hw6

Seongu Lee

5/26/2022

1.

```
set.seed(731)
poke <- read.csv("C:/Users/sungu/Desktop/homework-6/data/Pokemon.csv")
clean<- clean_names(poke)
head(poke)
```

```
##   X.                  Name Type.1 Type.2 Total HP Attack Defense Sp..Atk
## 1  1             Bulbasaur  Grass Poison   318 45     49      49      65
## 2  2               Ivysaur  Grass Poison   405 60     62      63      80
## 3  3              Venusaur  Grass Poison   525 80     82      83     100
## 4  3 VenusaurMega Venusaur  Grass Poison   625 80    100     123     122
## 5  4            Charmander   Fire          309 39     52      43      60
## 6  5            Charmeleon   Fire          405 58     64      58      80
##   Sp..Def Speed Generation Legendary
## 1      65    45          1     False
## 2      80    60          1     False
## 3     100    80          1     False
## 4     120    80          1     False
## 5      50    65          1     False
## 6      65    80          1     False
```

```
head(clean)
```

```
##   x                  name type_1 type_2 total hp attack defense sp_atk sp_def
## 1 1             Bulbasaur  Grass Poison   318 45     49      49     65     65
## 2 2               Ivysaur  Grass Poison   405 60     62      63     80     80
## 3 3              Venusaur  Grass Poison   525 80     82      83    100    100
## 4 3 VenusaurMega Venusaur  Grass Poison   625 80    100     123    122    120
## 5 4            Charmander   Fire          309 39     52      43     60     50
## 6 5            Charmeleon   Fire          405 58     64      58     80     65
##   speed generation legendary
## 1    45          1     False
## 2    60          1     False
## 3    80          1     False
## 4    80          1     False
## 5    65          1     False
## 6    80          1     False
```

```r
filtered <- clean %>%
  filter(type_1 == "Bug" | type_1 == "Fire" | type_1 == "Grass" | type_1 == "Normal" |type_1 == "Water"
filtered$legendary <- factor(filtered$legendary)
filtered$generation <- factor(filtered$generation)
filtered$type_1<- factor(filtered$type_1)
head(filtered)
```

```
##   x                  name type_1 type_2 total hp attack defense sp_atk sp_def
## 1 1             Bulbasaur  Grass Poison   318 45     49      49     65     65
## 2 2               Ivysaur  Grass Poison   405 60     62      63     80     80
## 3 3              Venusaur  Grass Poison   525 80     82      83    100    100
## 4 3 VenusaurMega Venusaur  Grass Poison   625 80    100     123    122    120
## 5 4            Charmander   Fire          309 39     52      43     60     50
## 6 5            Charmeleon   Fire          405 58     64      58     80     65
##   speed generation legendary
## 1    45          1     False
## 2    60          1     False
## 3    80          1     False
## 4    80          1     False
## 5    65          1     False
## 6    80          1     False
```

