Homework 6

Kabir Snell

2022-11-27

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

Question 1

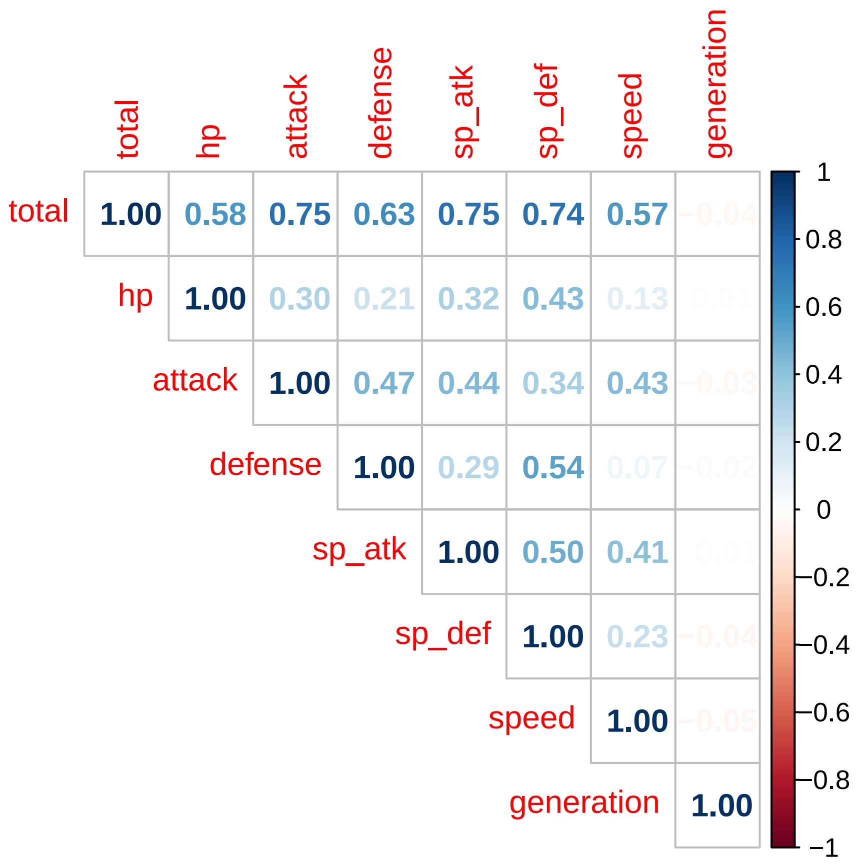
```
# (From Previous Homework)
pokemon <- read.csv('data/Pokemon.csv')

pokemon <- pokemon %>%
  clean_names()

pokemon <- pokemon %>%
  filter(type_1 =='Bug' | type_1 == 'Fire' | type_1 == 'Grass' | type_1 == 'Normal' | type_1 == 'Water'

pokemon$type_1 <- as.factor(pokemon$type_1)
pokemon$legendary <- as.factor(pokemon$legendary)
pokemon$generation <- as.factor(pokemon$generation)

set.seed(727)

pokemon_split <- initial_split(pokemon, prop = 0.80)

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

pokemon_split
```

```
## <Training/Testing/Total>
## <366/92/458>
```

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

pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_
  step_dummy(legendary) %>%
  step_dummy(generation) %>%
  step_normalize(all_predictors())
```

Question 2

```
M <- pokemon_train %>%
  select(total, hp, attack, defense, sp_atk, sp_def, speed, generation)

M$generation <- as.numeric(M$generation)

