PSTAT 131 HW 6

Raymond Lee

2022-05-19

```
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 0.2.0 --

## v broom        0.8.0     v recipes      0.2.0
## v dials        0.1.1     v rsample      0.1.1
## v dplyr        1.0.8     v tibble       3.1.6
## v ggplot2      3.3.5     v tidyr        1.2.0
## v infer        1.0.0     v tune         0.2.0
## v modeldata    0.1.1     v workflows    0.2.6
## v parsnip      0.2.1     v workflowsets 0.2.1
## v purrr        0.3.4     v yardstick    0.0.9

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x purrr::discard() masks scales::discard()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x recipes::step()  masks stats::step()
## * Learn how to get started at https://www.tidymodels.org/start/
```

```
tidymodels_prefer()
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v readr   2.1.2     v forcats 0.5.1
## v stringr 1.4.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x readr::col_factor() masks scales::col_factor()
## x purrr::discard()    masks scales::discard()
## x dplyr::filter()     masks stats::filter()
## x stringr::fixed()    masks recipes::fixed()
## x dplyr::lag()        masks stats::lag()
## x readr::spec()       masks yardstick::spec()
```

```
library(janitor)
library(rpart.plot)
```

```
## Loading required package: rpart

##
## Attaching package: 'rpart'

## The following object is masked from 'package:dials':
##
##     prune
```

```
library(randomForest)
```

```
## randomForest 4.7-1

## Type rfNews() to see new features/changes/bug fixes.
```

```
library(xgboost)
library(ranger)
library(vip)

pokemon = read.csv("pokemon.csv")
```

1.

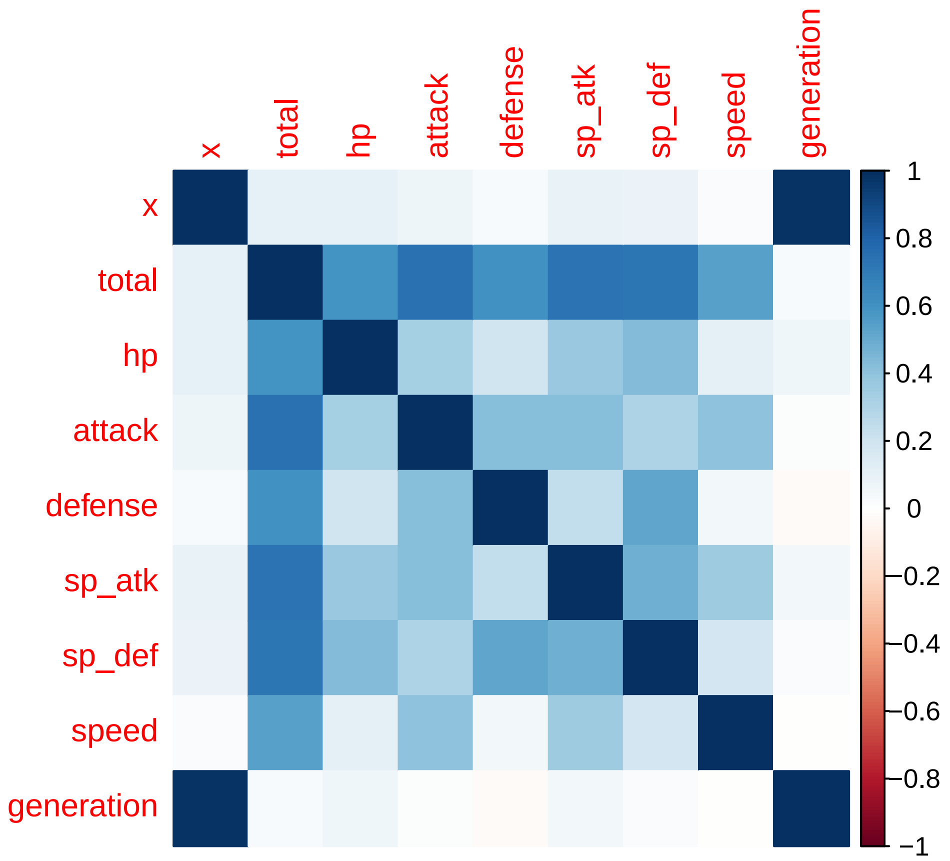
```
pokemon = clean_names(pokemon)
pokemon = filter(pokemon, type_1 == 'Bug' | type_1 == 'Fire' | type_1 == 'Grass' | type_1 == 'Normal' |
                   type_1 == 'Water' | type_1 == 'Psychic')
pokemon$type_1 = factor(pokemon$type_1, levels = c('Bug', 'Fire', 'Grass', 'Normal', 'Water', 'Psychic'
pokemon$legendary = factor(pokemon$legendary, levels = c('True', 'False'))

set.seed(1114)
pokemon_split = initial_split(pokemon, prop = .70, strata = type_1)
pokemon_train = training(pokemon_split)
pokemon_test = testing(pokemon_split)

pokemon_folds = vfold_cv(pokemon_train, v = 5, strata = type_1)

pokemon_recipe = recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense +
                          hp + sp_def, data = pokemon_train) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_predictors())
```

2.

