homework 6

Yingshan Li (7937790)

May 21, 2022

Exercise 1

```
pokemon <- read.csv(file = "Pokemon.csv" )
```

```
pokemon1 <- pokemon %>%
  clean_names()
```

```
filtered_pokemon <- pokemon1 %>%
  filter(type_1 == "Bug" | type_1 == "Fire" | type_1 == "Grass" |
           type_1 == "Normal" | type_1 == "Water" | type_1 == "Psychic")
```

```
pokemon2 <- filtered_pokemon %>%
  mutate(type_1 = factor(type_1)) %>%
  mutate(legendary = factor(legendary)) %>%
  mutate(generation = factor(generation))
```

```
set.seed(3435)
pokemon_split <- initial_split(pokemon2, strata = type_1, prop = 0.7)

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

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

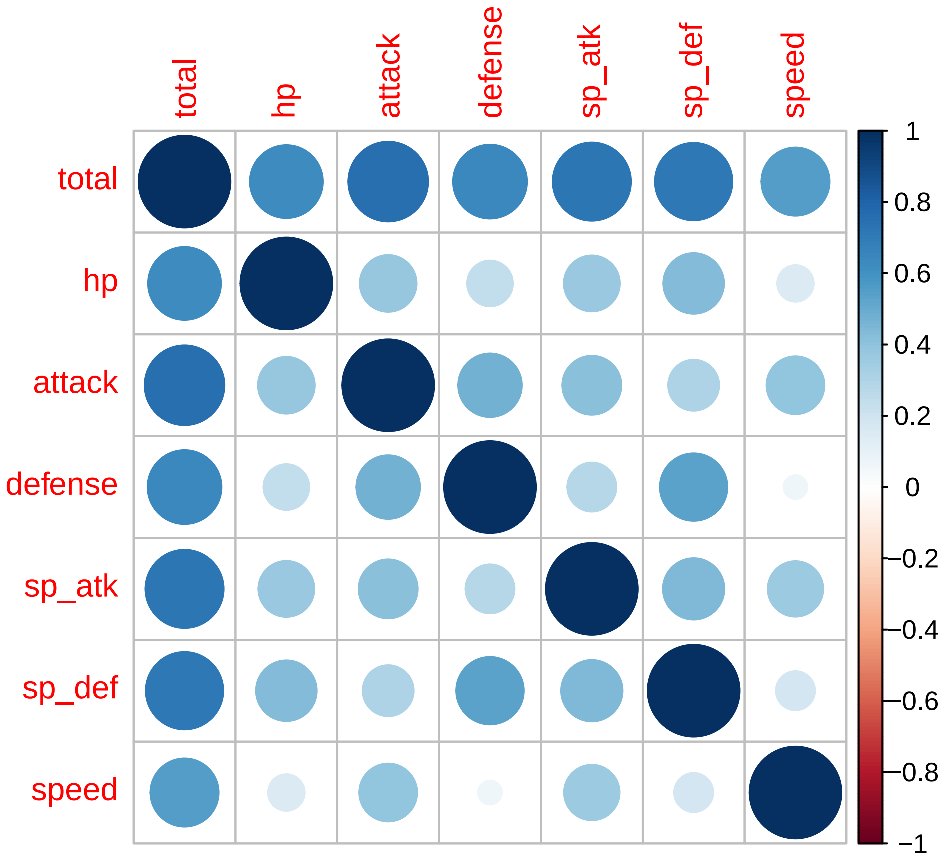
```
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk +
                           attack + speed + defense + hp + sp_def, data = pokemon_train) %>%
  step_dummy(legendary) %>%
  step_dummy(generation) %>%
  step_normalize(all_predictors())
```

Exercise 2

```
pokemon_train %>%
  select(-x) %>%
  select(is.numeric) %>%
  cor() %>%
  corrplot()
```

```
## Warning: Predicate functions must be wrapped in `where()`.
##
##   # Bad
##   data %>% select(is.numeric)
##
##   # Good
##   data %>% select(where(is.numeric))
```