M <- cor(M)

corrplot::corrplot(M, method = 'number', type = 'upper')
```

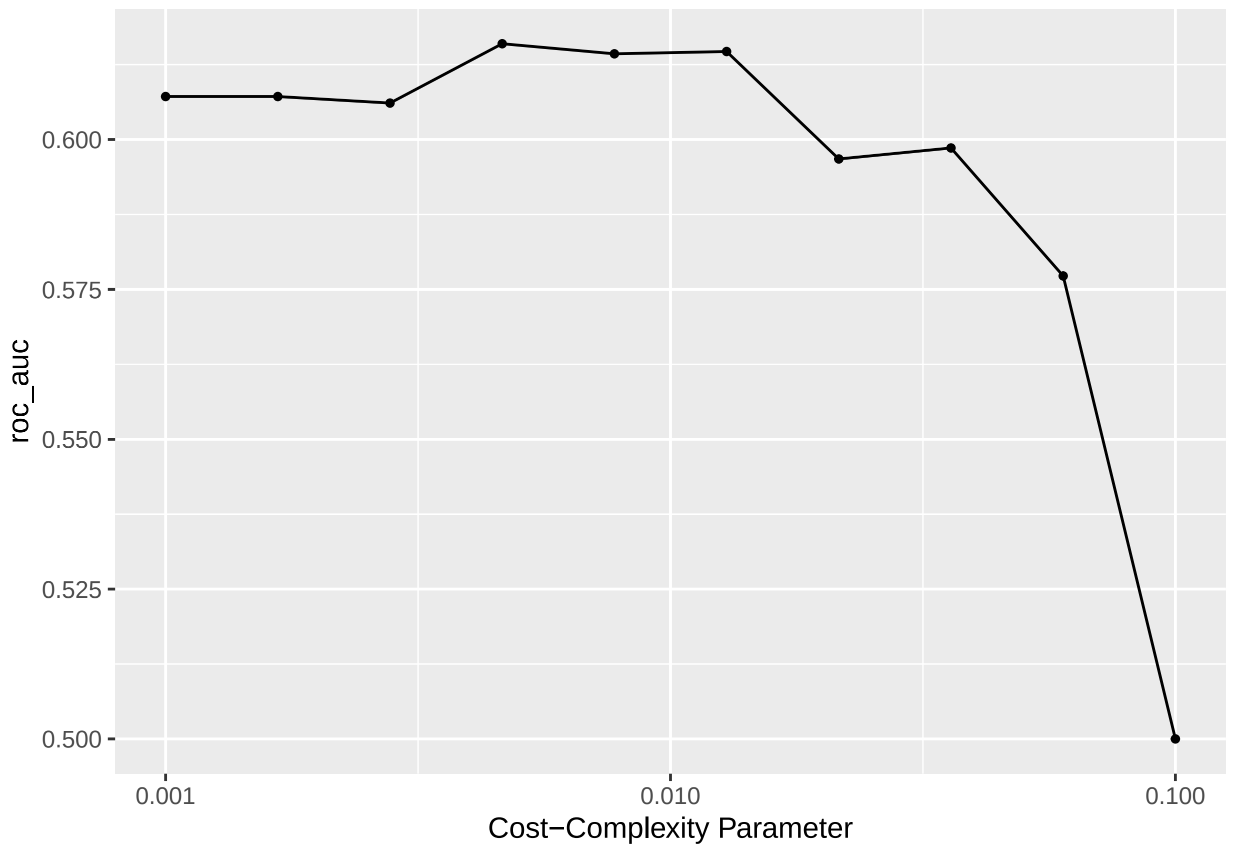


The reason I chose not to include the variables I took out was because there is no clear 'order' for the variables. For example, changing pokemon type to numeric would have an arbitrary ordering for pokemon type that would not be statistically significant. I kept generation as there is a clear ordering of the variables.

There are strong relationships between total and many of the other variables, this makes sense as total is a linear combination of all of the other variables except generation. There is also quite a strong correlation between special defense and special attack.

Question 3

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

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

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

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

tune_res <- tune_grid(
  pokemon_wkflow,
  resamples = pokemon_fold,
  grid = param_grid,
  metrics = metric_set(roc_auc))

autoplot(tune_res)
```



As the model increases in complexity the roc_auc metric increases, and peaks around .01. After this, the metric sharply decreases to its lowest value.