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

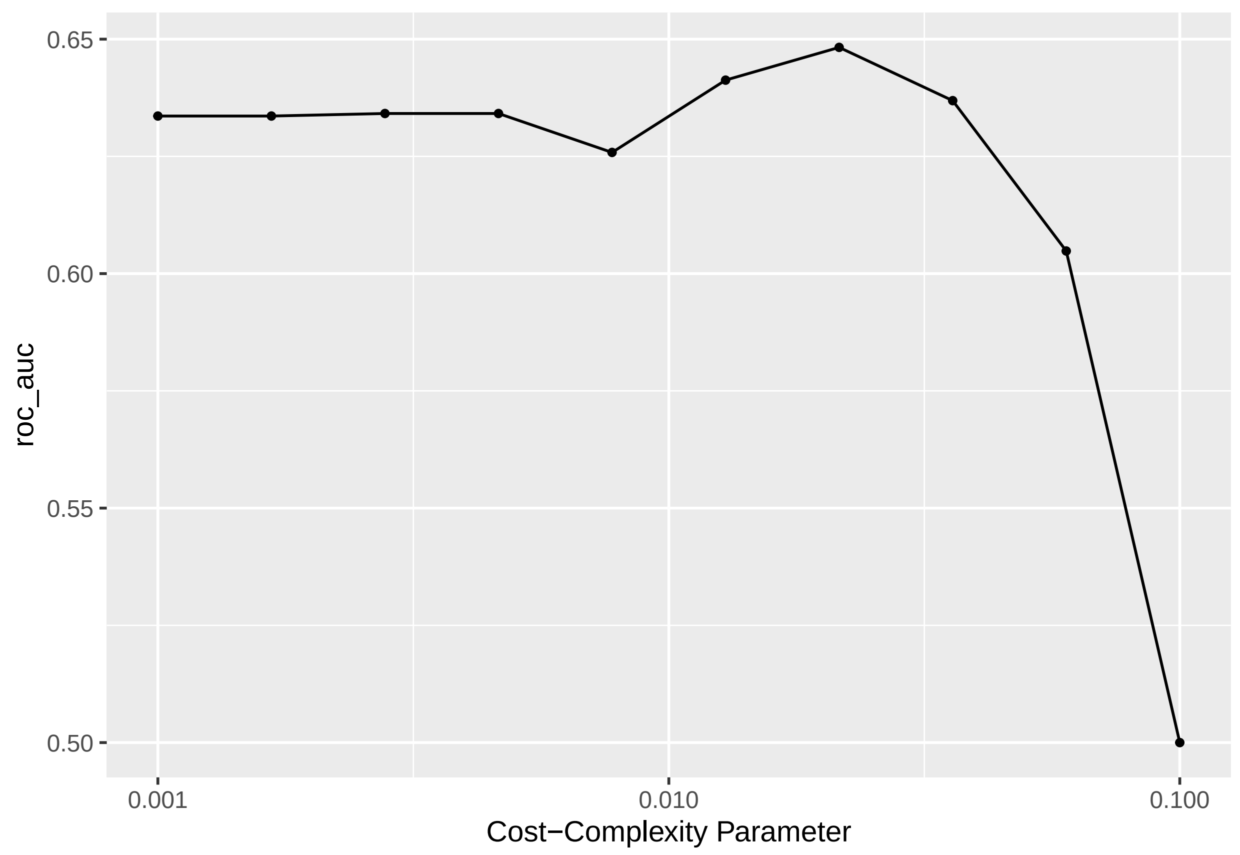
```
corrplot(cor(Filter(is.numeric, pokemon_train)), method = 'color')
```



I made a correlation matrix for all continuous variables. There seems to be fairly strong correlation between total and all of the stats. There also seems to be a fairly strong correlation between sp_def and defense. Generation has a weak correlation with all of the stats. These relationships make sense to me because higher stats would lead to a higher total, and pokemon with good defense will most likely also have good special defense.

