Homework06

Contents


```
set.seed(231)
pokemon = read.csv('data/Pokemon.csv')
```

Exercise 1

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

pokemon = pokemon[pokemon$type_1 == 'Bug' | pokemon$type_1 == 'Fire' | pokemon$type_1 == 'Grass' | poker

pokemon$type_1 = as.factor(pokemon$type_1)
pokemon$legendary = as.factor(pokemon$legendary)
pokemon$generation = as.factor(pokemon$generation)
```

```
pokemon_split = initial_split(pokemon, prop = 0.70, strata = type_1)

pokemon_train = training(pokemon_split)
pokemon_test = testing(pokemon_split)
```

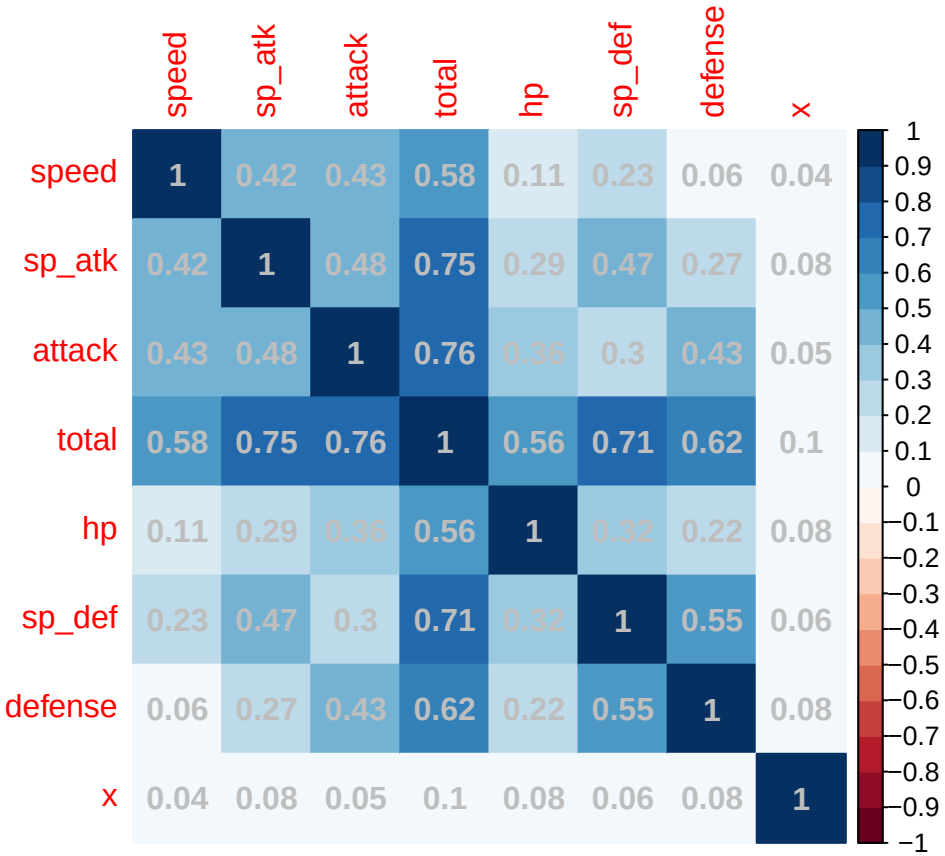
```
pokemon_folds <- vfold_cv(pokemon_train, v = 5, strata = type_1)
```

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

Exercise 2