Question 4

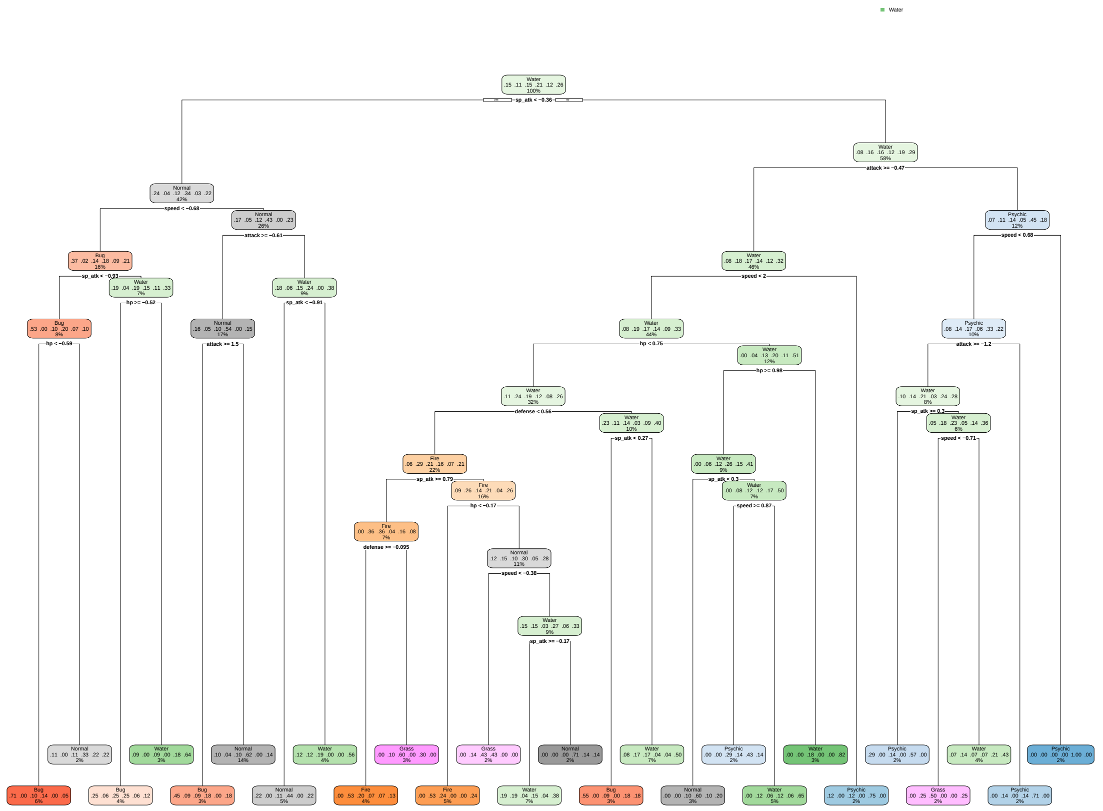
```
decision_tree_results <- collect_metrics(tune_res) %>%
  arrange(desc(mean)) %>%
  head(1)

decision_tree_results
```

```
## # A tibble: 1 x 7
##   cost_complexity .metric .estimator  mean     n std_err .config
##             <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1         0.00464 roc_auc hand_till  0.616     5  0.0143 Preprocessor1_Model04
```