3.

```
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)
```



ROC__AUC starts decreasing after little past a complexity penalty of 0.01. A single decision tree performs best with a fairly large complexity penalty of a little past 0.01.

4.

```
tune_res_metrics = collect_metrics(tune_res)
arrange(tune_res_metrics, desc(mean))
```

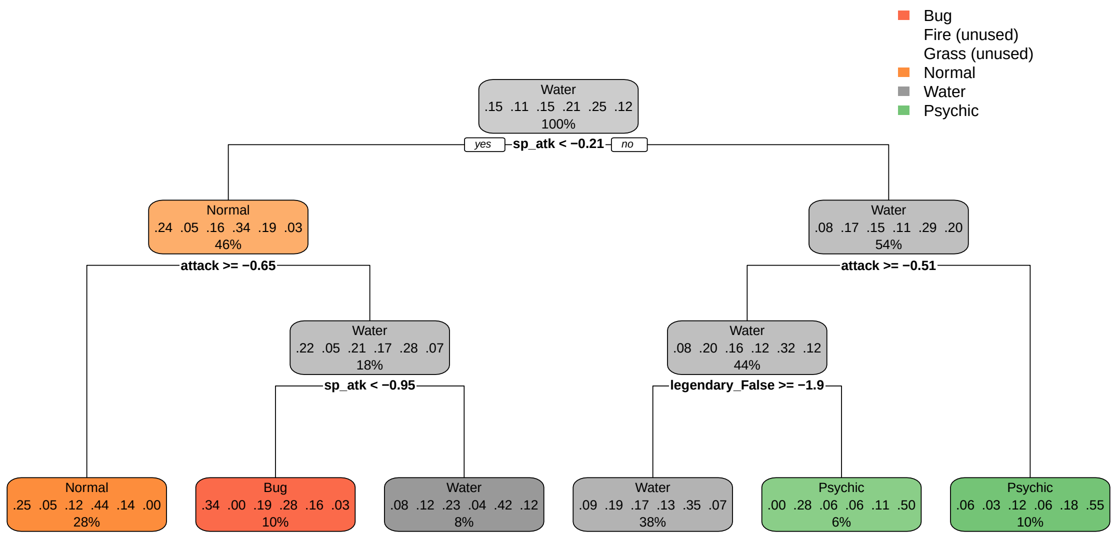
```
## # A tibble: 10 x 7
##    cost_complexity .metric .estimator  mean     n std_err .config
##              <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         0.0215  roc_auc hand_till  0.648     5 0.0127  Preprocessor1_Model07
##  2         0.0129  roc_auc hand_till  0.641     5 0.00868 Preprocessor1_Model06
##  3         0.0359  roc_auc hand_till  0.637     5 0.0193  Preprocessor1_Model08
##  4         0.00278 roc_auc hand_till  0.634     5 0.00839 Preprocessor1_Model03
##  5         0.00464 roc_auc hand_till  0.634     5 0.00839 Preprocessor1_Model04
##  6         0.001   roc_auc hand_till  0.634     5 0.00848 Preprocessor1_Model01
##  7         0.00167 roc_auc hand_till  0.634     5 0.00848 Preprocessor1_Model02
##  8         0.00774 roc_auc hand_till  0.626     5 0.00925 Preprocessor1_Model05
##  9         0.0599  roc_auc hand_till  0.605     5 0.00790 Preprocessor1_Model09
## 10         0.1     roc_auc hand_till  0.5       5 0       Preprocessor1_Model10
```

The best-performing pruned decision tree on the folds has a mean ROC__AUC of 0.6482470.

5.

```
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 = pokemon_train)

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.binar
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.
```



6.

```
rand_forest_spec <- rand_forest(mtry = tune(), trees = tune(), min_n = tune()) %>%
  set_engine("ranger", importance = 'impurity') %>%
  set_mode("classification")
```

