

# hw\_6

Sabrina Lem

5/16/2022

Exercise 1:

```
# clean data
pokemon <- janitor::clean_names(data)
#pokemon <- pokemon_clean %>% filter(type_1 == "Bug"/type_1 == "Fire"/
                                #type_1 == "Grass"/type_1 == "Normal"/
                                #type_1 == "Water"/type_1 == "Psychic")
pokemon <- pokemon[pokemon$type_1 %in% c("Bug", "Fire", "Grass", "Normal", "Water", "Psychic"),]
pokemon$type_1 <- as.factor(pokemon$type_1)
pokemon$legendary <- as.factor(pokemon$legendary)
#split
set.seed(1027)
pokemon_split <- initial_split(pokemon, prop= 0.8, strata = "type_1")

pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)
# folds
pokemon_folds <- vfold_cv(pokemon_train, strata = "type_1", v = 5)
# recipe
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk +
                          attack + speed + defense +
                          hp + sp_def, pokemon_train) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())
```

Exercise 2:

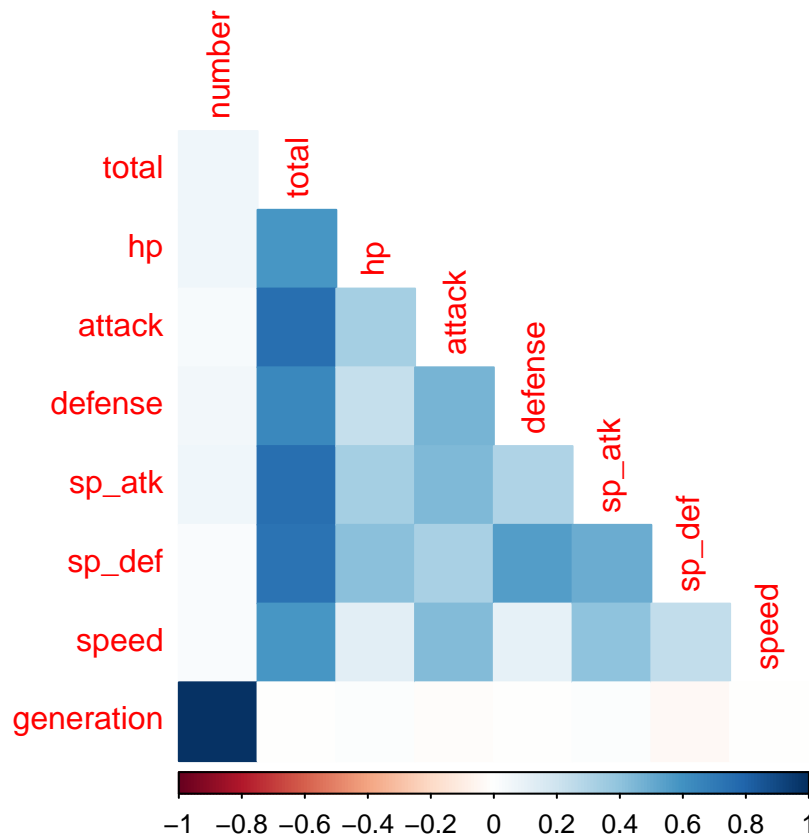
```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
pokemon %>%
  select(is.numeric) %>%
  cor() %>%
  corrplot(type="lower", diag= FALSE, method = 'color')
```

```
## 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.
```



Total is strongly positively correlated with all other variables. In general most variables are positively correlated with each other. These relationships makes sense because a Pokemon with a higher level of defense or attack will most likely have a higher level of speed or defense. In other words a better quality Pokemon will have overall relatively higher levels to their characteristics.