The roc_auc of my best performing pruned decision tree on the folds is .616

Question 5

```
best_complexity <- select_best(tune_res)
class_tree_final <- finalize_workflow(pokemon_wkflow, best_complexity)
class_tree_final_fit <- fit(class_tree_final, data=pokemon_train)

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



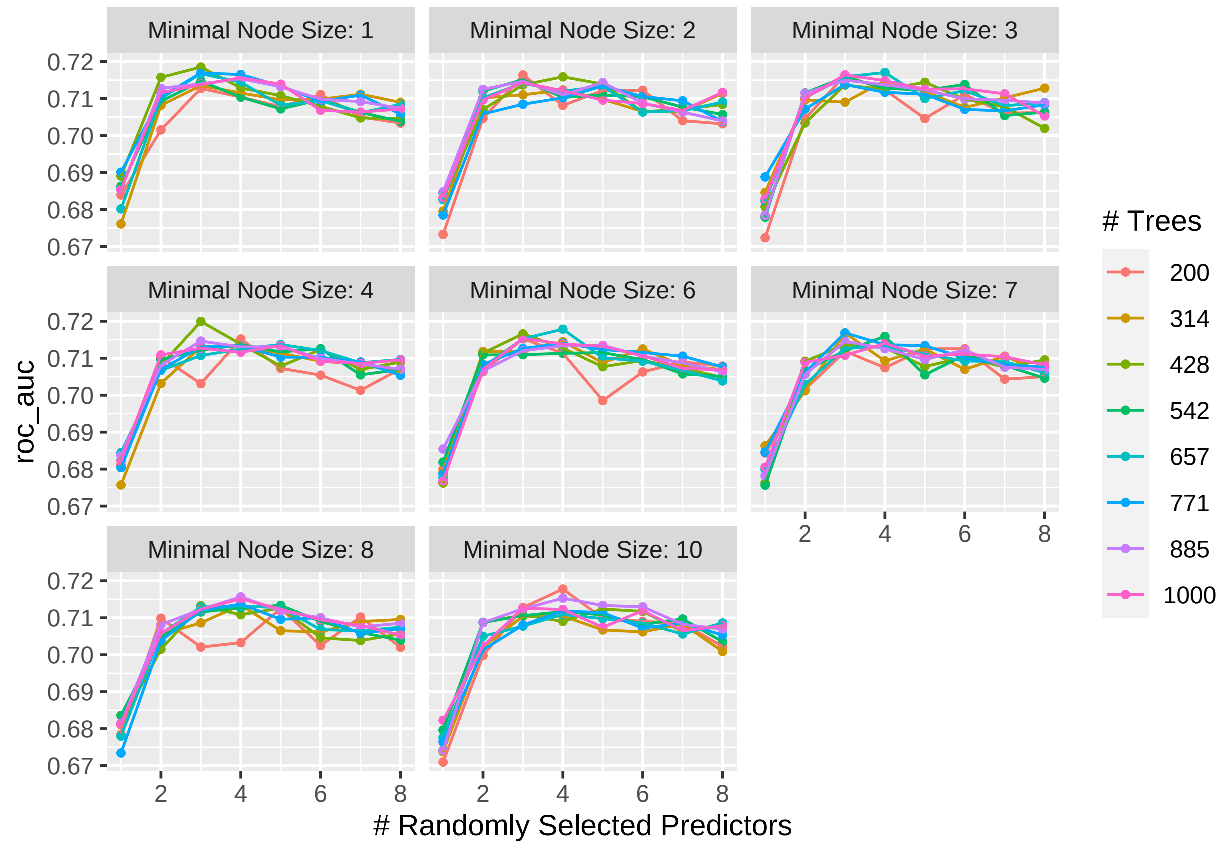
## Question 5

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

rf_workflow <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(rf_spec)

rf_parameter_grid <- grid_regular(mtry(range = c(1, 8)), trees(range = c(200,1000)), min_n(range = c(1,
```

Question 6

```
rf_tune_res <- tune_grid(
  rf_workflow,
  resamples = pokemon_fold,
  grid = rf_parameter_grid,
  metrics = metric_set(roc_auc)
)

write_rds(rf_tune_res, file = "rf.rds")
```

```r
rf_tuned <- read_rds(file = "rf.rds")

autoplot(rf_tuned)
```



In each of the models, a low number of randomly selected predictors (1-2) does not seem to perform very well. The peak roc_auc seems to be around 3-5 randomly selected predictors while numbers larger than this tends to decrease roc_auc.

As for minimal node size, I would say that generally they all performed relatively the same, with node size 8 and 10 having a slightly lower average roc_auc than the rest of the models.

The number of trees does not seem to effect the result that much since there is a lot of overlapping in the lines, but 800 trees generally seems to perform the best.