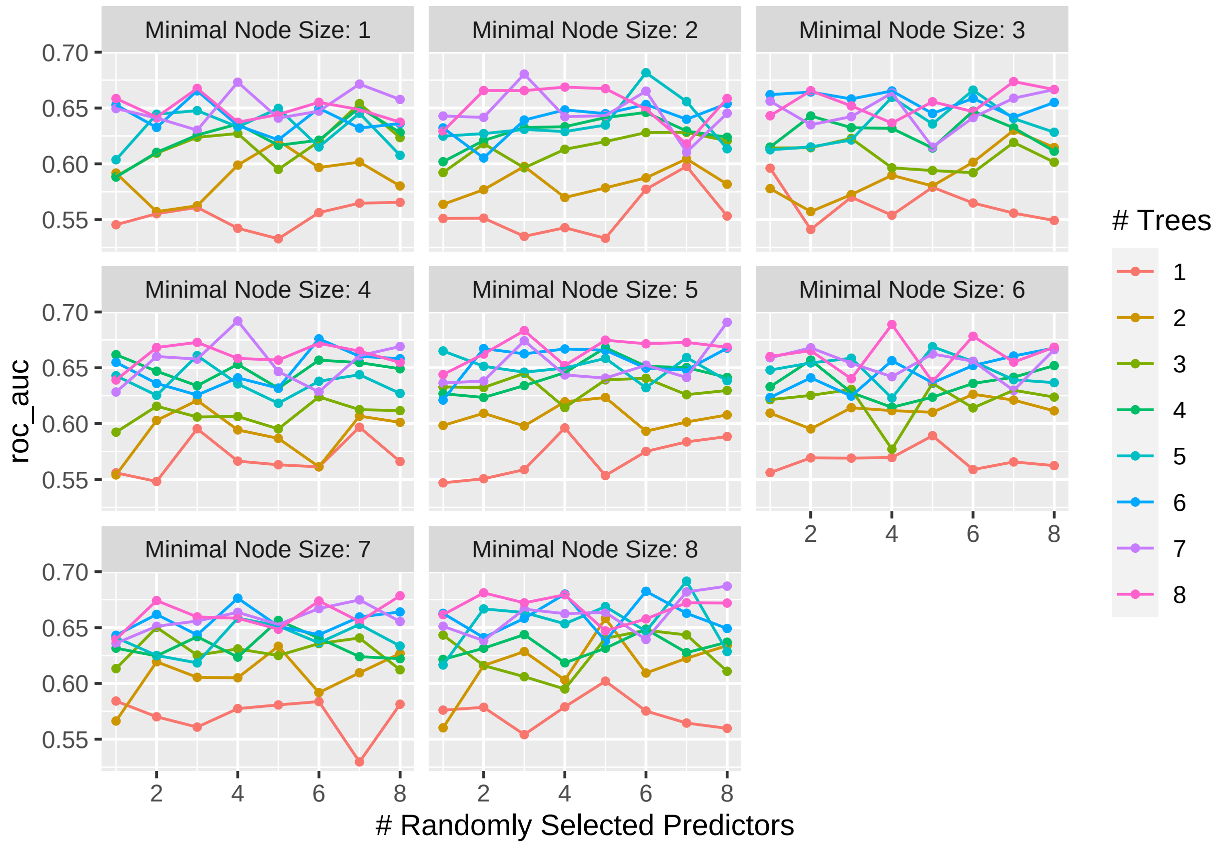
mtry is the number of predictors that will be randomly sampled at each split when creating the tree models. trees number of trees contained in the ensemble. min_n is the minimum number of data points in a node that are required for the node to be split further.

```
rand_forest_grid = grid_regular(mtry(range = c(1, 8)), trees(range = c(1, 8)),
                                min_n(range = c(1, 8)), levels = 8)
```

mtry cannot be greater than 8 because our model has 8 predictors. mtry = 8 represents a bagging model.

7.

```
rand_forest_wf = workflow() %>%
  add_model(rand_forest_spec) %>%
  add_recipe(pokemon_recipe)

rand_forest_tune_res <- tune_grid(
  rand_forest_wf,
  resamples = pokemon_folds,
  grid = rand_forest_grid,
  metrics = metric_set(roc_auc)
)

autoplot(rand_forest_tune_res)
```



8.