Exercise 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)

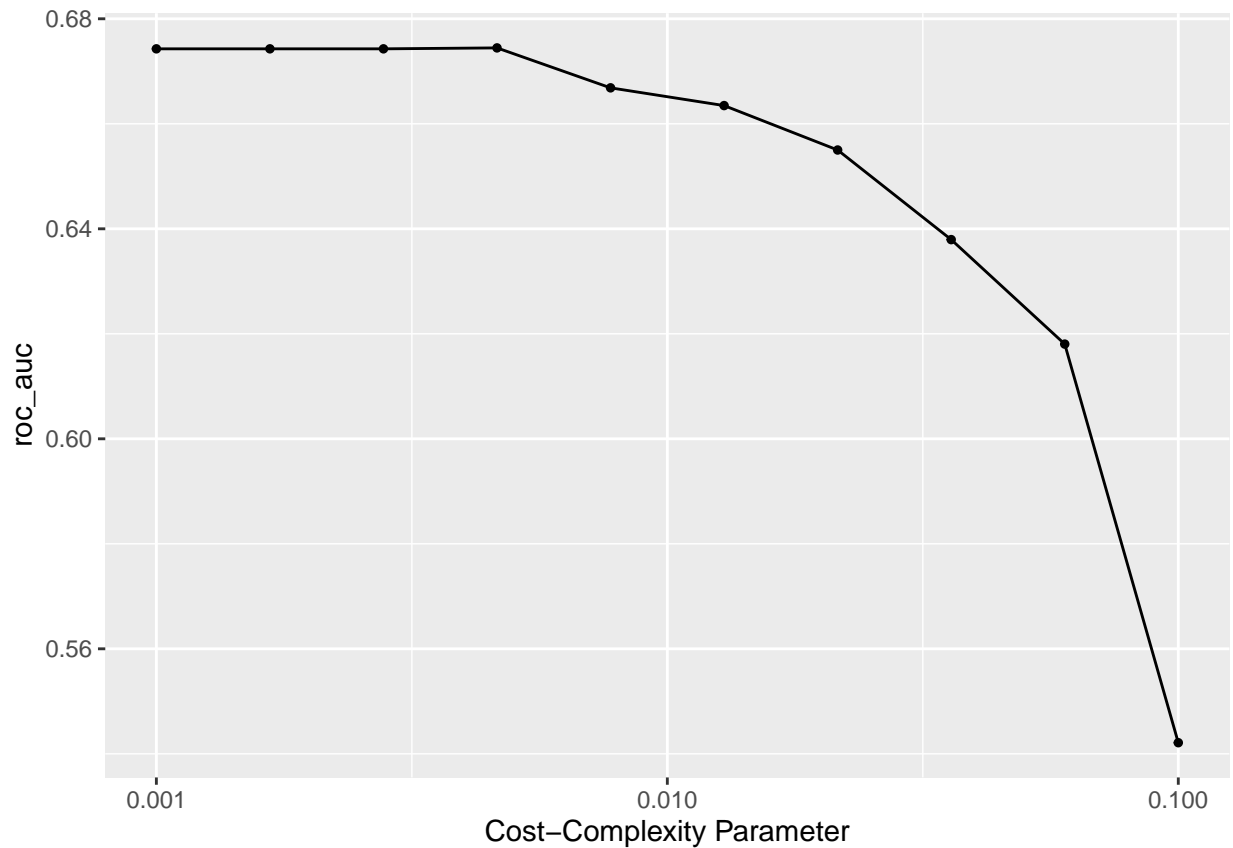
param_grid <- grid_regular(cost_complexity(range = c(-3, -1)), levels = 10)

tune_res <- tune_grid(
  class_tree_wf,
```

```

  resamples = pokemon_folds,
  grid = param_grid,
  metrics = metric_set(roc_auc)
)
autoplot(tune_res)

```



After .008 the curve takes a downward trend in roc-auc value. The curve continues hyperbolically fall as cost-complexity increases. A single decision tree has a better performance with a lower cost-complexity parameter. Exercise 4:

```

best_compl <- select_best(tune_res)
best_compl

```

```

## # A tibble: 1 x 2
##   cost_complexity .config
##           <dbl> <chr>
## 1      0.00464 Preprocessor1_Model04

```

```

arrange(collect_metrics(tune_res), cost_complexity)

```

```

## # A tibble: 10 x 7
##   cost_complexity .metric .estimator  mean     n std_err .config
##           <dbl> <chr>   <chr>    <dbl> <int>   <dbl> <chr>
## 1      0.001   roc_auc hand_till 0.674     5  0.0112 Preprocessor1_Model01
## 2      0.00167 roc_auc hand_till 0.674     5  0.0112 Preprocessor1_Model02

```



```

class_for_spec <- rand_forest() %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")
class_for_wf <- workflow() %>%
  add_model(class_for_spec %>%
    set_args(mtry = tune(), trees = tune(), min_n = tune())) %>%
  add_recipe(pokemon_recipe)

param_grid_for <- grid_regular(mtry(range = c(1,8)), trees(range = c(1,5)),
                               min_n(range = c(3,10)), levels = 8)

```

mtry is number of variables to possibly split at in each node.

trees is number of trees in the model.

min\_n is the minimal node size.

mtry should not be between 1 and 8 because there are only 8 predictor variables in our recipe. mtry 8 would be a model that uses all 8 of our predictor variables.

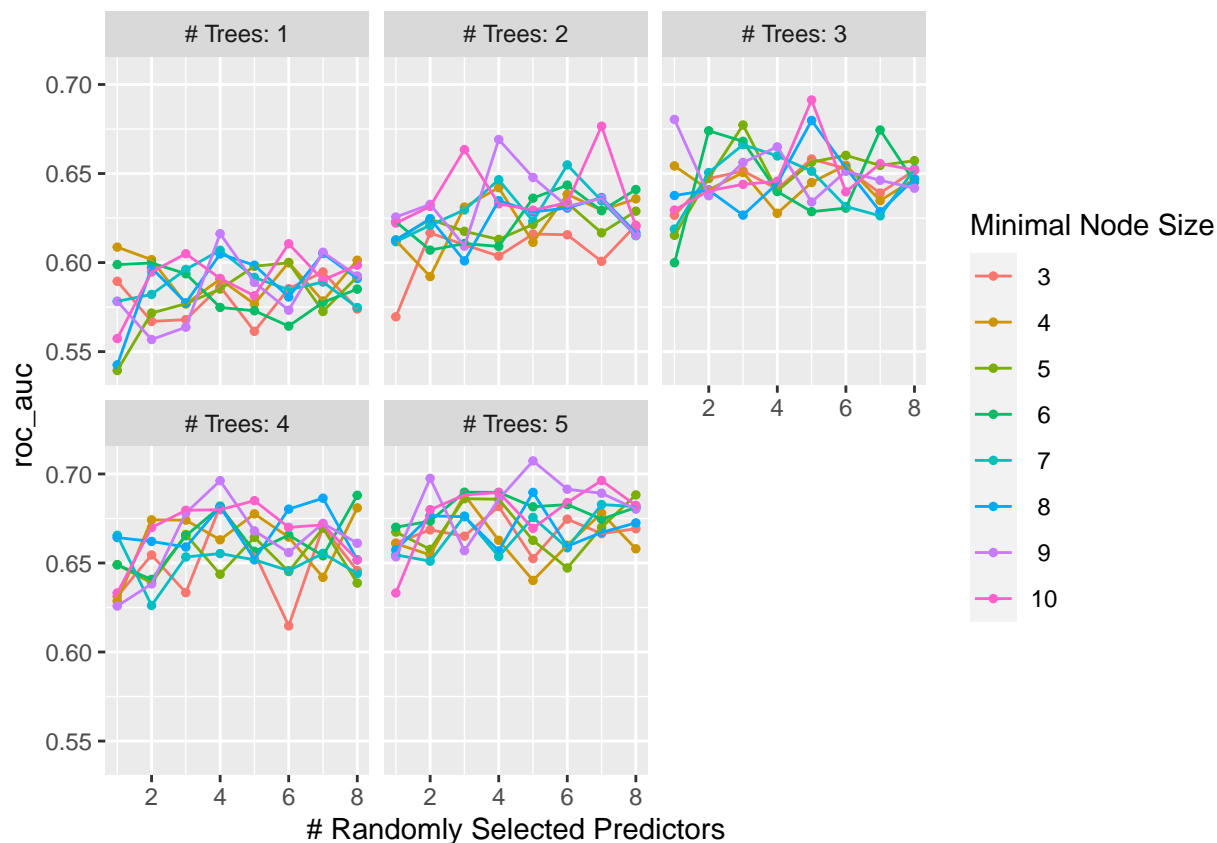
Exercise 6:

```

tune_res_for <- tune_grid(
  class_for_wf,
  resamples = pokemon_folds,
  grid = param_grid_for,
  metrics = metric_set(roc_auc))

autoplot(tune_res_for)

```



Exercise 7:

```
best_for <- select_best(tune_res_for)
best_for
```

```
## # A tibble: 1 x 4
##   mtry trees min_n .config
##   <int> <int> <int> <chr>
## 1     5     5     9 Preprocessor1_Model277
```

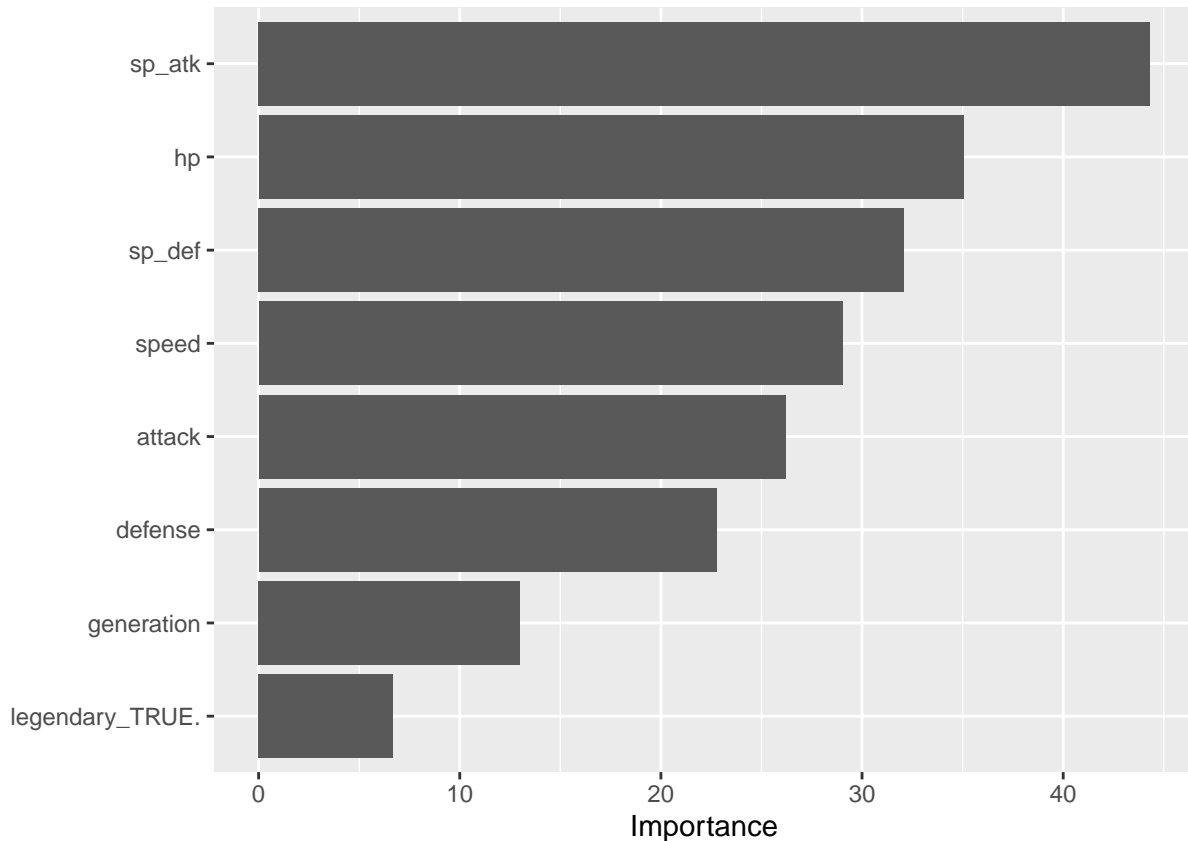
```
arrange(collect_metrics(tune_res_for), mtry, trees, min_n)
```