```
M = cor(select_if(pokemon_train, is.numeric))
corrplot(M, method = 'color', col = COL2(n=20), cl.length = 21, order = 'AOE',
```

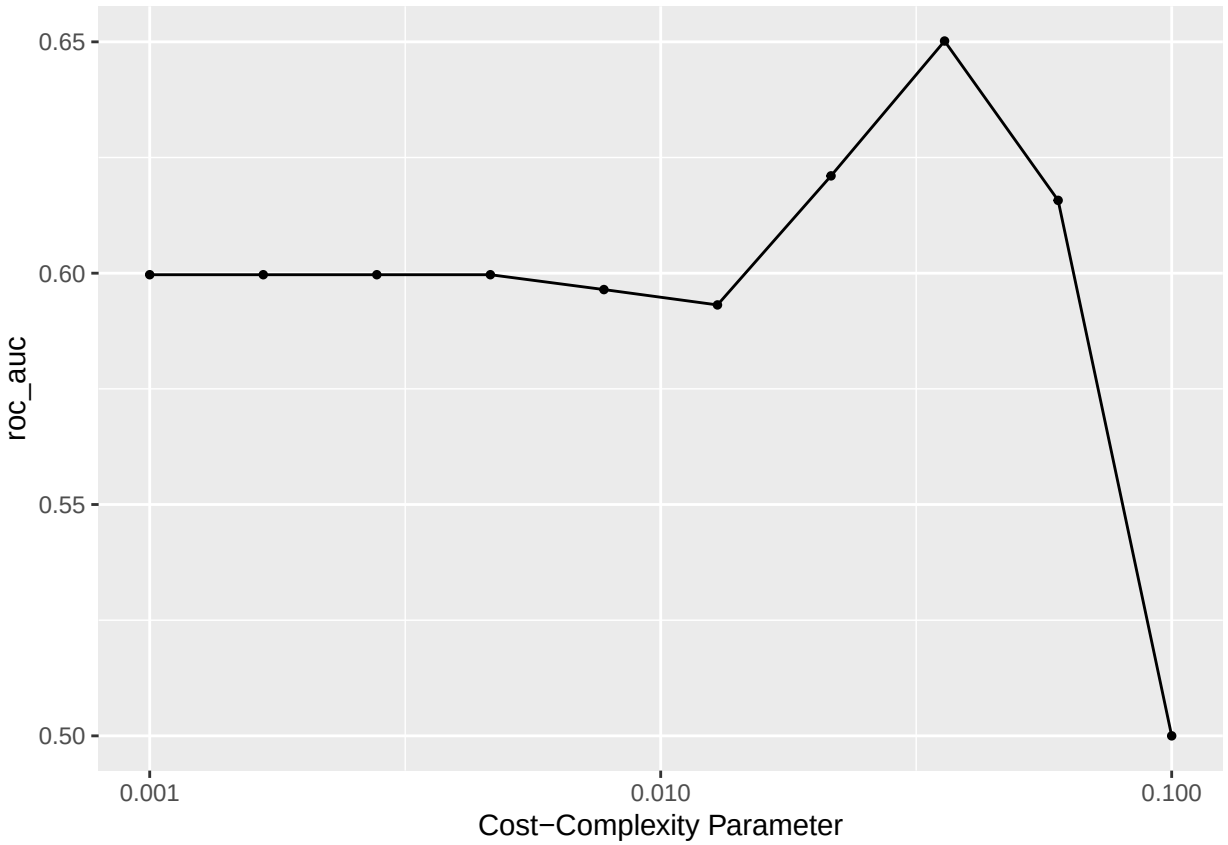
```
        addCoef.col = 'grey')
```



'total' variable has a positive relationship with other variables.

Exercise 3

```
tree_spec = decision_tree(cost_complexity = tune()) %>%
        set_engine('rpart') %>%
        set_mode('classification')

tree_wf = workflow() %>%
        add_model(tree_spec) %>%
        add_recipe(pokemon_recipe)

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

tree_tune <- tune_grid(
        tree_wf,
        resamples = pokemon_folds,
        grid = tree_grid,
        metrics = metric_set(roc_auc))

autoplot(tree_tune)
```



As complexity getting bigger, roc_auc getting higher. However, roc_auc goes down when complexity is close to 0.1.

Exercise 4