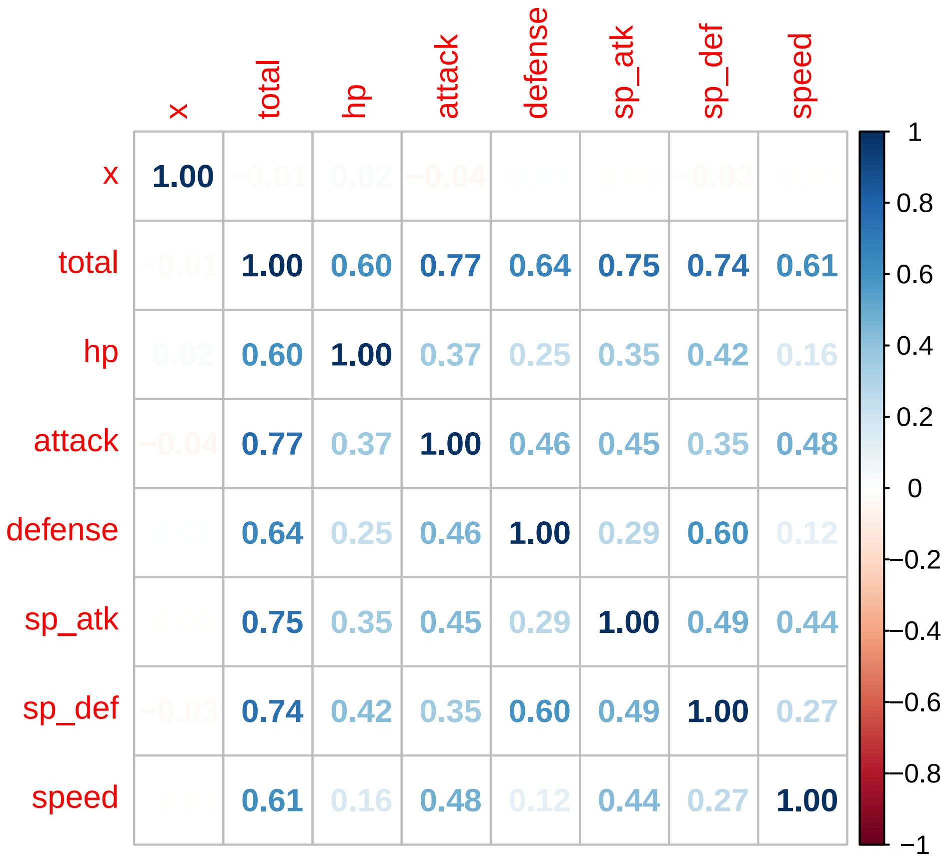
```r
split <- initial_split(filtered, strata = type_1, prop = 0.7)
train <- training(split)
test <- testing(split)
```

```r
fold <- vfold_cv(train, strata = type_1, v = 5)
```

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

2.

```r
train %>%
  select(where(is.numeric)) %>%
  cor() %>%
  corrplot(method = 'number')
```



Total and attack, total and defense, total and sp_atk, total and sp_def are correlated. These make sense to me, because the total is sum of all stats. The total should be correlated to other stats.

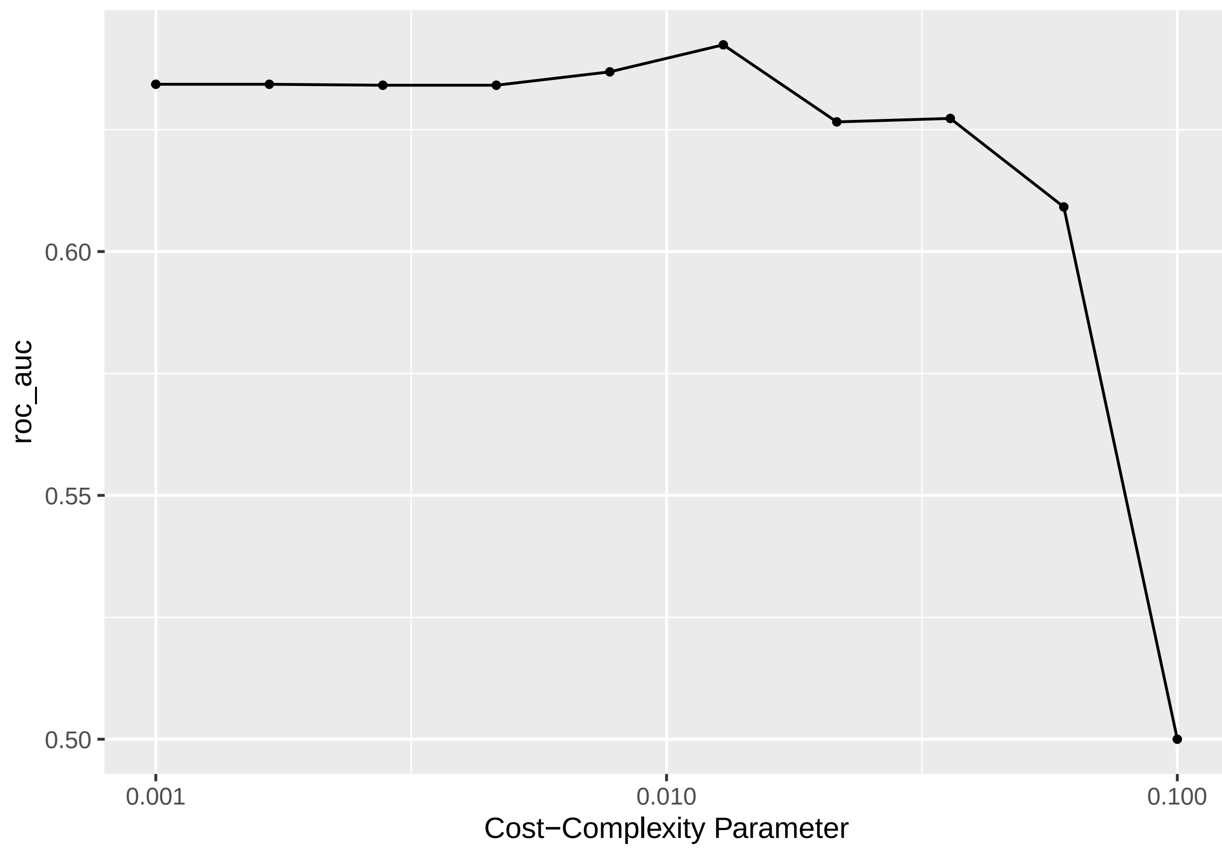
3.