```
## # A tibble: 320 x 9
##   mtry trees min_n .metric .estimator mean      n std_err .config
##   <int> <int> <int> <chr>   <chr>    <dbl> <int>   <dbl> <chr>
## 1     1     1     3 roc_auc hand_till 0.589     5 0.0150 Preprocessor1_Model~
## 2     1     1     4 roc_auc hand_till 0.609     5 0.0139 Preprocessor1_Model~
## 3     1     1     5 roc_auc hand_till 0.539     5 0.00782 Preprocessor1_Model~
## 4     1     1     6 roc_auc hand_till 0.599     5 0.0170 Preprocessor1_Model~
## 5     1     1     7 roc_auc hand_till 0.578     5 0.0237 Preprocessor1_Model~
## 6     1     1     8 roc_auc hand_till 0.543     5 0.00932 Preprocessor1_Model~
## 7     1     1     9 roc_auc hand_till 0.578     5 0.0235 Preprocessor1_Model~
## 8     1     1    10 roc_auc hand_till 0.557     5 0.0104 Preprocessor1_Model~
## 9     1     2     3 roc_auc hand_till 0.570     5 0.0158 Preprocessor1_Model~
## 10    1     2     4 roc_auc hand_till 0.612     5 0.0130 Preprocessor1_Model~
## # ... with 310 more rows
```

The ROC AUC is 0.7073355

Exercise 8:

```
class_for_final <- finalize_workflow(class_for_wf, best_for)
class_for_final_fit <- fit(class_for_final, data = pokemon_train)
class_for_final_fit %>%
  extract_fit_parsnip() %>%
  vip()
```



sp\_attack and hp were the most useful, while generation and legendary were the least useful. Yes, I assume that you need to have successful sp\_attack and hp make for the 'best' Pokemon.

Exercise 9:

```
boost_spec <- boost_tree() %>%
  set_engine("xgboost") %>%
  set_mode("classification")
boost_wf <- workflow() %>%
  add_model(boost_spec %>%
    set_args(trees = tune())) %>%
  add_recipe(pokemon_recipe)
param_grid_boost <- grid_regular(trees(range = c(10,2000)), levels = 10)