Question 7

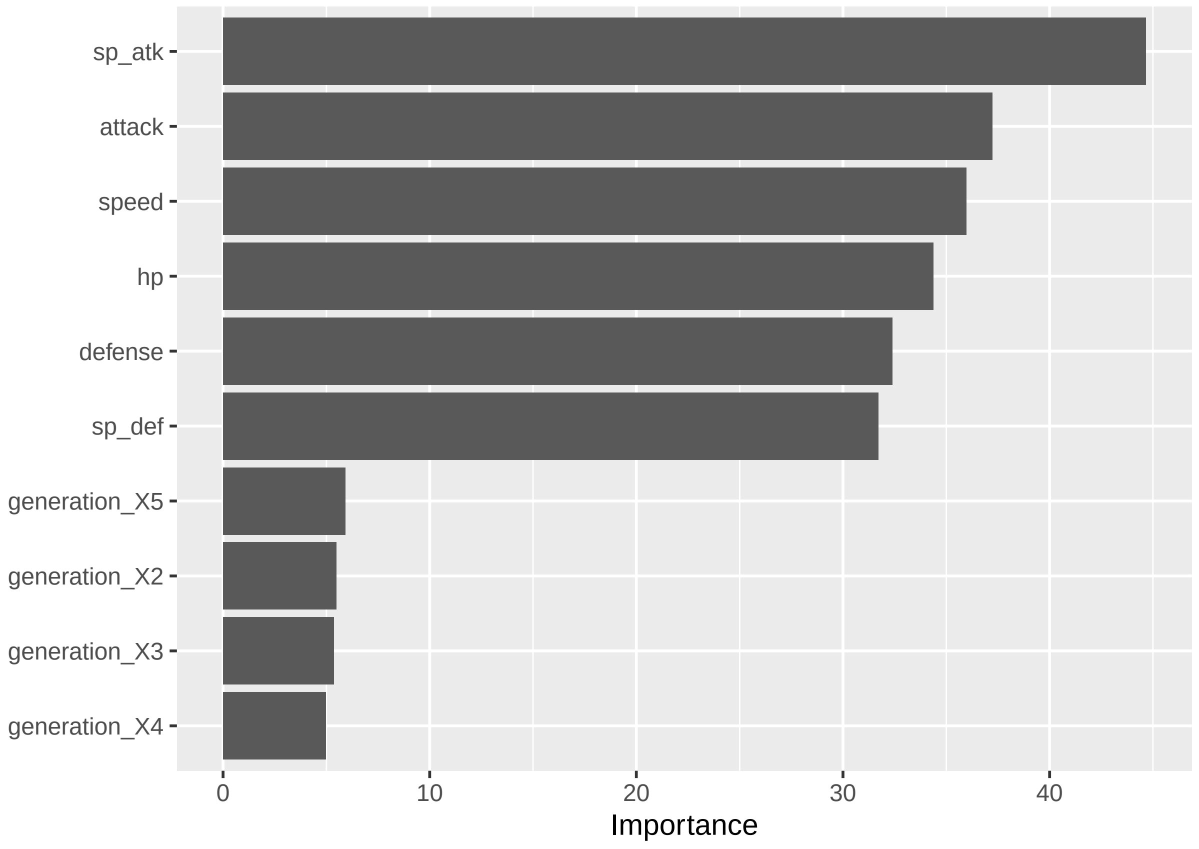
```r
rf_best_roc_auc <- rf_tuned %>%
  collect_metrics() %>%
  arrange(desc(mean)) %>%
  head(1)

rf_best_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     3   428     4 roc_auc hand_till  0.720     5  0.0147 Preprocessor1_Model2~
```

The ‘roc_auc’ of the best performing model is .7199304

Question 8

```r
best_rf <- select_best(rf_tuned)
rf_final_workflow <- finalize_workflow(rf_workflow, best_rf)
rf_final_fit <- fit(rf_final_workflow, data = pokemon_train)
rf_final_fit %>%
  extract_fit_engine() %>%
  vip()
```