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

tree_wk <- workflow() %>%
  add_recipe(recipe) %>%
  add_model(tree)

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

tune_res <- tune_grid(tree_wk,
                      resamples = fold,
                      grid = param_grid,
                      metrics = metric_set(roc_auc))
autoplot(tune_res)
```



It dropped rapidly after 0.05. Also, the decision tree performs better with smaller penalty as the plot showed.

4.

```
best<- collect_metrics(tune_res) %>%
        arrange(mean)
best_auc<- max(best$mean)
best_auc
```

```
## [1] 0.6423822
```

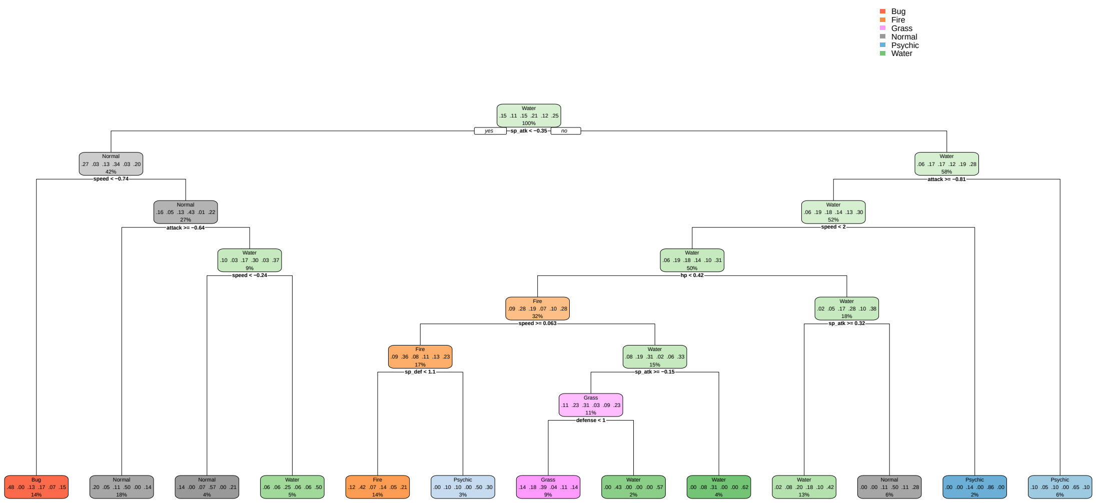
Best-performing is 0.642.

5.

```
final_tree = finalize_workflow(tree_wk, select_best(tune_res))

fit_tree = fit(final_tree, train)
```

```
fit_tree %>%
  extract_fit_engine() %>%
  rpart.plot(roundint=FALSE)
```



5.