```
tree_tune %>% collect_metrics() %>% arrange(desc(mean))
```

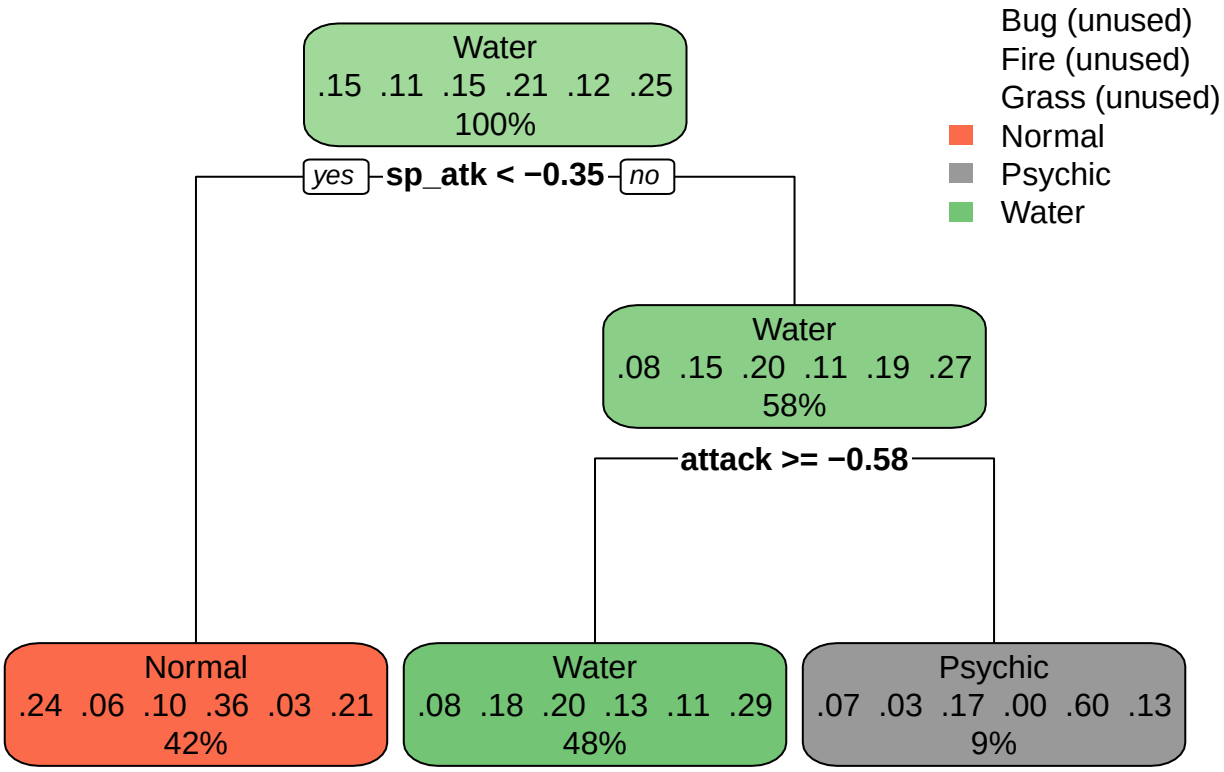
```
## # A tibble: 10 x 7
##    cost_complexity .metric .estimator  mean     n std_err .config
##              <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         0.0359  roc_auc hand_till  0.650     5  0.0195 Preprocessor1_Model08
##  2         0.0215  roc_auc hand_till  0.621     5  0.0302 Preprocessor1_Model07
##  3         0.0599  roc_auc hand_till  0.616     5  0.0142 Preprocessor1_Model09
##  4         0.001   roc_auc hand_till  0.600     5  0.0233 Preprocessor1_Model01
##  5         0.00167 roc_auc hand_till  0.600     5  0.0233 Preprocessor1_Model02
##  6         0.00278 roc_auc hand_till  0.600     5  0.0233 Preprocessor1_Model03
##  7         0.00464 roc_auc hand_till  0.600     5  0.0233 Preprocessor1_Model04
##  8         0.00774 roc_auc hand_till  0.596     5  0.0222 Preprocessor1_Model05
##  9         0.0129  roc_auc hand_till  0.593     5  0.0222 Preprocessor1_Model06
## 10         0.1     roc_auc hand_till  0.5       5  0      Preprocessor1_Model10
```

0.62 is the highest roc_auc.

Exercise 5

```
tree_best = tree_tune %>% select_best(metric = 'roc_auc')
tree_final = tree_wf %>% finalize_workflow(tree_best)
```

```
tree_final_fit = tree_final %>% fit(pokemon_train)

rpart.plot(extract_fit_engine(tree_final_fit), roundint=FALSE)
```



Exercise 6

```
rf_spec = rand_forest(mtry = tune(), trees = tune(), min_n = tune()) %>%
        set_engine('ranger', importance = 'impurity') %>%
        set_mode('classification')
```

'mtry' represents the number of selected variables which we give to each tree to make decision.
'tree' represents the number of total trees.
'min_n' represents the minimum number of data points in each node to split a brunch.