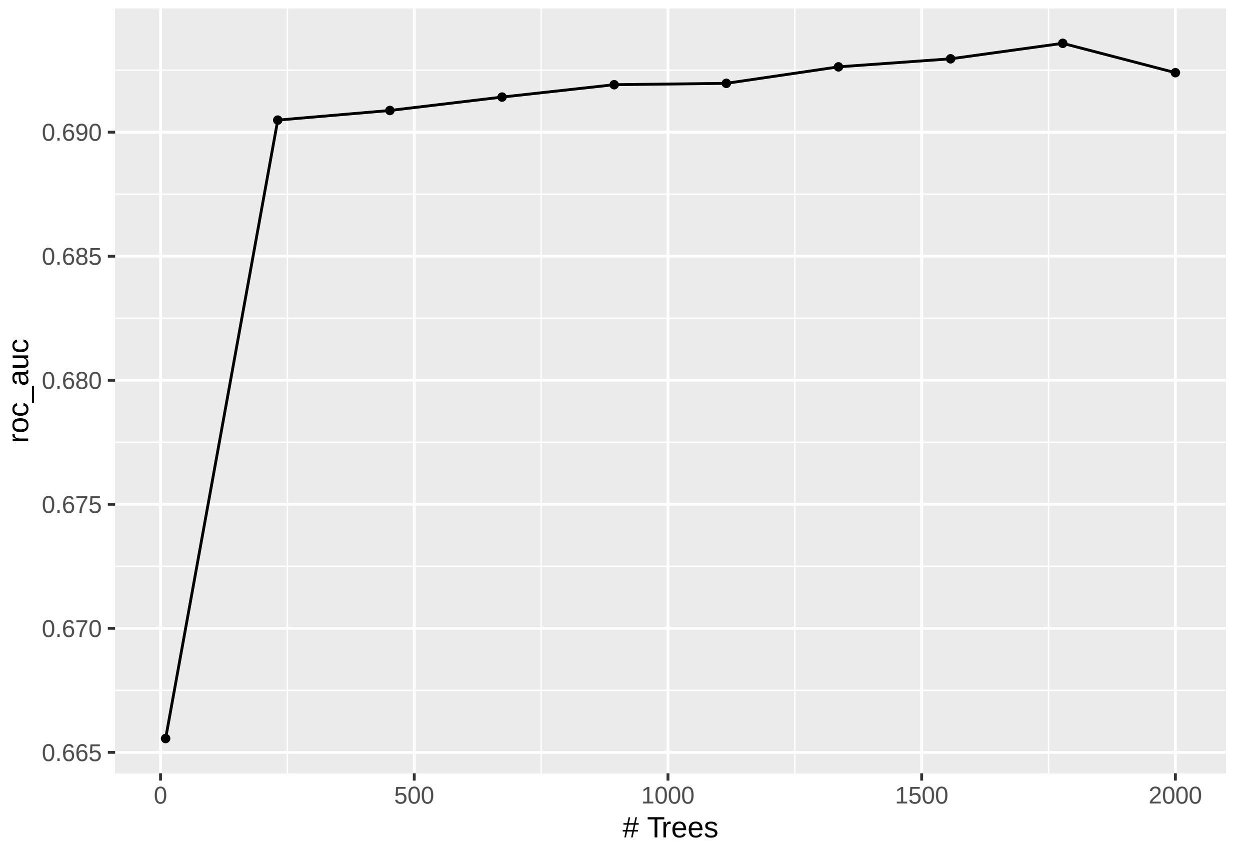


Question 9

```r
boosted_spec <- boost_tree(trees = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

```r
boosted_workflow <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(boosted_spec)

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

```r
boosted_tune_res <- tune_grid(
  boosted_workflow,
  resamples = pokemon_fold,
  grid = boosted_grid,
  metrics = metric_set(roc_auc)
)

autoplot(boosted_tune_res)
```



As the number of trees increases, the roc_auc also increases. However, when the number of trees reaches 2000, the roc_auc begins to decline.

```r
boosted_best_roc_auc <- boosted_tune_res %>%
  collect_metrics() %>%
  arrange(desc(mean)) %>%
  head(1)

boosted_best_roc_auc
```

```
## # A tibble: 1 x 7
##   trees .metric .estimator  mean     n std_err .config
##   <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1  1778 roc_auc hand_till  0.694     5  0.0140 Preprocessor1_Model09
```