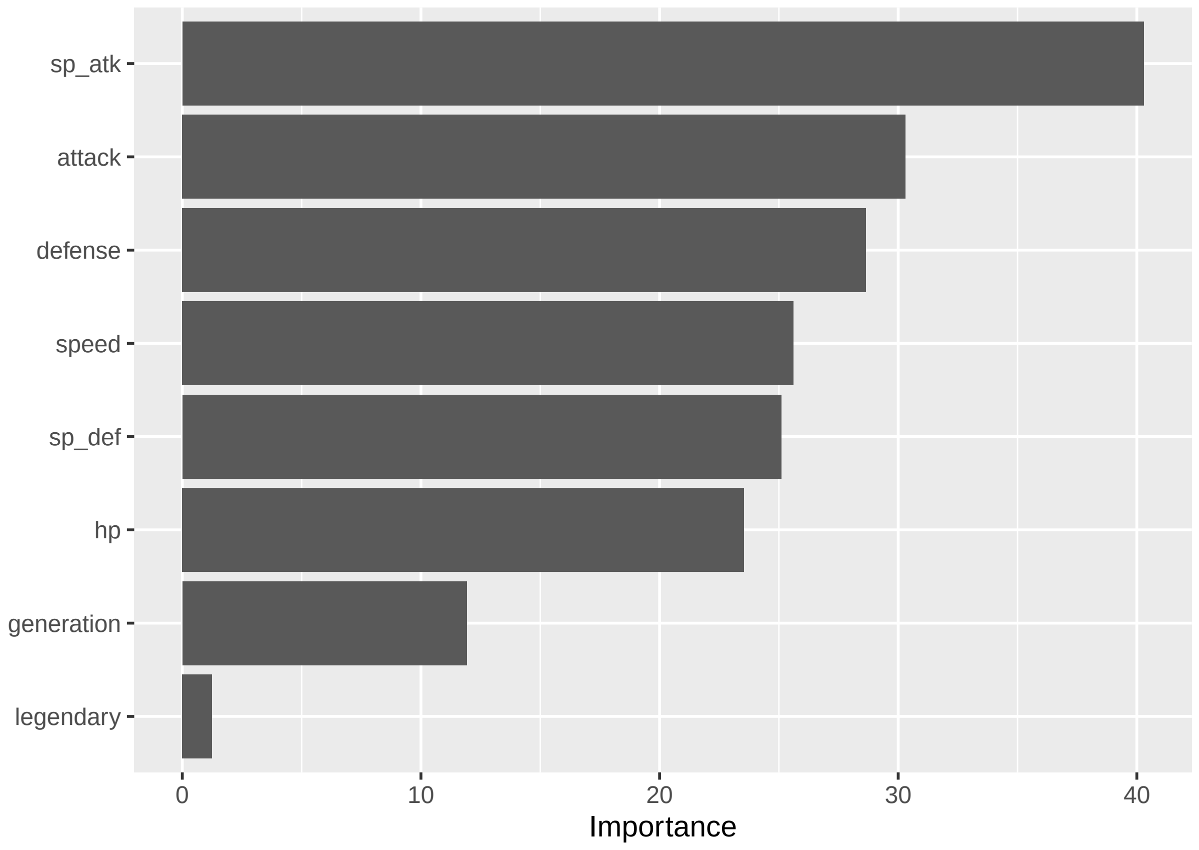
```r
rf_tune_res_metrics = collect_metrics(rand_forest_tune_res)
arrange(rf_tune_res_metrics, desc(mean))
```

```
## # 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     4     7     4 roc_auc hand_till  0.692     5  0.0129 Preprocessor1_Model~
##  2     7     5     8 roc_auc hand_till  0.692     5  0.0114 Preprocessor1_Model~
##  3     8     7     5 roc_auc hand_till  0.691     5  0.0228 Preprocessor1_Model~
##  4     4     8     6 roc_auc hand_till  0.689     5  0.0175 Preprocessor1_Model~
##  5     8     7     8 roc_auc hand_till  0.687     5  0.0192 Preprocessor1_Model~
##  6     3     8     5 roc_auc hand_till  0.683     5  0.0169 Preprocessor1_Model~
##  7     6     6     8 roc_auc hand_till  0.682     5  0.0241 Preprocessor1_Model~
##  8     7     7     8 roc_auc hand_till  0.682     5  0.0132 Preprocessor1_Model~
##  9     6     5     2 roc_auc hand_till  0.682     5  0.0231 Preprocessor1_Model~
## 10     2     8     8 roc_auc hand_till  0.681     5  0.0135 Preprocessor1_Model~
## # ... with 502 more rows
```

The best-performing random forest model on the folds has a mean ROC_AUC of 0.6953591.

9.