```
rf_wf = workflow() %>%
        add_recipe(pokemon_recipe) %>%
        add_model(rf_spec)
```

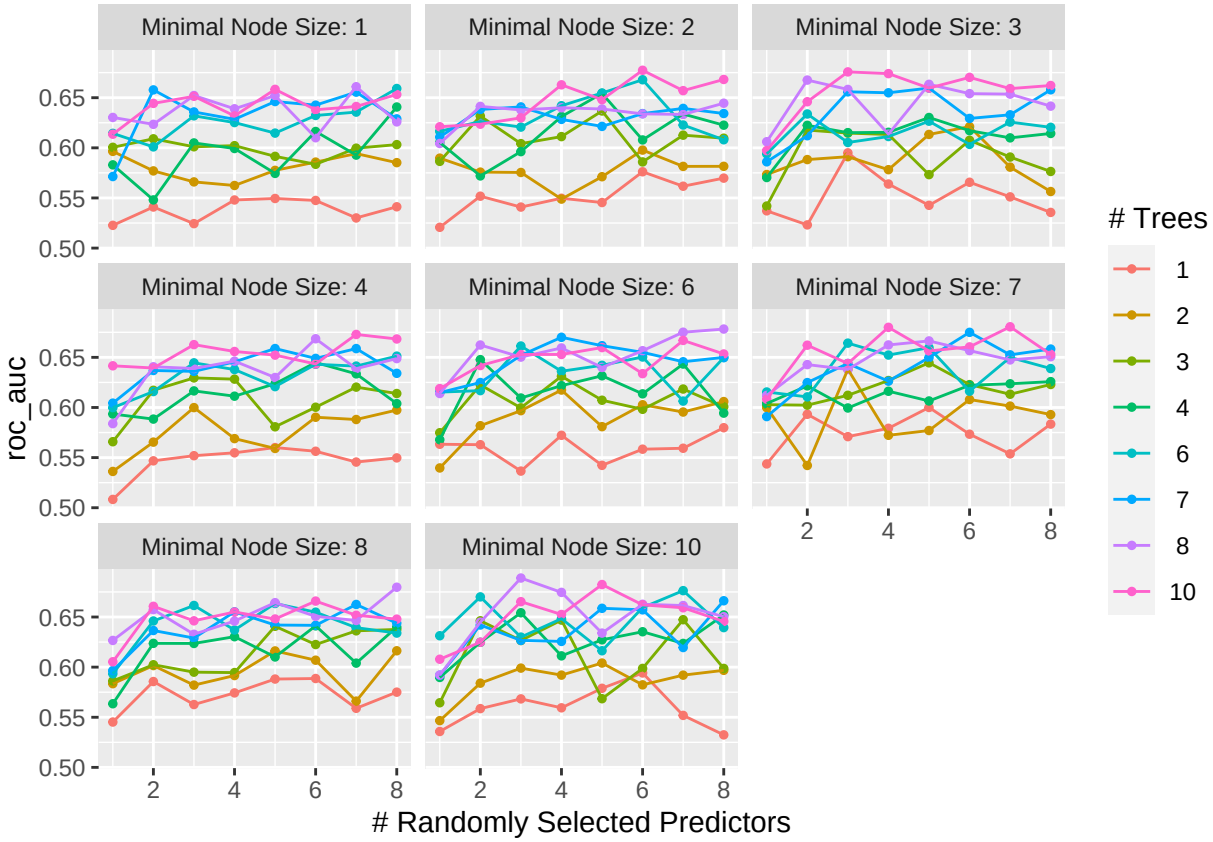
```
rf_grid = grid_regular(mtry(range = c(1,8)),
                       trees(range = c(1,10)),
                       min_n(range = c(1,10)),
                       levels = 8)
```

There are 8 predictors in the dataset. So, select more than 8 variables could not work.

Exercise 7

```
rf_tune = tune_grid(
        rf_wf,
```

```
        resamples = pokemon_folds,
        grid = rf_grid,
        metrics = metric_set(roc_auc))