```
##
## i Please update your code.
## This message is displayed once per session.
```



I choose to remove the numeric variable x because this just record the ID number of each pokemon, and I also remove all the categorical variables in the dataset. From this correlation matrix, we can see the total is positively correlated to all the six battle statistics including hp, attack, defense, sp_atk,sp_def, and speed. This makes sense to me because total is the sum of all stats, which represent in general how strong is the pokemon. if the pokemon is outstanding in these separate statistics such as attack or defense, they are stronger and will definitely got a higher number of total.

Exercise 3

```
tree_spec <- decision_tree() %>%
  set_engine("rpart") %>%
  set_mode("classification") %>%
  set_args(cost_complexity = tune())
```

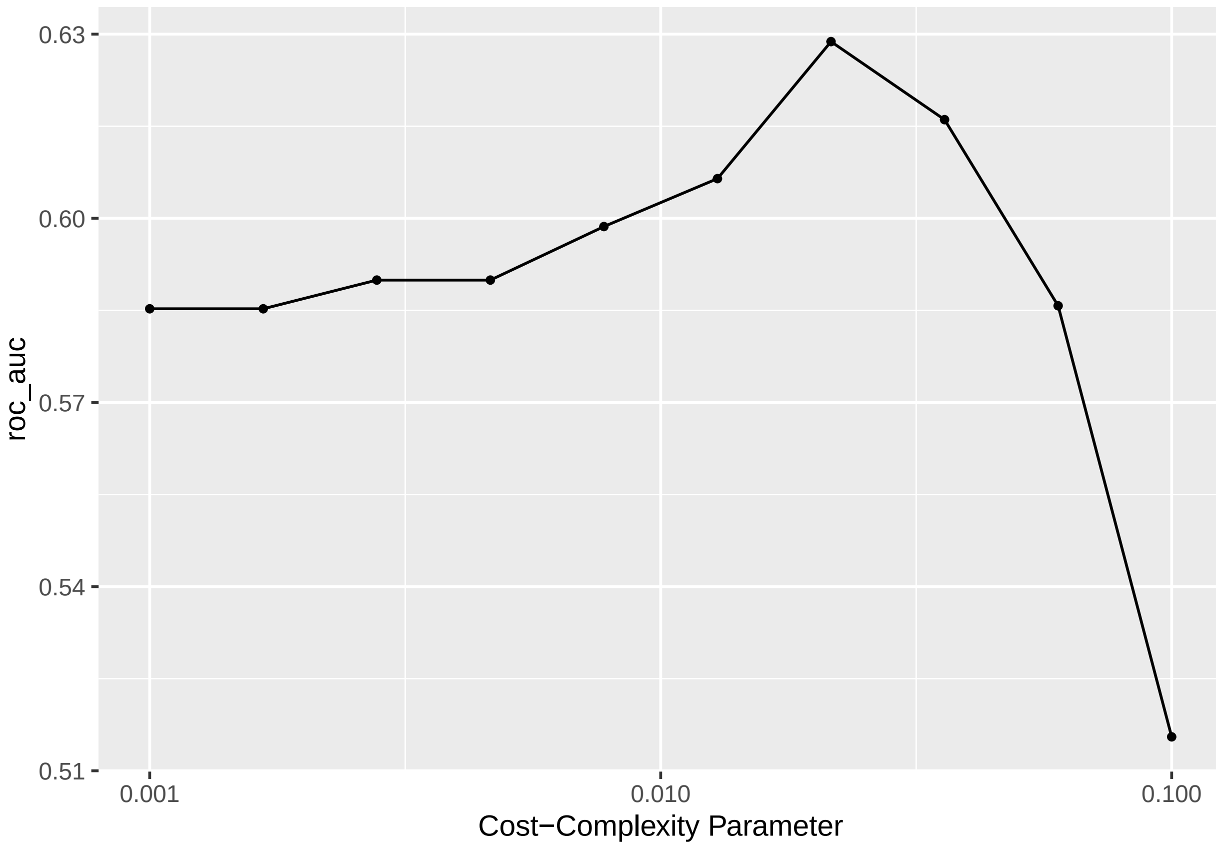
```
tree_wf <- workflow() %>%
  add_model(tree_spec) %>%
  add_recipe(pokemon_recipe)
```

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

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