```
rf <- rand_forest() %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")

rf_wk <- workflow() %>%
  add_model(rf %>% set_args(mtry = tune(), trees = tune(), min_n = tune())) %>%
  add_recipe(recipe)
```

mtry: The number of predictors that will be sampled in tree model.

trees: The number of trees created in tree model.

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

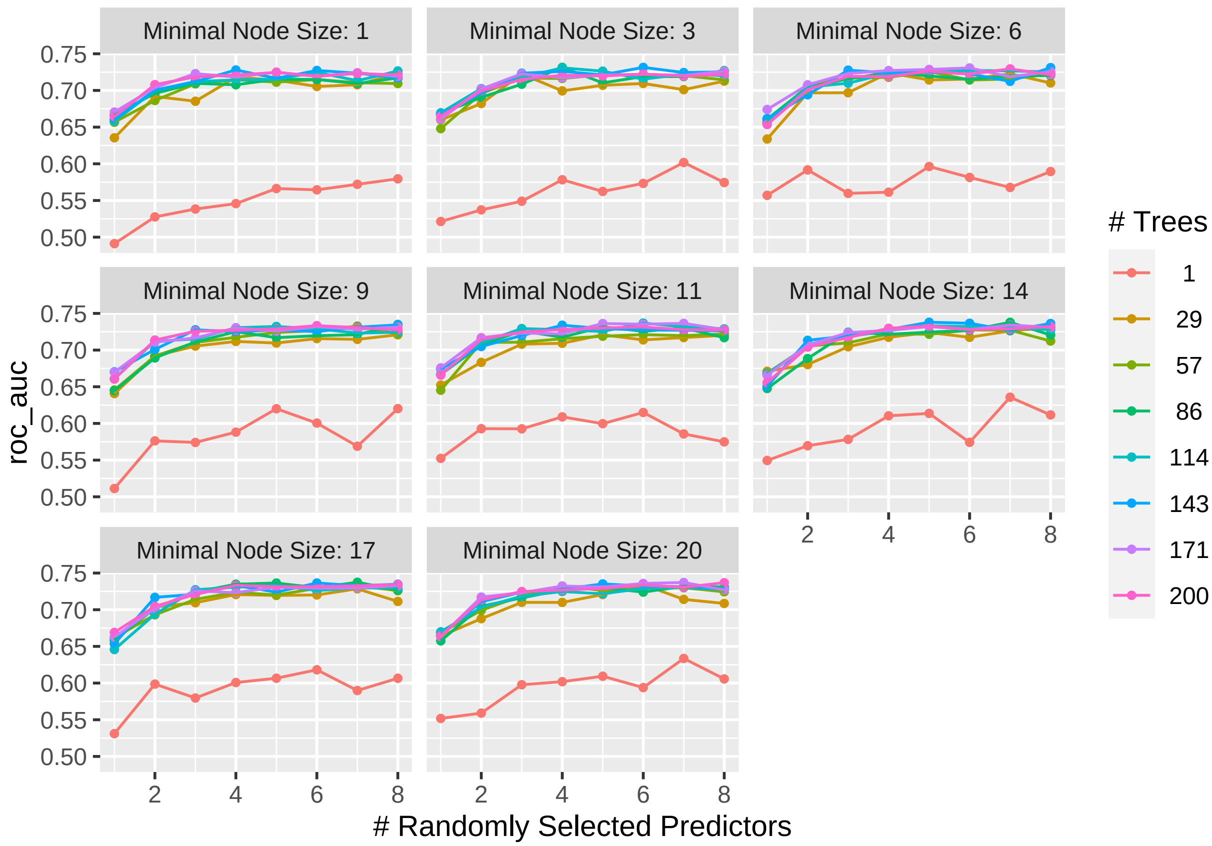
```
rf_grid <- grid_regular(mtry(range = c(1, 8)), trees(range = c(1,200)), min_n(range = c(1, 20)), levels
```

There are 8 predictors. So, if we should use 1 to 8 numbers to represent the all predictors. mtry = 8 represents a random sampled predictor.

6.