autoplot(rf_tune)
```




```
rf_best = rf_tune %>% select_best(metric = 'roc_auc')
rf_best
```

```
## # A tibble: 1 x 4
##    mtry trees min_n .config
##   <int> <int> <int> <chr>
## 1     3     8    10 Preprocessor1_Model499
```

Usually, the minimal node size give a effect to roc_auc. When the size is 4 and 6, the lines are located on the high parts.

Exercise 8

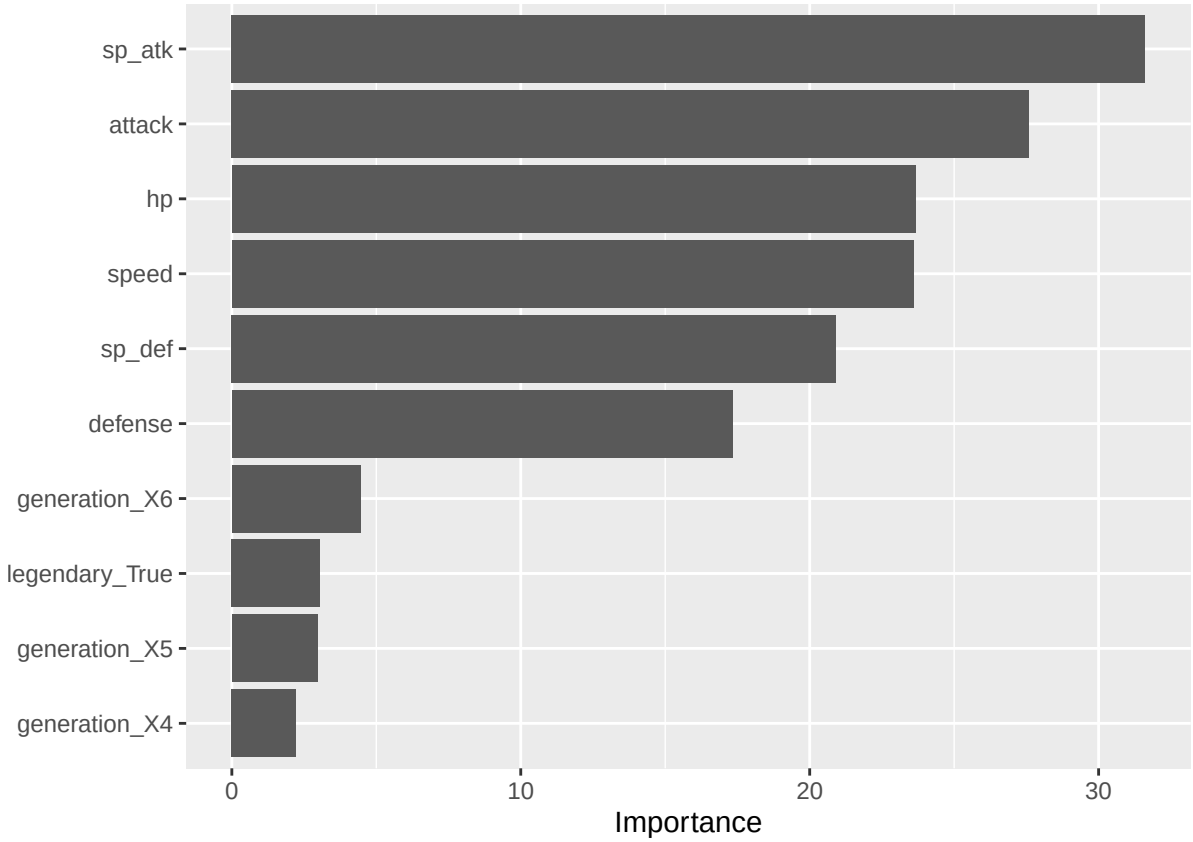
```
rf_tune %>% collect_metrics() %>% arrange(desc(mean))
```

```
## # 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     3     8    10 roc_auc hand_till  0.689     5 0.0221  Preprocessor1_Model~
##  2     5    10    10 roc_auc hand_till  0.682     5 0.0137  Preprocessor1_Model~
##  3     7    10     7 roc_auc hand_till  0.681     5 0.0199  Preprocessor1_Model~
```

```
##  4     4    10     7 roc_auc hand_till  0.680     5 0.0134  Preprocessor1_Model~
##  5     8     8     8 roc_auc hand_till  0.680     5 0.00744 Preprocessor1_Model~
##  6     8     8     6 roc_auc hand_till  0.678     5 0.0123  Preprocessor1_Model~
##  7     6    10     2 roc_auc hand_till  0.677     5 0.00456 Preprocessor1_Model~
##  8     7     6    10 roc_auc hand_till  0.676     5 0.0202  Preprocessor1_Model~
##  9     3    10     3 roc_auc hand_till  0.676     5 0.0144  Preprocessor1_Model~
## 10     7     8     6 roc_auc hand_till  0.675     5 0.0117  Preprocessor1_Model~
## # ... with 502 more rows
```

Exercise 9

```
rf_final = rf_wf %>% finalize_workflow(rf_best)
rf_final_fit = rf_final %>% fit(data = pokemon_train)