The best performing boosted tree model on the folds had a roc_auc of .6935

Question 10

```
# Printing table of ROC AUC values
roc_auc_tibble <- tibble(Models = c("Pruned Tree", "Random Forest", "Boosted Tree"), ROC_AUC_Value = c(

roc_auc_tibble
```

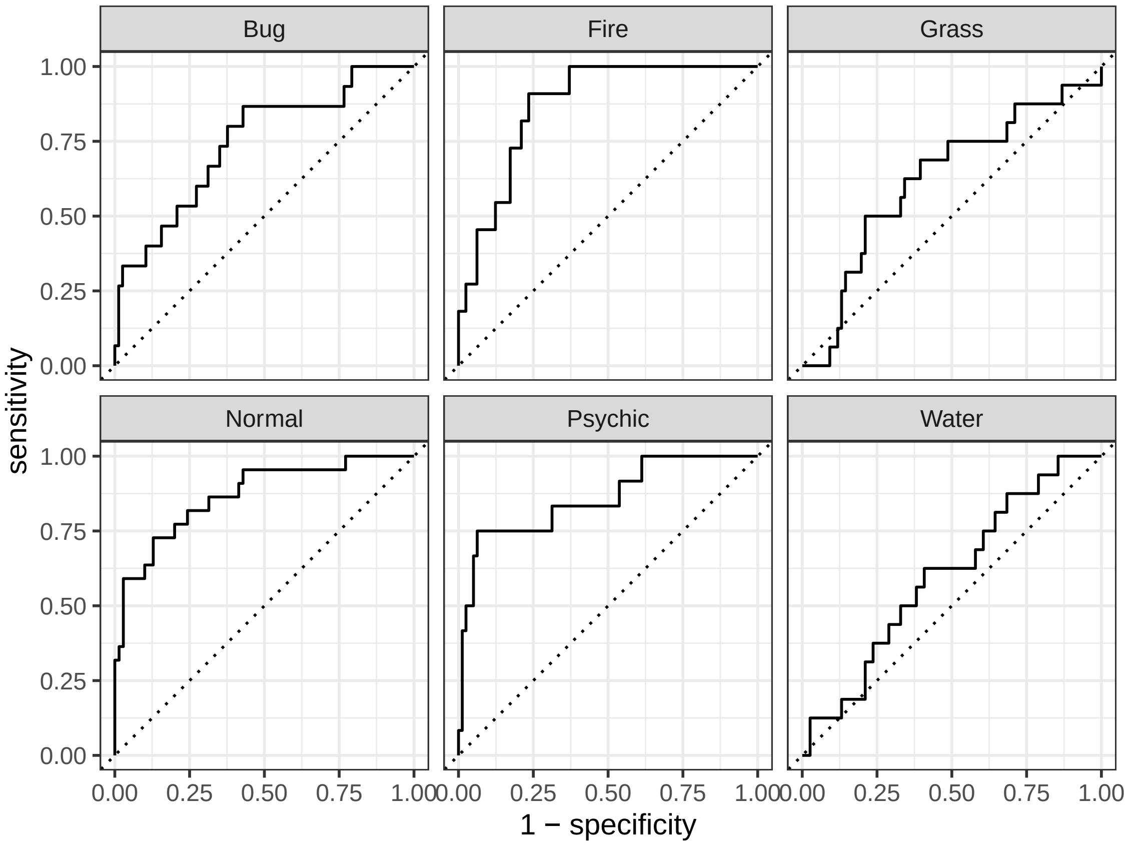
```
## # A tibble: 3 x 2
##   Models        ROC_AUC_Value
##   <chr>                 <dbl>
## 1 Pruned Tree           0.616
## 2 Random Forest         0.720
## 3 Boosted Tree          0.694
```

```
best_rf_final <- select_best(rf_tuned)
rf_final_workflow_testing <- finalize_workflow(rf_workflow, best_rf_final)
rf_final_fit_testing <- fit(rf_final_workflow_testing, data = pokemon_train)
```