```r
rf_tune <- tune_grid(
  rf_wk,
  resamples = fold,
  grid = rf_grid,
  metrics = metric_set(roc_auc)
  )

autoplot(rf_tune)
```



I observed that higher number of trees shows more accuracy. Also, when tree is 1, the accuracy is low. mtry should be (1,8) and trees should be at least more than 2 as I observed. And min_n doesn't really affect to the best performance.

7.

```r
random <- collect_metrics(rf_tune) %>%
          arrange(mean)
random_auc<- max(tail(random$mean))
random_auc
```

```
## [1] 0.7379538
```

The best model is 0.738

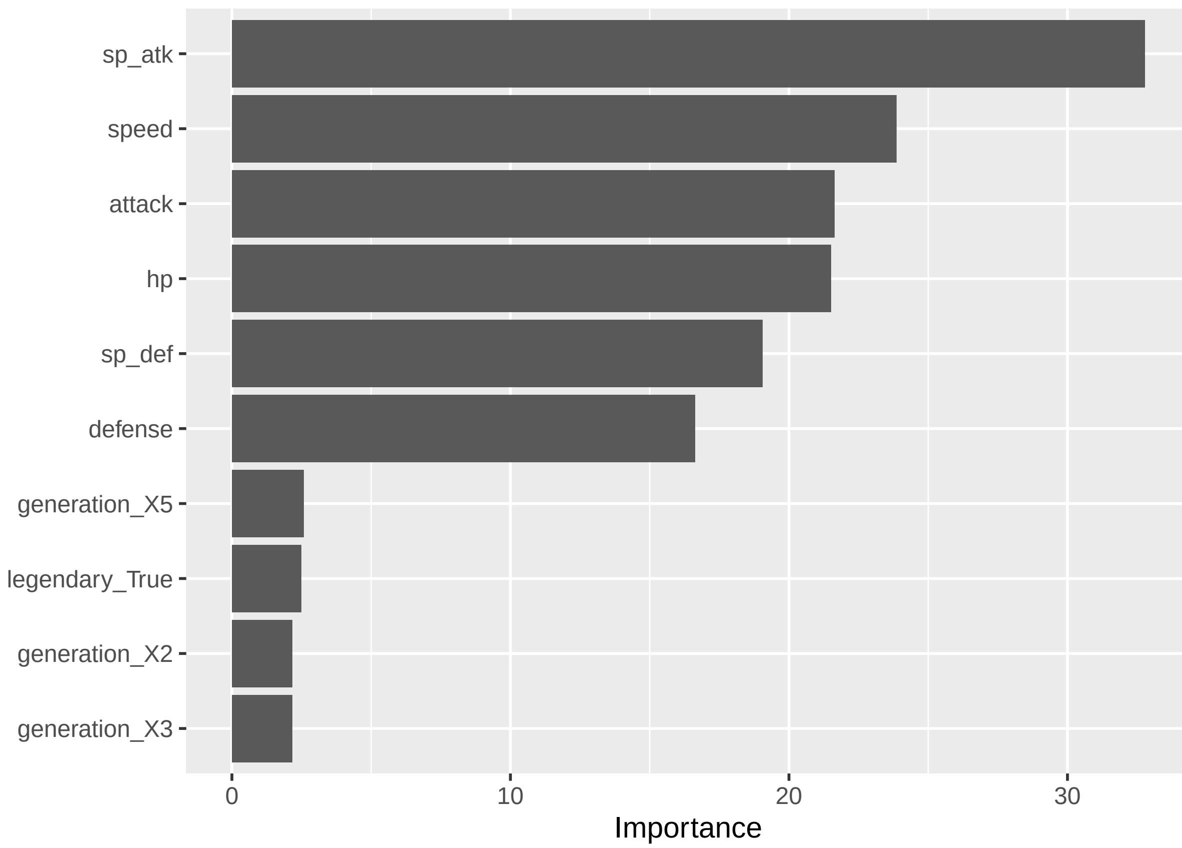
8.