```
)
```

```
autoplot(tune_res)
```



The general trend of roc_auc increases as the Cost_Complexity increases, so it performs better with larger complexity penalty. After the roc_auc reach the peak, if we further increase the Cost_Complexity, the roc_auc will drop drastically.

Exercise 4

```
tree_roc_auc <- collect_metrics(tune_res) %>%
  arrange(-mean) %>%
  filter(row_number()==1)
tree_roc_auc
```

```
## # A tibble: 1 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  0.629     5  0.0191 Preprocessor1_Model07
```

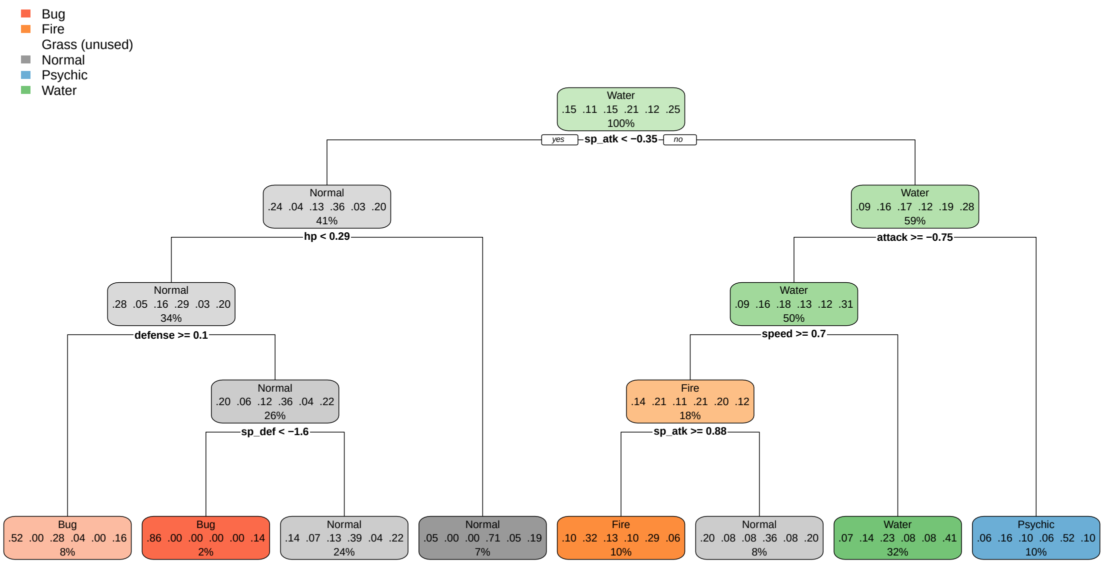
The roc_auc of the best-performing pruned decision tree on the folds is 0.6287888.

Exercise 5

```
best_complexity <- select_best(tune_res)
tree_final <- finalize_workflow(tree_wf, best_complexity)
tree_final_fit <- fit(tree_final, data = pokemon_train)
```

```
tree_final_fit %>%
  extract_fit_engine() %>%
```

```r
rpart.plot()
```



Exercise 5

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