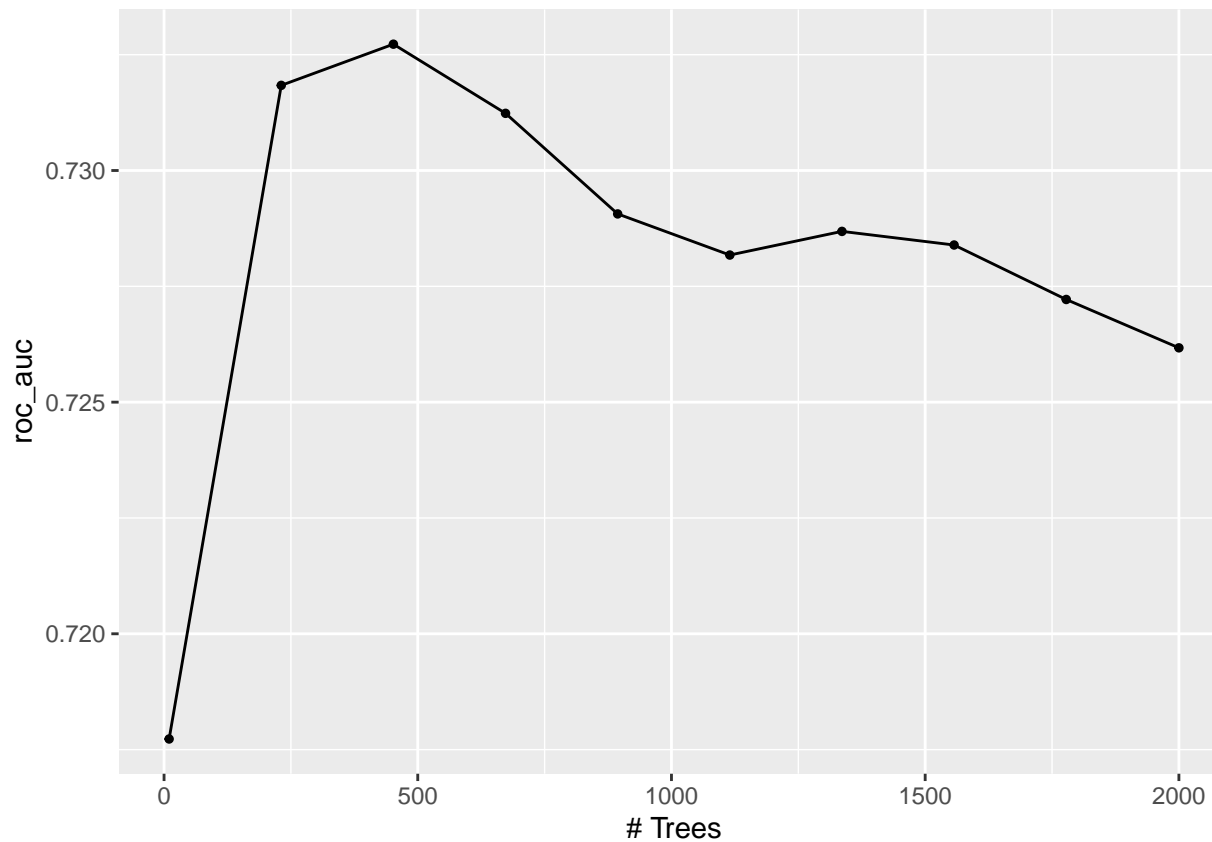
tune_res_boost <- tune_grid(
  boost_wf,
```

```

resamples = pokemon_folds,
grid = param_grid_boost,
metrics = metric_set(roc_auc))

autoplot(tune_res_boost)

```



```

best_boost <- select_best(tune_res_boost)
best_boost

```

```

## # A tibble: 1 x 2
##   trees .config
##   <int> <chr>
## 1   452 Preprocessor1_Model103

```

```

arrange(collect_metrics(tune_res_boost), trees)

```

```

## # A tibble: 10 x 7
##   trees .metric .estimator mean      n std_err .config
##   <int> <chr>    <chr>    <dbl> <int>  <dbl> <chr>
## 1    10 roc_auc hand_till  0.718     5 0.00725 Preprocessor1_Model101
## 2   231 roc_auc hand_till  0.732     5 0.00802 Preprocessor1_Model102
## 3   452 roc_auc hand_till  0.733     5 0.00728 Preprocessor1_Model103
## 4   673 roc_auc hand_till  0.731     5 0.00779 Preprocessor1_Model104
## 5   894 roc_auc hand_till  0.729     5 0.00856 Preprocessor1_Model105

```



```
## 6 1115 roc_auc hand_till 0.728 5 0.00889 Preprocessor1_Model106
## 7 1336 roc_auc hand_till 0.729 5 0.00929 Preprocessor1_Model107
## 8 1557 roc_auc hand_till 0.728 5 0.00913 Preprocessor1_Model108
## 9 1778 roc_auc hand_till 0.727 5 0.00877 Preprocessor1_Model109
## 10 2000 roc_auc hand_till 0.726 5 0.00860 Preprocessor1_Model110
```

The ROC-AUC of the best model is 0.7327269