```
best <- select_best(rf_tune)

final <- finalize_workflow(rf_wk, best)

final_fit <- fit(final, data = train)

final_fit %>%
  extract_fit_engine() %>%
  vip()
```



sp_atk is most useful and generation is the least useful. I didn't expect the sp_atk will be the most important variable.

9.

```
boost = boost_tree(trees = tune()) %>%
  set_engine("xgboost") %>%
```

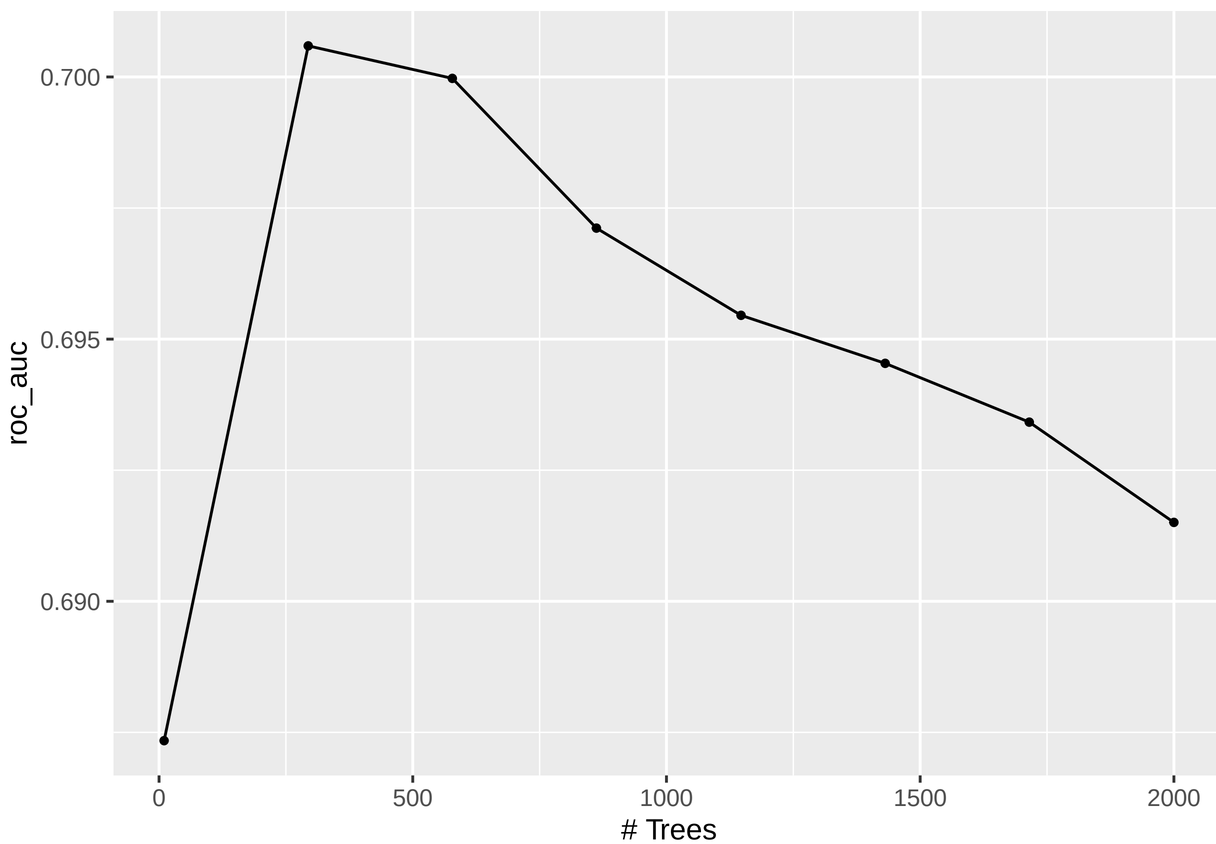
```
  set_mode("classification")

boost_wk = workflow() %>%
  add_recipe(recipe) %>%
  add_model(boost)
```

```
boost_grid <- grid_regular(trees(range = c(10,2000)), levels = 8)
```

```
boost_tune <- tune_grid(
  boost_wk,
  resamples = fold,
  grid = boost_grid,
  metrics = metric_set(roc_auc)
  )
```

```
autoplot(boost_tune)
```



After the number of trees reach to 250, the roc_auc decreased. Before 250 trees, roc_auc increases with increasing number of trees. After 250, the roc_auc decreases.