```r
rf_wf <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(rf_spec)
```

```r
rfp_grid <- grid_regular(mtry(range = c(2,7)), trees(range = c(10,2000)), min_n(range = c(2, 10)), leve
```

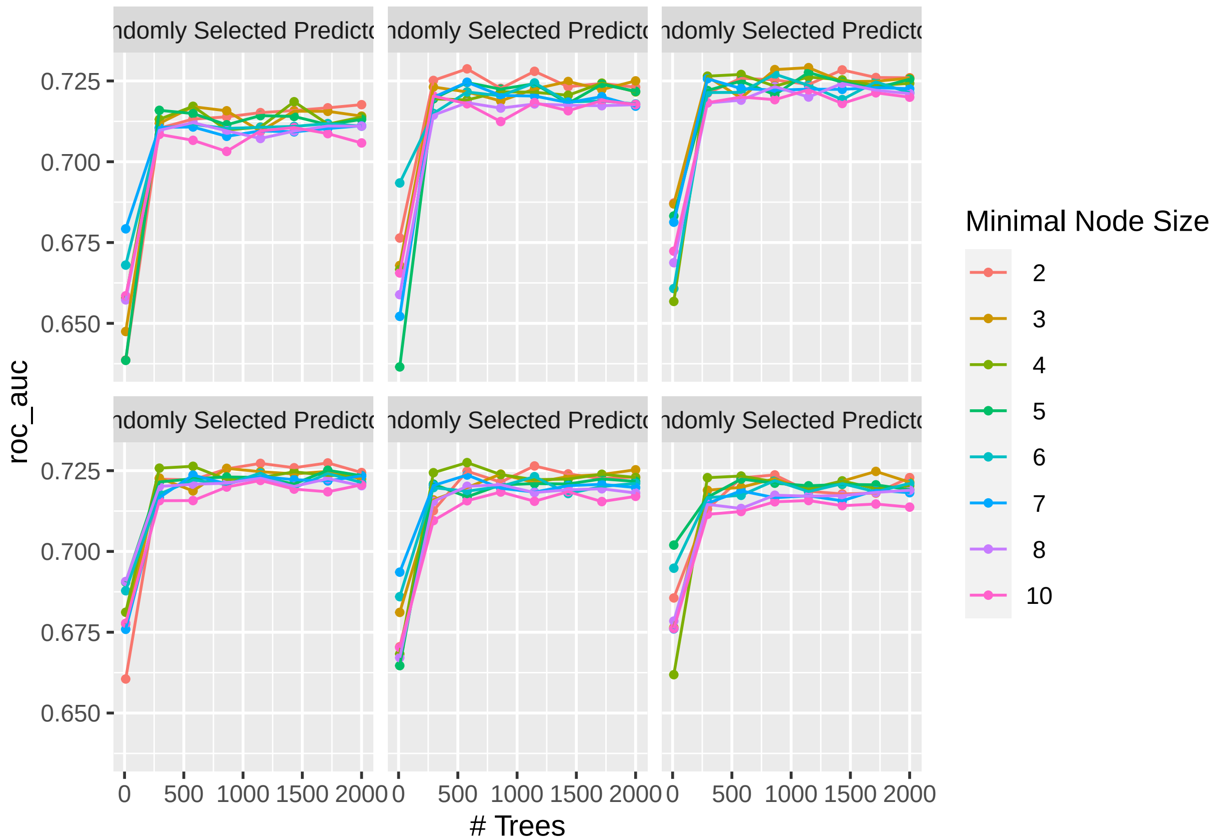
mtry: The number of predictors that will be randomly sampled with each split of the tree models. trees: The number of trees in the ensemble tree models. min_n: minimum number of data points in a node required to make a split.

mtry should not be smaller than 1 or larger than 8, because the model we specify have a total of 8 predictors. If we set the mtry = 8, it represent the bagging model

Exercise 6

```r
tune_rf <- tune_grid(
  rf_wf,
  resamples = pokemon_folds,
  grid = rfp_grid,
  metrics = metric_set(roc_auc)
)
```

```r
autoplot(tune_rf)
```



From the plot we can observe the smaller node size has better performance. The roc-auc increases as the number of trees increase to around 250, and it will not increase significantly anymore if we further increase the number of trees. There are not many difference in roc_auc between the number of randomly selected predictors, 3, 4, or 5 randomly selected predictors seems perform slightly better.

Exercise 7

