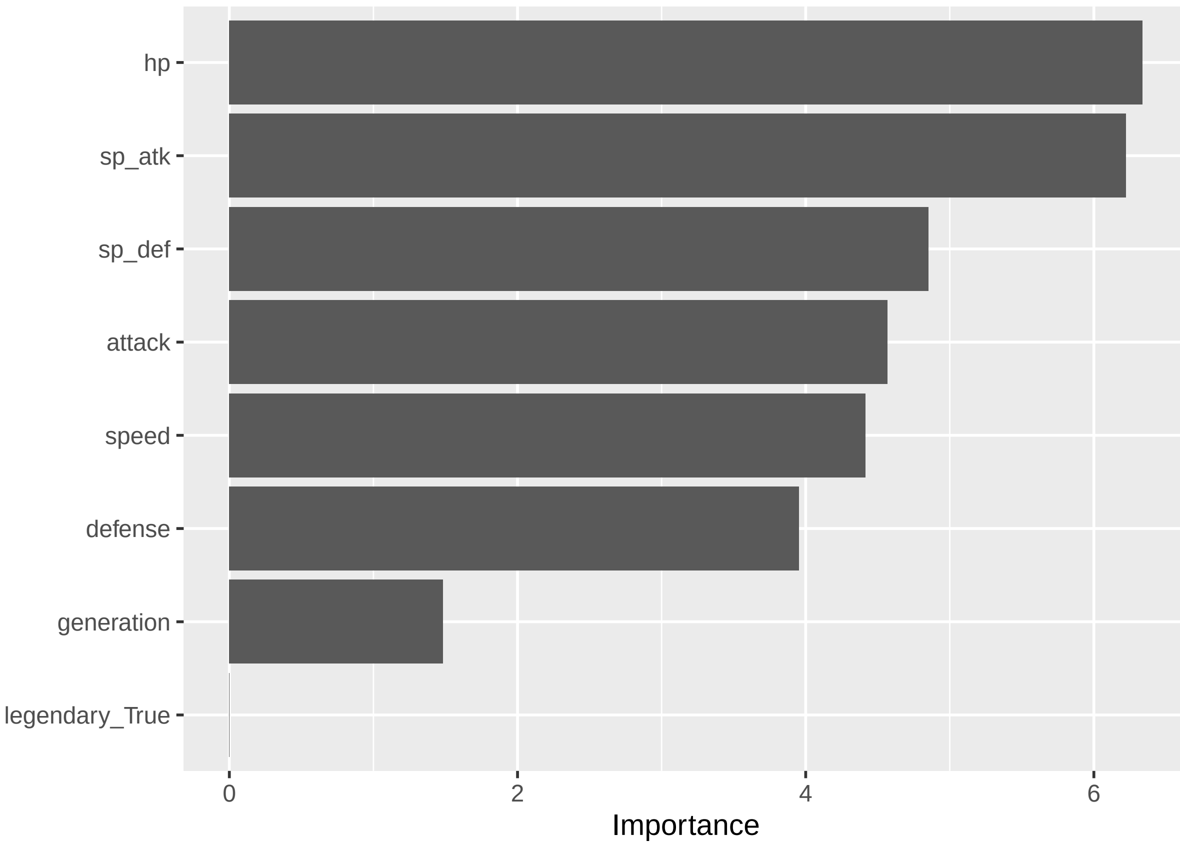
```
## # A tibble: 512 x 9
##     mtry trees min_n .metric .estimator  mean     n std_err .config
##    <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     5     8     7 roc_auc hand_till  0.711     5  0.0230 Preprocessor1_Model~
##  2     8     8     7 roc_auc hand_till  0.705     5  0.0407 Preprocessor1_Model~
##  3     1     5     7 roc_auc hand_till  0.702     5  0.0292 Preprocessor1_Model~
##  4     4     8     5 roc_auc hand_till  0.700     5  0.0426 Preprocessor1_Model~
##  5     4     5     7 roc_auc hand_till  0.697     5  0.0271 Preprocessor1_Model~
##  6     8     7     2 roc_auc hand_till  0.695     5  0.0350 Preprocessor1_Model~
##  7     5     8     3 roc_auc hand_till  0.692     5  0.0118 Preprocessor1_Model~
##  8     6     8     7 roc_auc hand_till  0.691     5  0.0262 Preprocessor1_Model~
##  9     7     4     8 roc_auc hand_till  0.690     5  0.0156 Preprocessor1_Model~
## 10     4     7     4 roc_auc hand_till  0.686     5  0.0420 Preprocessor1_Model~
## # ... with 502 more rows
```

Solution: The roc_auc of your best-performing random forest model on the folds is 0.7114815.