```
boostM<- collect_metrics(boost_tune) %>%
          arrange(mean)
boost_auc<- max(boostM$mean)
boost_auc
```

```
## [1] 0.7005931
```

The best roc_auc is 0.700

10.

```r
table <- matrix(c(best_auc, random_auc, boost_auc),ncol=3)
rownames(table) <- c('roc auc')
colnames(table) <- c('best-performing pruned tree', 'randomforest','boosted tree models')
table
```

```
##         best-performing pruned tree randomforest boosted tree models
## roc auc                   0.6423822    0.7379538           0.7005931
```

The best performed one is random forest.

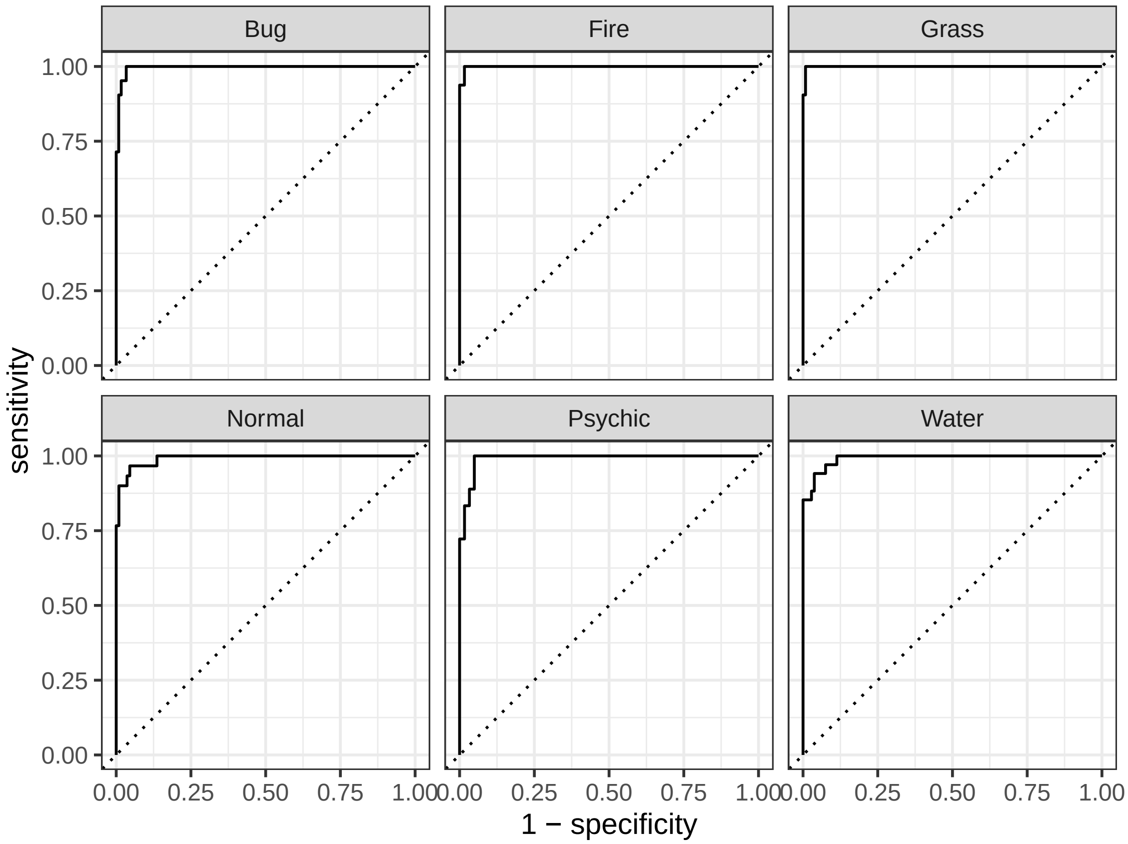
```r
best_model <- select_best(rf_tune, metric = 'roc_auc')
final1<- finalize_workflow(rf_wk, best_model)
final_fit1<- fit(final1, test)
```