```
rf_roc_auc <- collect_metrics(tune_rf) %>%
  arrange(-mean) %>%
  filter(row_number()==1)
rf_roc_auc
```

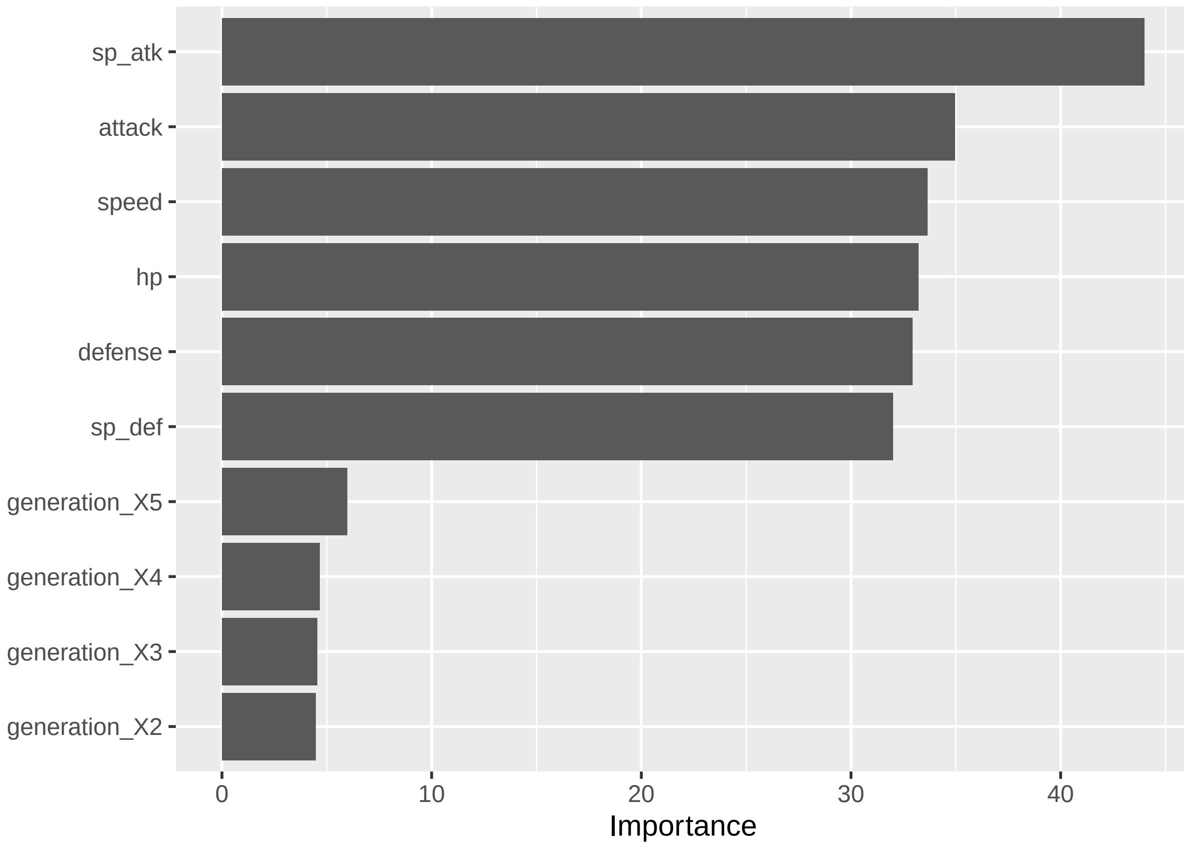
```
## # A tibble: 1 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     4  1147     3 roc_auc hand_till  0.729     5  0.0139 Preprocessor1_Model0~
```

The roc_auc of my best-performing random forest model is 0.7295976.

Exercise 8

```
best_rf_parameters <- select_best(tune_rf, metric = "roc_auc")
rf_final <- finalize_workflow(rf_wf, best_rf_parameters)
rf_final_fit <- fit(rf_final, data = pokemon_train)
```

```
rf_final_fit %>%
  extract_fit_engine() %>%
  vip()
```




```
rf_final_fit %>%
  extract_fit_engine() %>%
  vip::vi() %>%
  arrange(Importance)
```