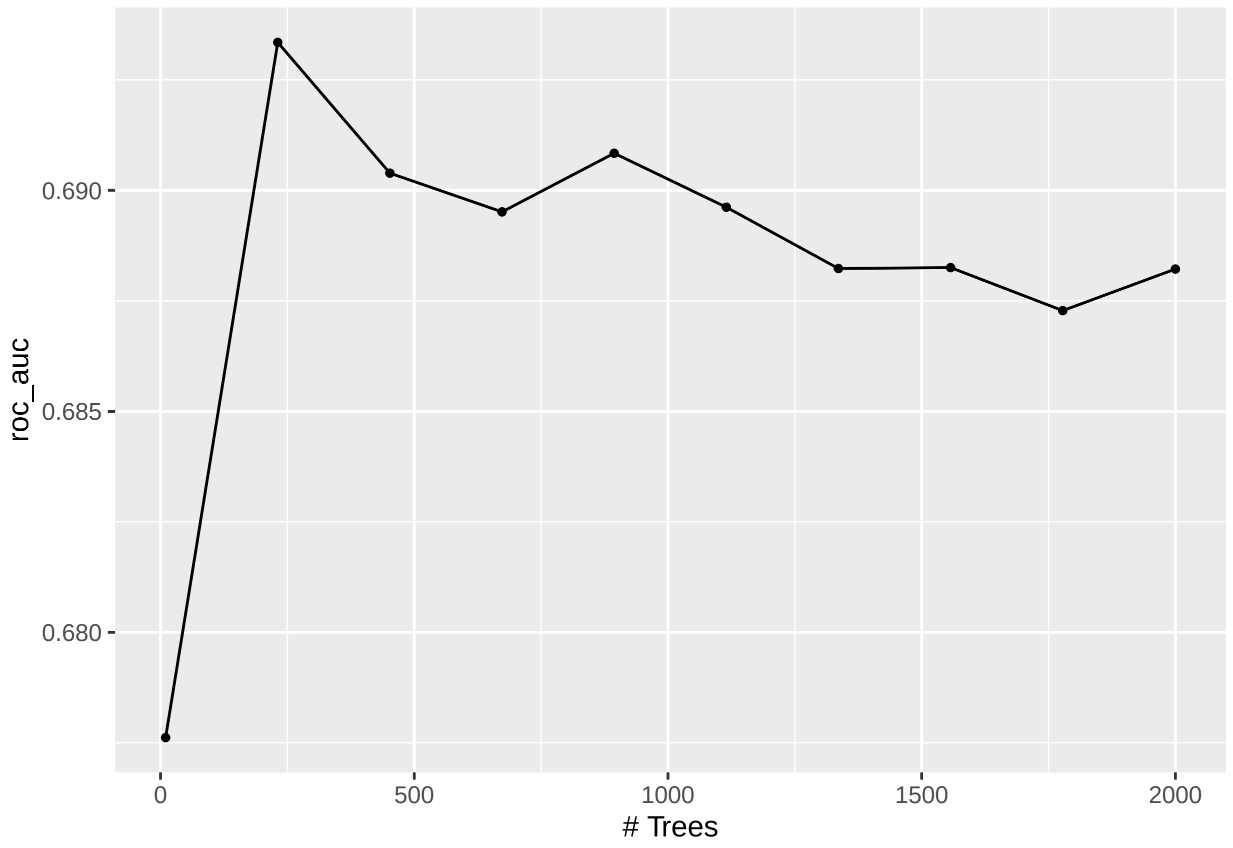
```r
best_rf = select_best(rand_forest_tune_res, metric = 'roc_auc')
best_rand_forest_spec <- rand_forest(mtry = 6, trees = 6, min_n = 8) %>%
  set_engine("ranger", importance = 'impurity') %>%
  set_mode("classification")
best_rf_fit = fit(best_rand_forest_spec, formula = type_1 ~ legendary + generation + sp_atk + attack + 
                          hp + sp_def, data = pokemon_train)
vip(best_rf_fit)
```



sp_atk is the most important, and legendary is the least important. In the last homework, I thought perhaps each pokemon type has certain stats that are significantly different from other types (so each type is distinct), so this does make some sense. This suggests that sp_atk values are most important when determining a pokemon's primary type. It also makes sense that a pokemon's legendary status would not be strongly related to its primary type.

10.

```
boost_spec <- boost_tree(trees = tune(), tree_depth = 8) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

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

boost_wf = workflow() %>%
  add_model(boost_spec) %>%
  add_recipe(pokemon_recipe)

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

autoplot(boost_tune_res)
```