Exercise 8

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

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

rf_final <- finalize_workflow(rf_wf, best_complexity)

rf_final_fit <- fit(rf_final, data = pokemon_train)

rf_final_fit %>%
  pull_workflow_fit() %>%
  vip()
```



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

Solution: sp_atk is most useful and legendary is least useful. Yes, the sp_atk is the most important character of the type of pokemons.

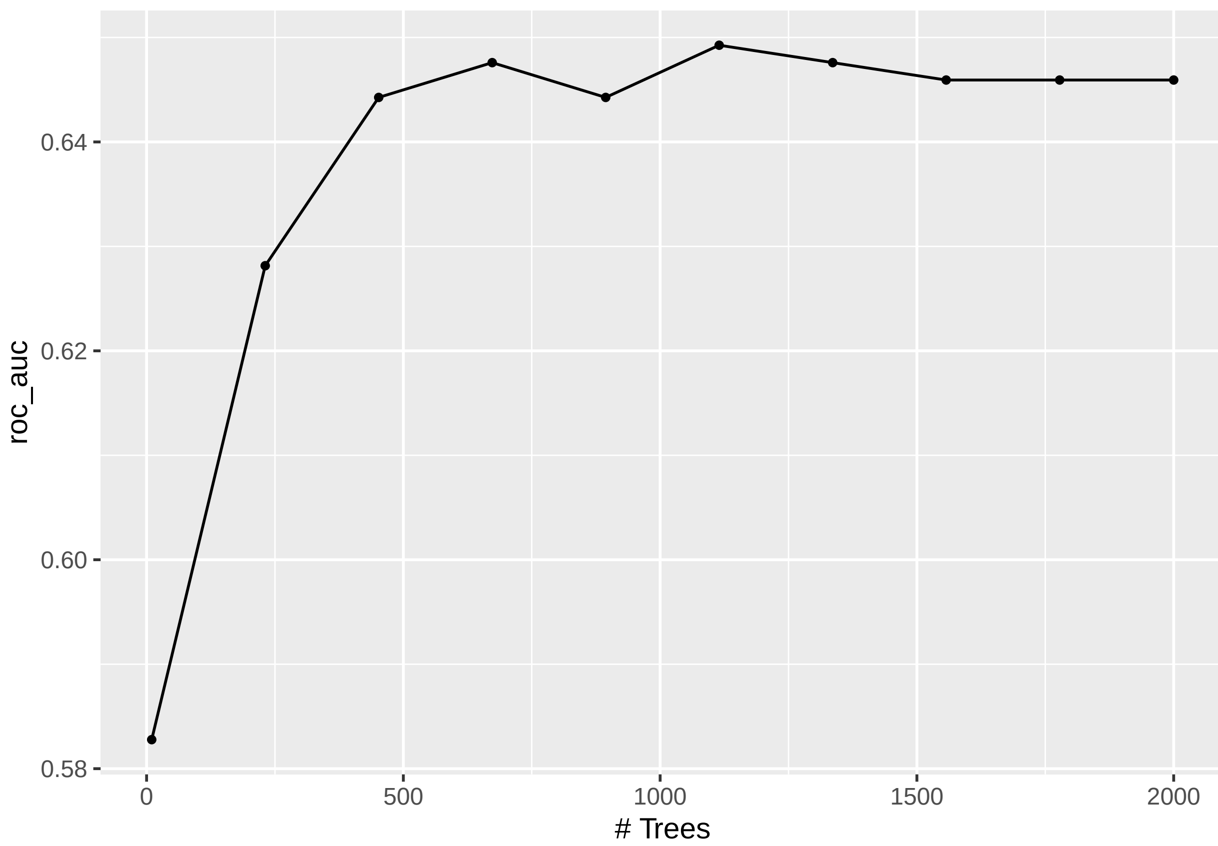
Exercise 9

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?

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() %>%
  set_engine("xgboost") %>%
  set_mode("classification")

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

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

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

autoplot(boost_tune_res)
```




```
collect_metrics(boost_tune_res) %>%
  arrange(desc(mean), by_group = TRUE)
```

```
## # A tibble: 10 x 7
##    trees .metric .estimator  mean     n std_err .config
##    <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
```

```
## 1 1115 roc_auc hand_till 0.649 5 0.0340 Preprocessor1_Model06
## 2 673 roc_auc hand_till 0.648 5 0.0336 Preprocessor1_Model04
## 3 1336 roc_auc hand_till 0.648 5 0.0336 Preprocessor1_Model07
## 4 1557 roc_auc hand_till 0.646 5 0.0337 Preprocessor1_Model08
## 5 1778 roc_auc hand_till 0.646 5 0.0337 Preprocessor1_Model09
## 6 2000 roc_auc hand_till 0.646 5 0.0337 Preprocessor1_Model10
## 7 452 roc_auc hand_till 0.644 5 0.0344 Preprocessor1_Model03
## 8 894 roc_auc hand_till 0.644 5 0.0338 Preprocessor1_Model05
## 9 231 roc_auc hand_till 0.628 5 0.0397 Preprocessor1_Model02
## 10 10 roc_auc hand_till 0.583 5 0.0370 Preprocessor1_Model01
```