```
## # A tibble: 12 x 2
##    Variable       Importance
##    <chr>               <dbl>
##  1 legendary_True       1.86
##  2 generation_X6        3.09
##  3 generation_X2        4.48
##  4 generation_X3        4.55
##  5 generation_X4        4.65
##  6 generation_X5        5.98
##  7 sp_def              32.0
##  8 defense             32.9
##  9 hp                  33.2
## 10 speed               33.7
## 11 attack              35.0
## 12 sp_atk              44.0
```

The most useful variable is sp_attack, and the least useful variable is legendary_True. Other than sp_attack, the variables attack, speed, hp , defense, and sp_def are all shown some levels of importance. This is not an unexpected result since the type_1 of the pokemon determines the weakness or resistance to attacks.

Exercise 9

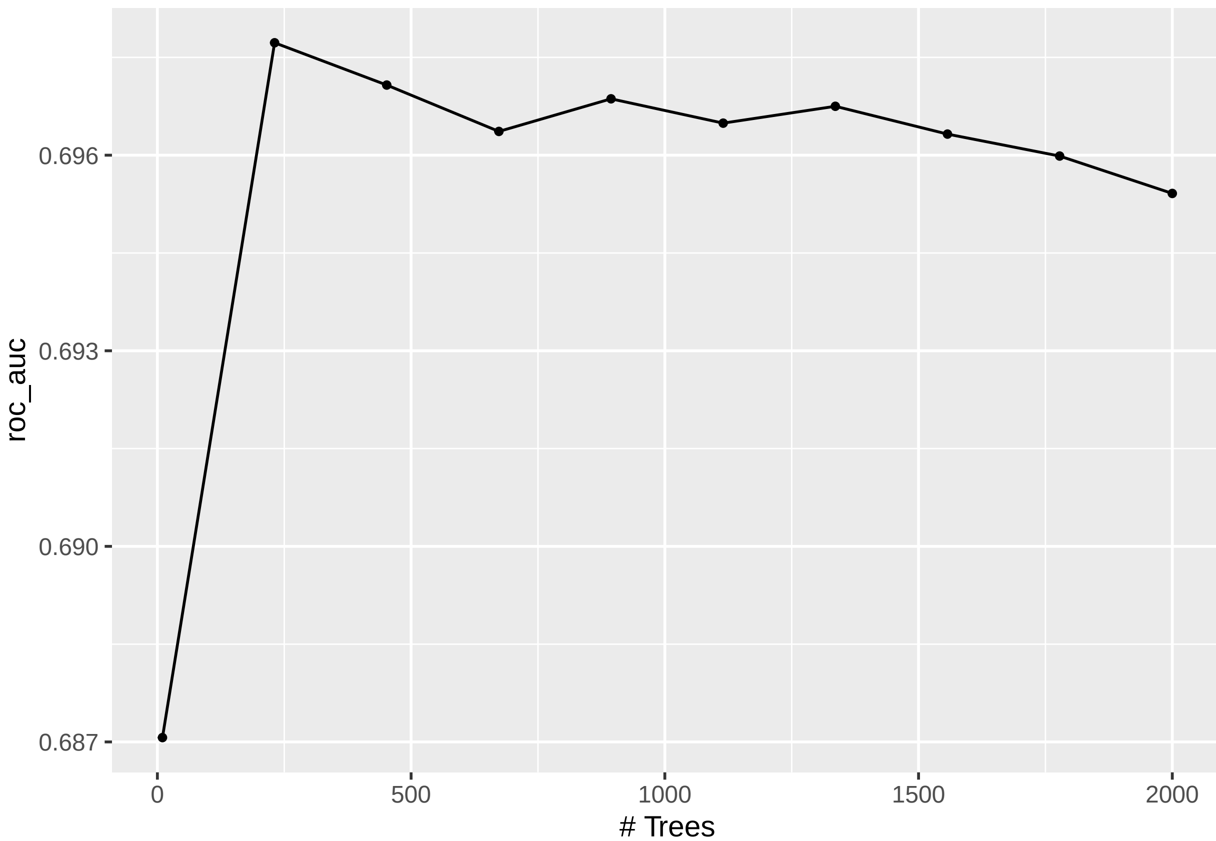
```
boost_spec <- boost_tree(trees = tune()) %>%
  set_engine("xgboost") %>%
```

```
  set_mode("classification")

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

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

```
tune_bt <- tune_grid(bt_wf,resamples = pokemon_folds,grid = t_grid, metrics = metric_set(roc_auc))
```

```
autoplot(tune_bt)
```



The roc_auc increases and reach the highest point when we increase to around 250 trees, then the roc_auc show a gradually decreasing trend.

```
boost_roc_auc <- collect_metrics(tune_bt) %>%
  arrange(-mean) %>%
  filter(row_number()==1)
boost_roc_auc
```

```
## # A tibble: 1 x 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
```

The roc_auc of the best-performing model is 0.6977248.