The ROC_AUC is highest at just below 250 trees.

```
boost_tune_res_metrics = collect_metrics(boost_tune_res)
arrange(boost_tune_res_metrics, desc(mean))
```

```
## # A tibble: 10 x 7
##    trees .metric .estimator  mean     n std_err .config
##    <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1   231 roc_auc hand_till  0.693     5  0.0166 Preprocessor1_Model02
##  2   894 roc_auc hand_till  0.691     5  0.0172 Preprocessor1_Model05
##  3   452 roc_auc hand_till  0.690     5  0.0174 Preprocessor1_Model03
##  4  1115 roc_auc hand_till  0.690     5  0.0176 Preprocessor1_Model06
##  5   673 roc_auc hand_till  0.690     5  0.0179 Preprocessor1_Model04
##  6  1557 roc_auc hand_till  0.688     5  0.0178 Preprocessor1_Model08
##  7  1336 roc_auc hand_till  0.688     5  0.0173 Preprocessor1_Model07
##  8  2000 roc_auc hand_till  0.688     5  0.0184 Preprocessor1_Model10
##  9  1778 roc_auc hand_till  0.687     5  0.0183 Preprocessor1_Model09
## 10    10 roc_auc hand_till  0.678     5  0.0209 Preprocessor1_Model01
```

The best-performing boosted tree model on the folds has a mean ROC_AUC of 0.6933468.

11.

```
best_performing_models = c('pruned tree', 'random forest', 'boosted tree')
roc_auc = c(0.6482470, 0.6953591, 0.6933468)
table = data.frame(best_performing_models, roc_auc)
table
```

```
##   best_performing_models   roc_auc
## 1            pruned tree 0.6482470
## 2          random forest 0.6953591
## 3           boosted tree 0.6933468
```

Random forest performed the best on the folds.

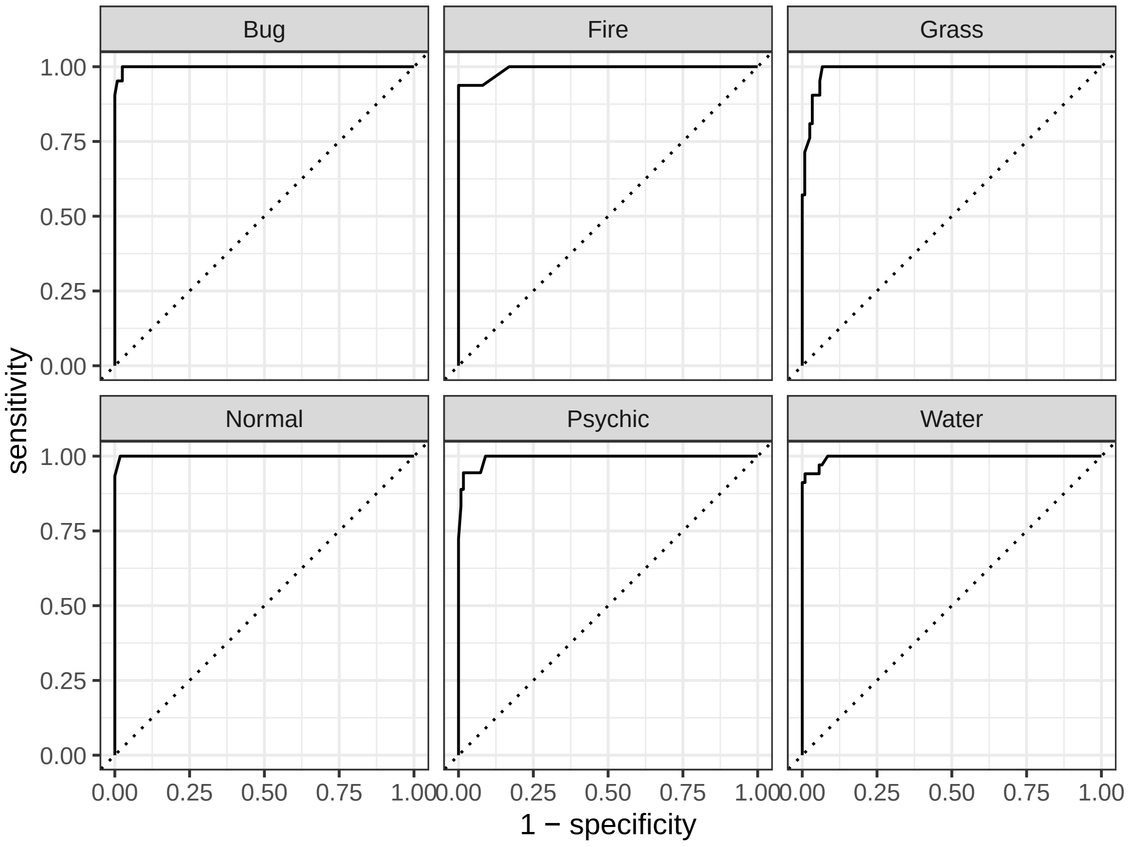
```
rf_final = finalize_workflow(rand_forest_wf, best_rf)
rf_final_fit = fit(rf_final, data = pokemon_test)
```

```
predicted_data = augment(rf_final_fit, new_data = pokemon_test) %>%
  select(type_1, starts_with('.pred'))
predicted_data %>% roc_auc(type_1, .pred_Bug:.pred_Psychic)
```