Solution: We observe that with the increase of the trees, the roc_auc increases quickly and slow down and last finally. The roc_auc of the best-performing boosted tree model on the folds is 0.6492593.

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?

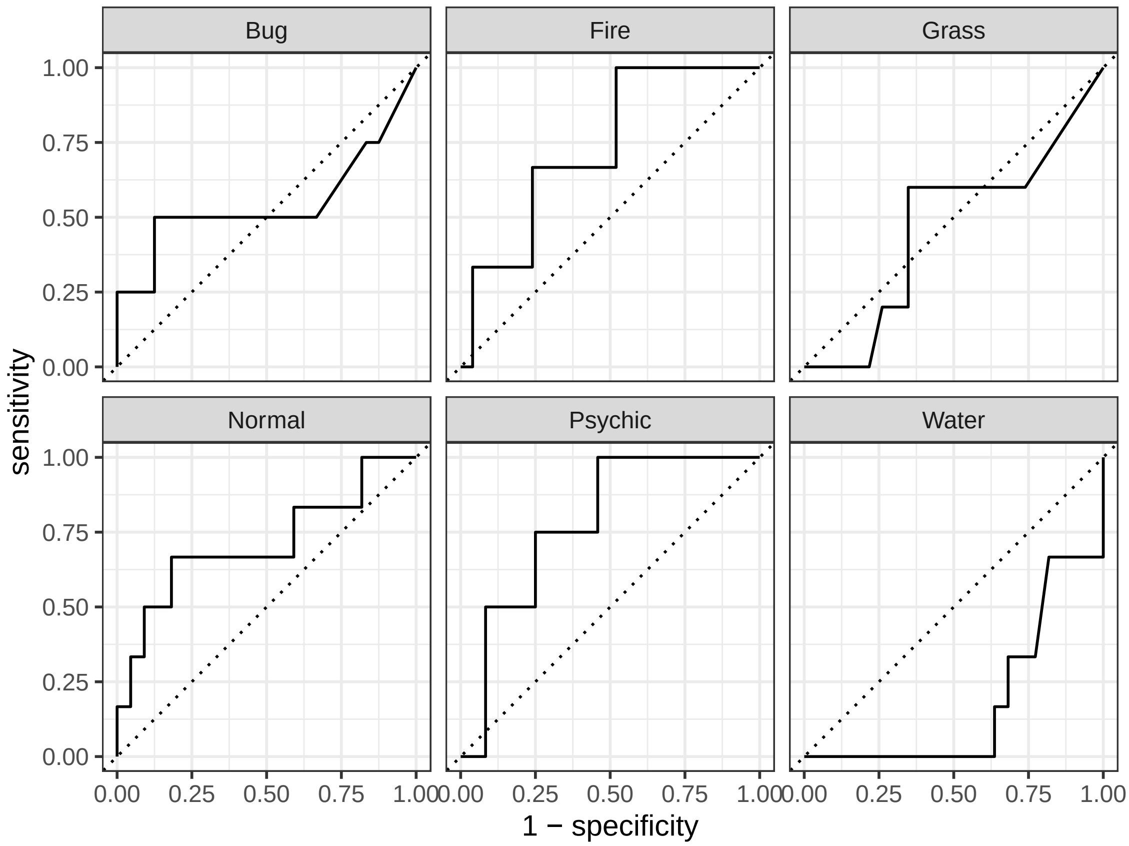
```r
roc_auc_pre <- c(0.6516667, 0.7114815, 0.6492593)
models <- c("best-performing pruned tree model", "random forest model", "boosted tree model")
results <- tibble(roc_auc_pre = roc_auc_pre, models = models)

results %>%
  arrange(-roc_auc_pre)
```

```
## # A tibble: 3 x 2
##   roc_auc_pre models
##         <dbl> <chr>
## 1       0.711 random forest model
## 2       0.652 best-performing pruned tree model
## 3       0.649 boosted tree model
```