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

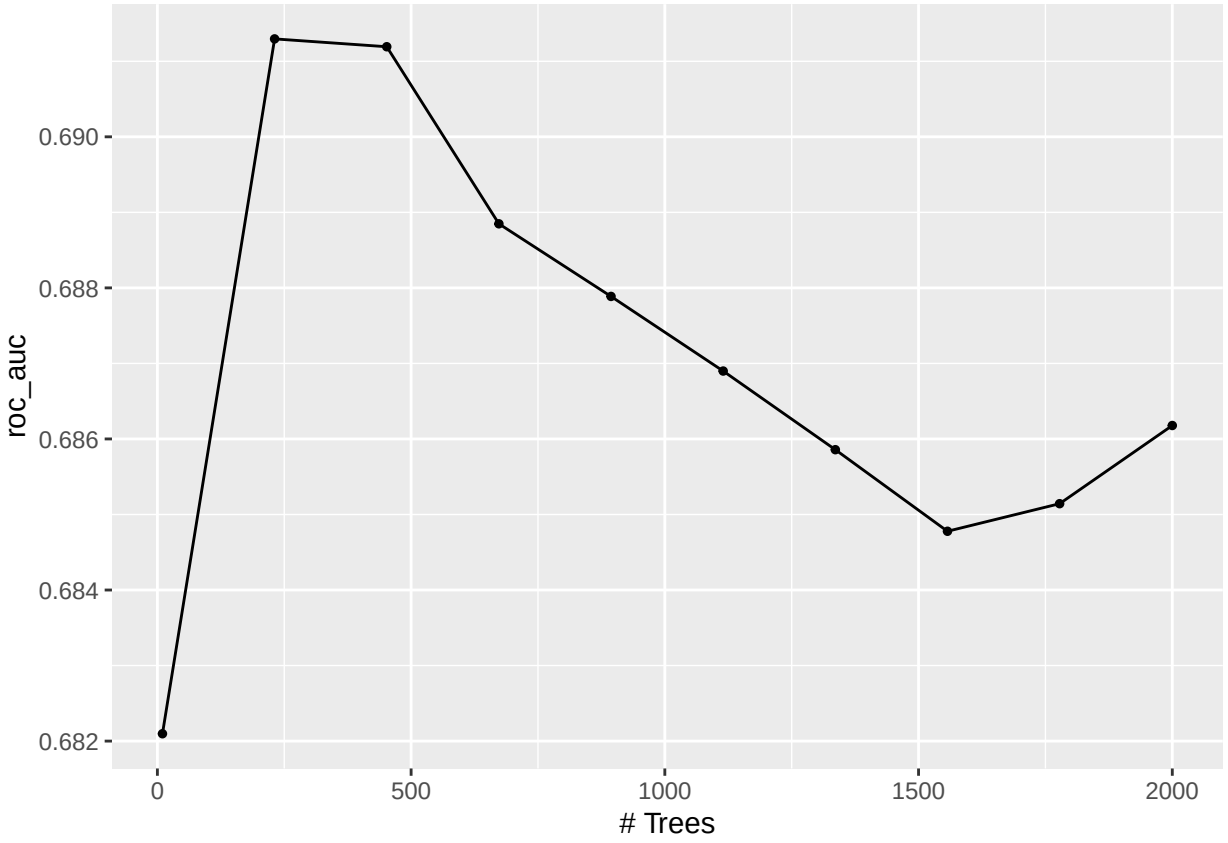


'sp_atk' is the most important variable. Other variables also seem important except 'generation' variables.

Exercise 10

```
xg_spec = boost_tree(trees = tune()) %>%
        set_engine('xgboost') %>%
        set_mode('classification')

xg_wf = workflow() %>%
        add_recipe(pokemon_recipe) %>%
        add_model(xg_spec)
```

```
xg_grid = grid_regular(trees(range = c(10, 2000)),
                       levels = 10)

xg_tune = tune_grid(
        xg_wf,
        resamples = pokemon_folds,
        grid = xg_grid,
        metrics = metric_set(roc_auc))

autoplot(xg_tune)
```