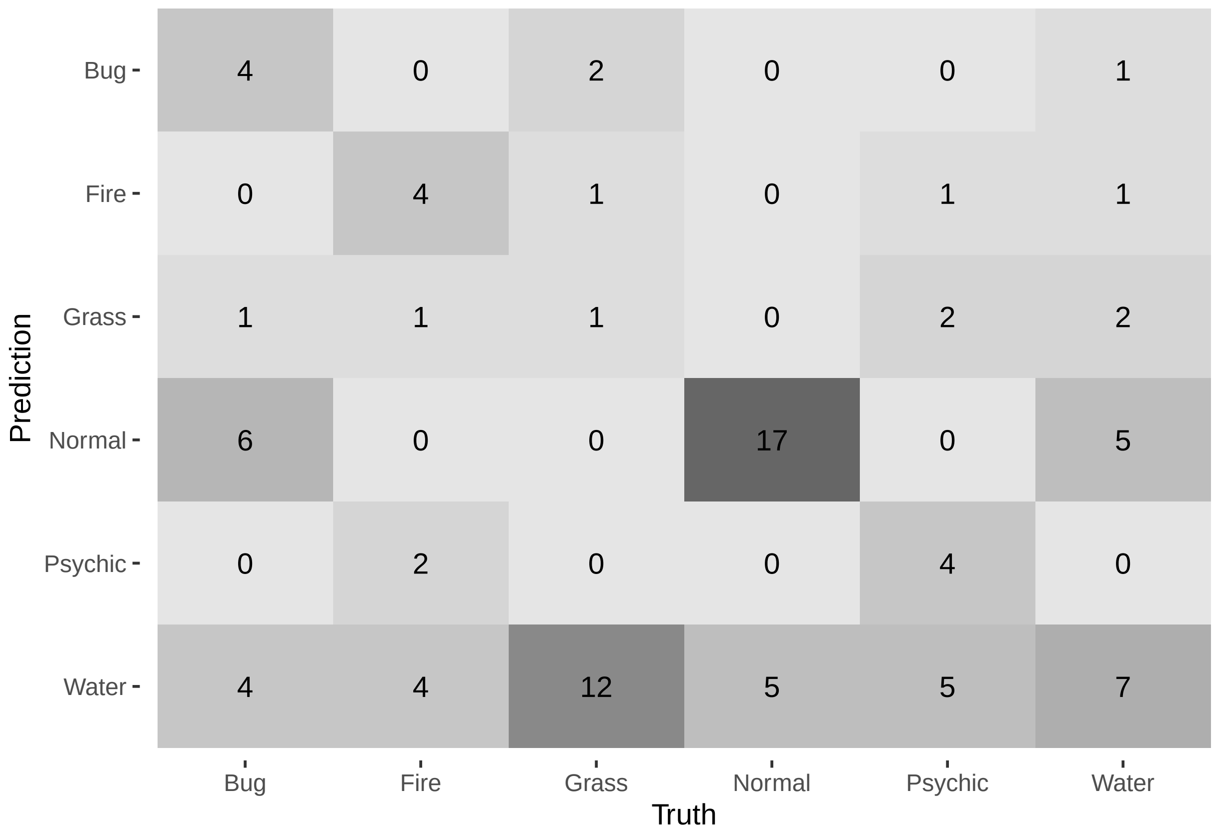
```
final_tibble <- augment(rf_final_fit_testing, new_data = pokemon_test)
final_tibble %>%
  roc_auc(truth = type_1, estimate = .pred_Bug:.pred_Water)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.756
```

```
all_roc_curves <- final_tibble %>%
  roc_curve(truth = type_1, estimate = .pred_Bug:.pred_Water) %>%
  autoplot()
all_roc_curves
```



```
confusion_matrix <- final_tibble %>%
  conf_mat(type_1, .pred_class) %>%
  autoplot(type = "heatmap")
confusion_matrix
```



After fitting the model to the training data set and applying it to the testing data set, we can see that the best prediction classes are Bug, Fire, Normal, and Psychic.

Much like in the last homework assignment, our model had a very difficult time predicting grass type pokemon as well as water type pokemon.