Exercise 10

```r
roc_auc <- c(tree_roc_auc$mean, rf_roc_auc$mean, boost_roc_auc$mean)
models <- c("Decision Tree", "Random Forest", "Boosted Tree")
results <- tibble(roc_auc = roc_auc, models = models)
results %>%
  arrange(-roc_auc)
```

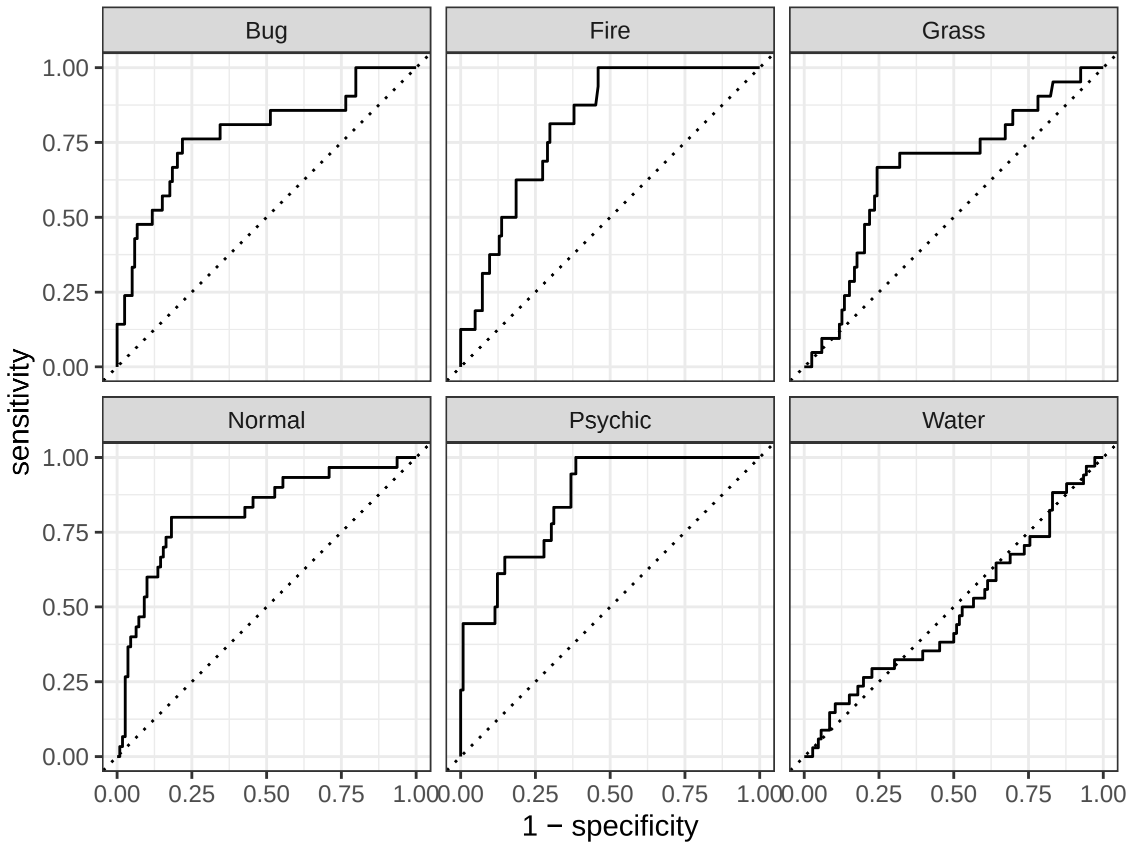
```
## # A tibble: 3 x 2
##   roc_auc models
##     <dbl> <chr>
## 1   0.729 Random Forest
## 2   0.698 Boosted Tree
## 3   0.629 Decision Tree
```

```r
best_rf_parameters <- select_best(tune_rf, metric = "roc_auc")
rf_final_test <- finalize_workflow(rf_wf, best_rf_parameters)
rf_final_testfit <- fit(rf_final_test, data = pokemon_test)
```