```r
result <- augment( final_fit1, new_data = test)

roc_auc(result, type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic, .pred_Water)
```

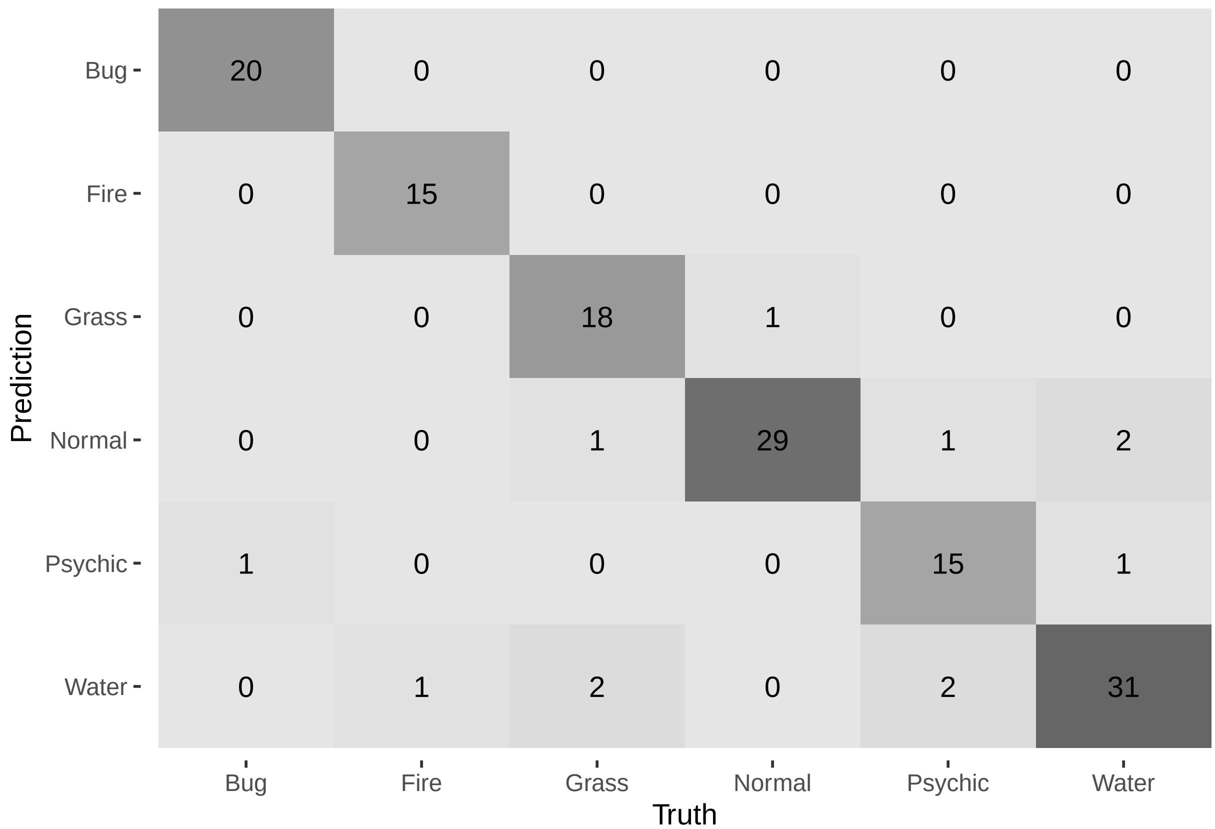
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.995
```

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



auc is 0.995!

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



Water was best at predicting. Normal was great. But, the fire was worst at predicting.