It was getting good at small tree numbers, but after 500, it is getting worse.

```
xg_tune %>% collect_metrics() %>% arrange(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  0.691     5 0.0206  Preprocessor1_Model02
##  2   452 roc_auc hand_till  0.691     5 0.0193  Preprocessor1_Model03
##  3   673 roc_auc hand_till  0.689     5 0.0200  Preprocessor1_Model04
##  4   894 roc_auc hand_till  0.688     5 0.0202  Preprocessor1_Model05
##  5  1115 roc_auc hand_till  0.687     5 0.0198  Preprocessor1_Model06
##  6  2000 roc_auc hand_till  0.686     5 0.0195  Preprocessor1_Model10
##  7  1336 roc_auc hand_till  0.686     5 0.0191  Preprocessor1_Model07
##  8  1778 roc_auc hand_till  0.685     5 0.0193  Preprocessor1_Model09
##  9  1557 roc_auc hand_till  0.685     5 0.0192  Preprocessor1_Model08
```

```
## 10    10 roc_auc hand_till  0.682     5 0.00912 Preprocessor1_Model01
```

Exercise 11

```
xg_best = xg_tune %>% select_best(metric = 'roc_auc')
xg_best
```

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

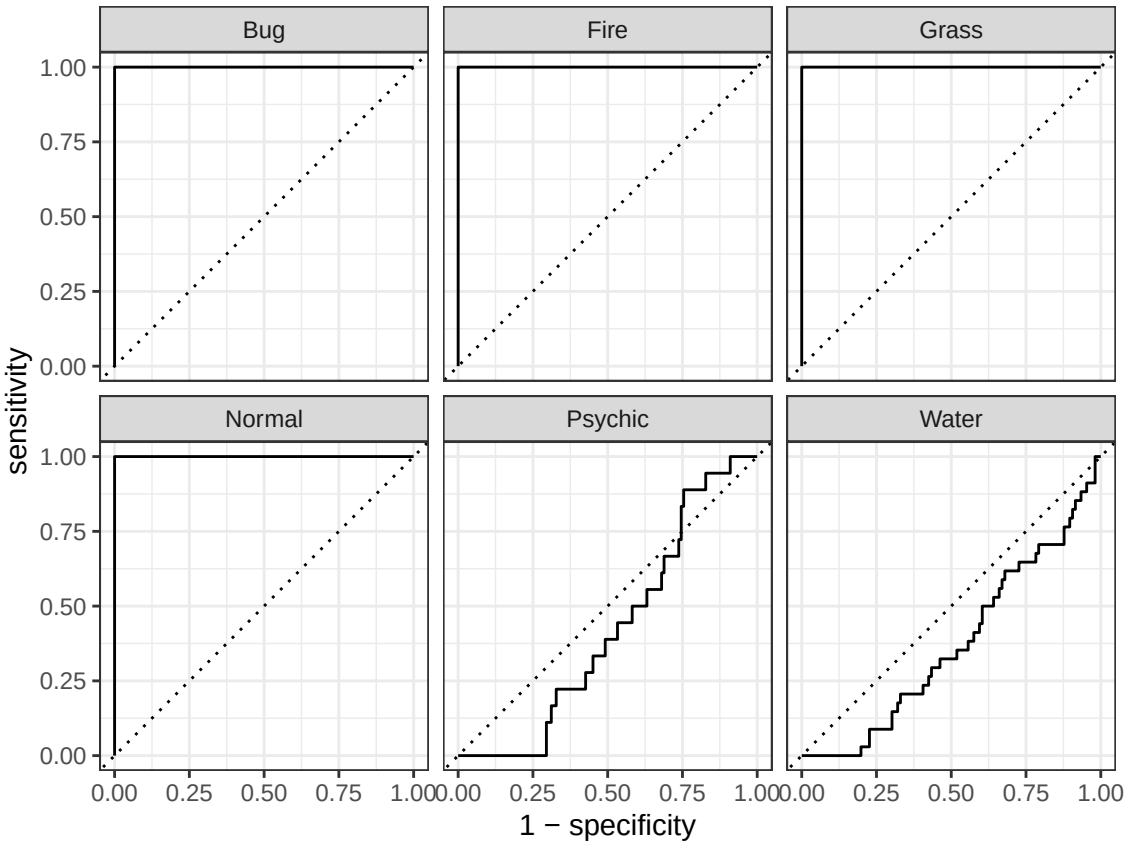
```
score = bind_rows(tree_best, rf_best, xg_best)
score = score %>% add_column('model' = c('Decision Tree', 'Random Forest',
                                         'Boosted Tree'),
                             'roc_auc' = c(0.65, 0.69, 0.69))
score = score[, c('model', 'roc_auc')]

score
```

```
## # A tibble: 3 x 2
##   model         roc_auc
##   <chr>           <dbl>
## 1 Decision Tree    0.65
## 2 Random Forest    0.69
## 3 Boosted Tree     0.69
```