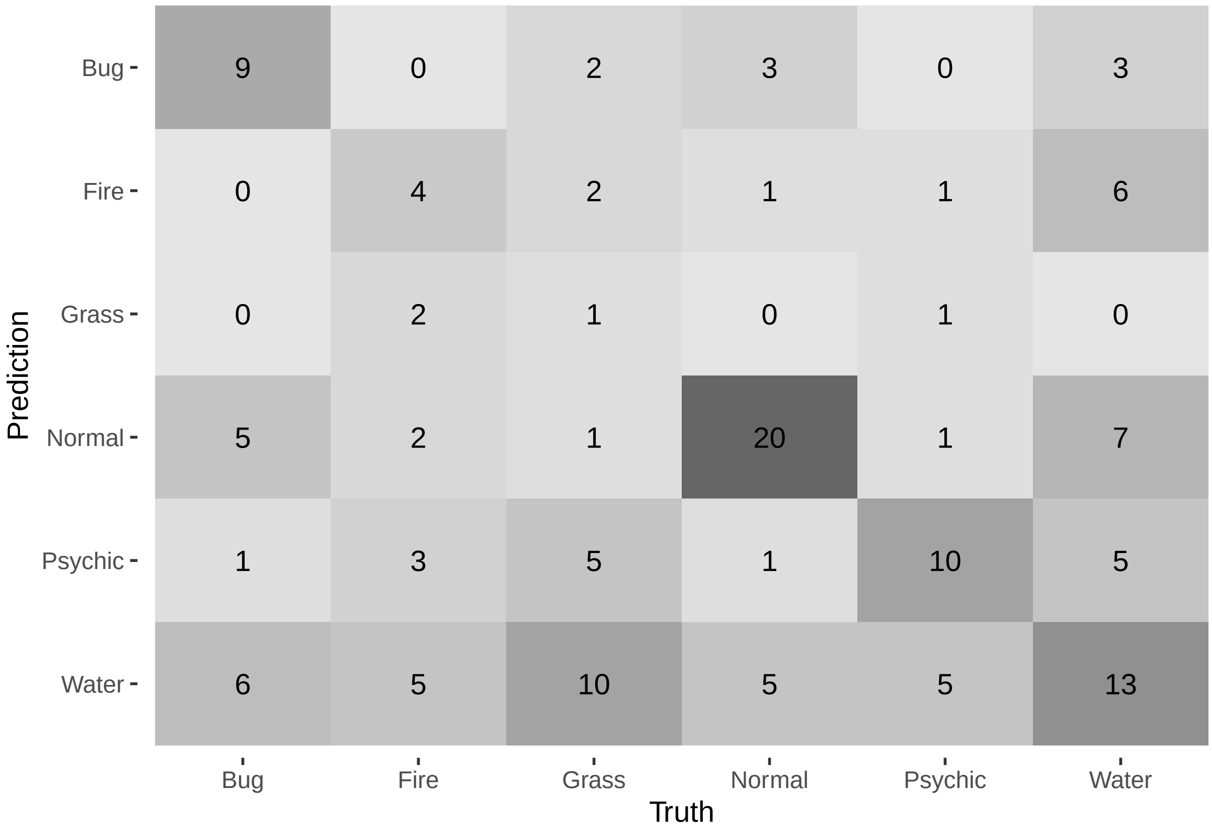
```r
prediction_result <- augment(rf_final_fit, new_data = pokemon_test) %>%
    select(type_1, .pred_class, .pred_Bug, .pred_Fire, .pred_Grass,
           .pred_Normal, .pred_Psychic, .pred_Water)

roc_aunp(prediction_result, type_1, .pred_Bug:.pred_Water)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 roc_aunp macro          0.712
```

```r
prediction_result %>%
  roc_curve(type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal,
            .pred_Psychic, .pred_Water) %>%
  autoplot()
```




```
prediction_result %>%
  conf_mat(type_1, .pred_class) %>%
  autoplot(type = "heatmap")
```



The model most accurate at predicting the class Normal and Bug, but worst at predicting fire. The model is also not predict very well for the class Water.