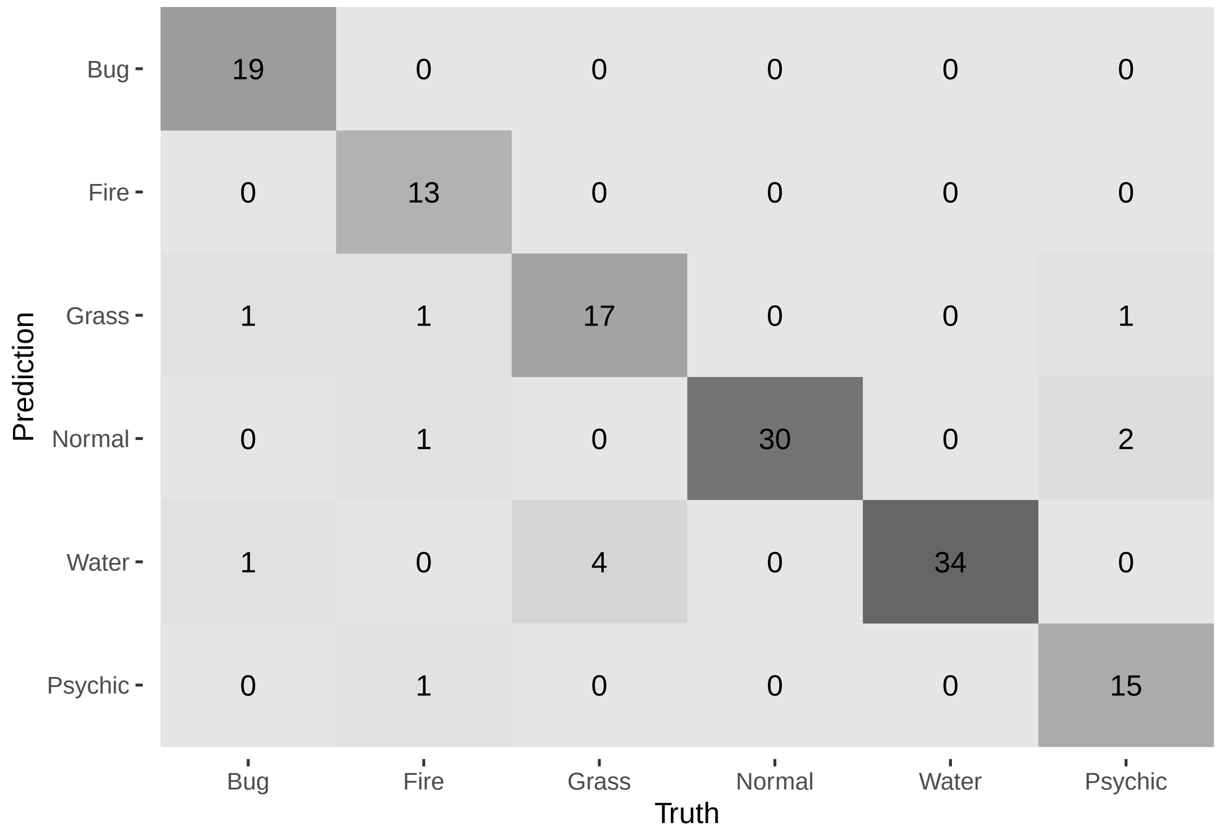
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.994
```

The AUC value of the best performing model on the testing set is 0.9785463.

```
predicted_data %>% roc_curve(type_1, .pred_Bug:.pred_Psychic) %>%
  autoplot()
```



```
predicted_data %>% conf_mat(truth = type_1, estimate = .pred_class) %>%
  autoplot(type = 'heatmap')
```



	Bug	Fire	Grass	Normal	Water	Psychic
Bug	19	0	0	0	0	0
Fire	0	13	0	0	0	0
Grass	1	1	17	0	0	1
Normal	0	1	0	30	0	2
Water	1	0	4	0	34	0
Psychic	0	1	0	0	0	15

The model predicts normal and water types the best. It predicts fire types the worst.