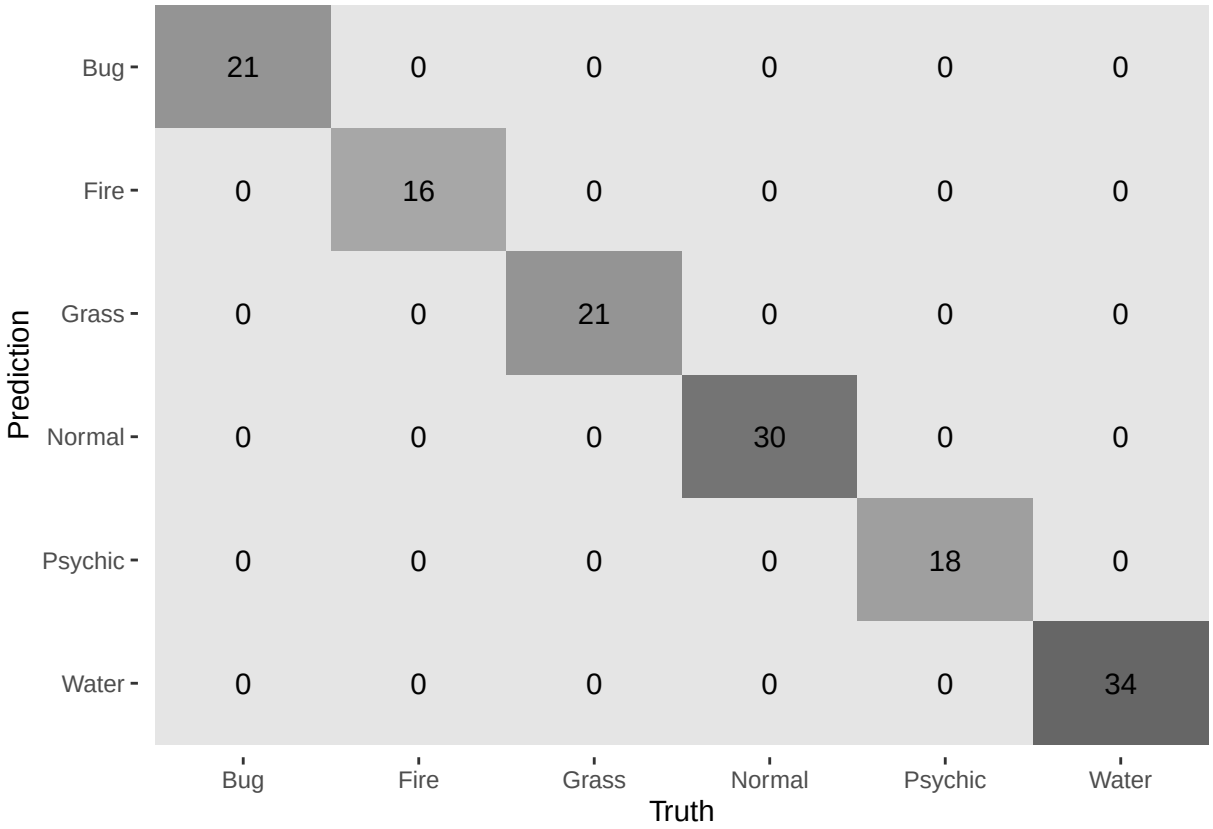
```
final = xg_wf %>% finalize_workflow(xg_best)
final_fit = final %>% fit(data = pokemon_test)

augment(final_fit, new_data = pokemon_test) %>%
        roc_auc(truth = type_1, estimate = c(.pred_Bug, .pred_Fire, .pred_Grass,
                                             .pred_Normal, .pred_Water, .pred_Psychic))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.800
```

```
augment(final_fit, new_data = pokemon_test) %>%
        roc_curve(truth = type_1, estimate = c(.pred_Bug, .pred_Fire, .pred_Grass,
                                               .pred_Normal, .pred_Water, .pred_Psychic)) %>% autoplot()
```




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



Boosted tree is the best model and pure tree is the worst. I don't understand why a few types could not work on the model. The results of decision tree and roc_curve are pretty bad.