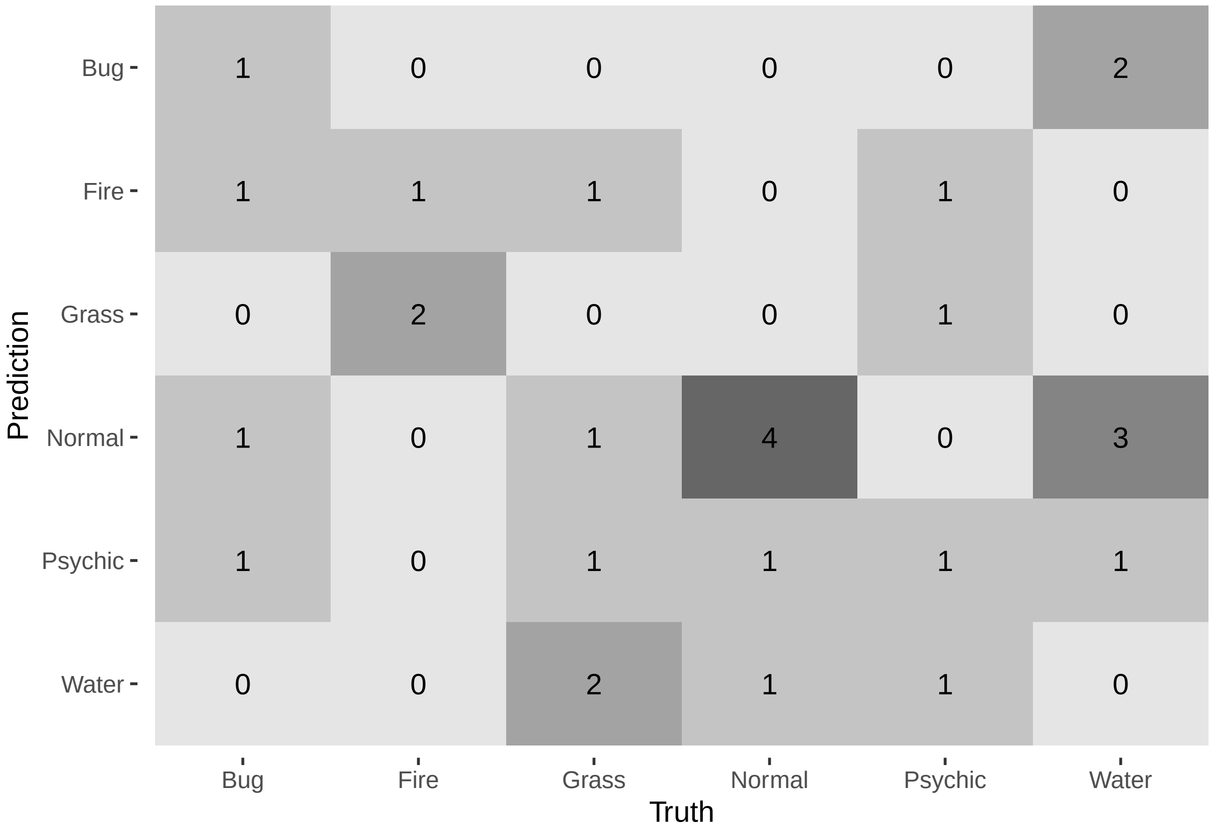
```r
best_complexity <- select_best(rf_tune_res, metric = "roc_auc")

rf_final <- finalize_workflow(rf_wf, best_complexity)

rf_final_fit <- fit(rf_final, data = pokemon_train)

augment(rf_final_fit, new_data = pokemon_test) %>%
  roc_auc(type_1, .pred_Bug:.pred_Water)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.573
```

```r
augment(rf_final_fit, new_data = pokemon_test) %>%
  roc_curve(type_1, .pred_Bug:.pred_Water) %>%
  autoplot()
```




```
augment(rf_final_fit , new_data = pokemon_test) %>%
  conf_mat(type_1, .pred_class) %>%
  autoplot(type = "heatmap")
```



Solution: The random forest model performed best on the folds. The model is best at predicting fire and is worst at predicting grass.