```
compl <- collect_metrics(tune_res)
roc_compl <- subset(compl, .config=='Preprocessor1_Model108')[,4]
roc_compl$type <- c('pruned tree')

forest <- collect_metrics(tune_res_for)
roc_forest <- subset(forest, .config == 'Preprocessor1_Model118')[,6]
roc_forest$type <- c('random forest')

boost <- collect_metrics(tune_res_boost)
roc_boost <- subset(boost, .config == 'Preprocessor1_Model101')[,4]
roc_boost$type <- c('boosted tree')

x<-rbind(roc_compl, roc_forest, roc_boost )
table <- x[,c(2,1)] %>% rename(roc_auc = mean)
table
```

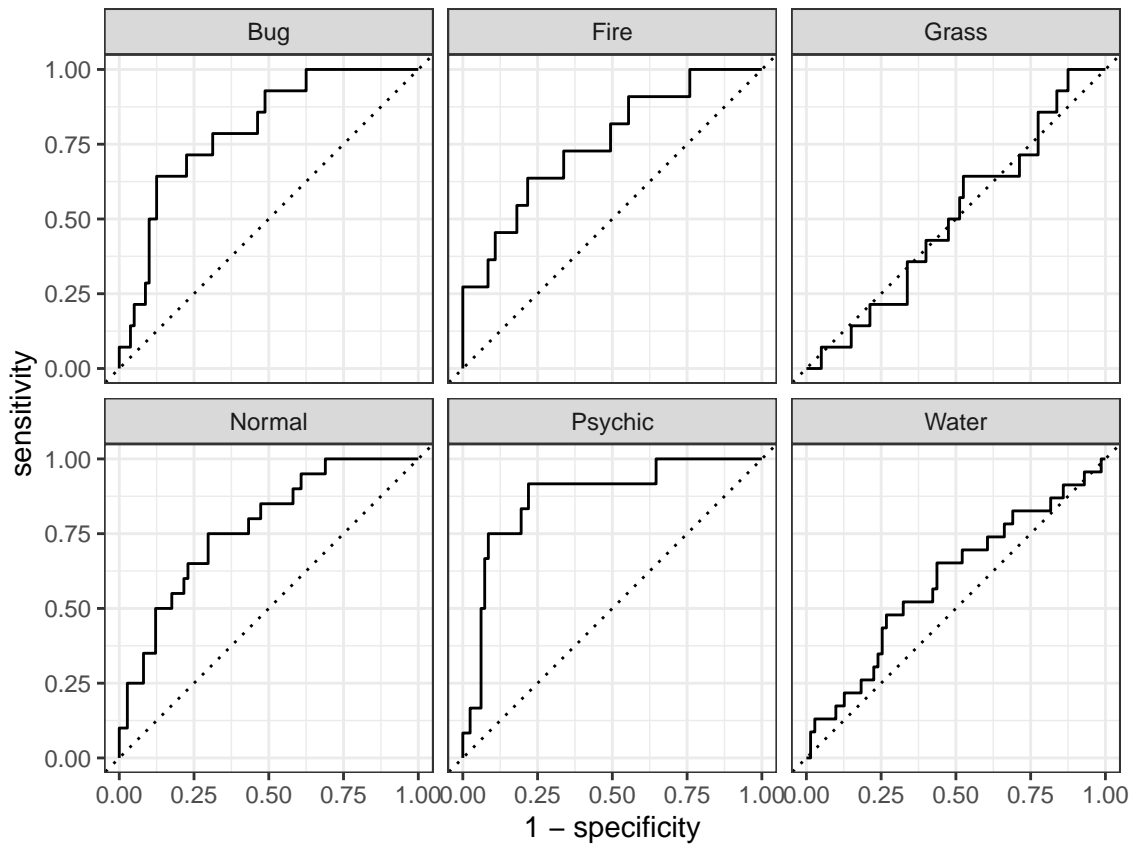
```
## # A tibble: 3 x 2
##   type      roc_auc
##   <chr>      <dbl>
## 1 pruned tree 0.638
## 2 random forest 0.647
## 3 boosted tree 0.718
```

```
best_final <- select_best(tune_res_boost)
final_wf <- finalize_workflow(boost_wf, best_final)
final_fit <- fit(final_wf, data = pokemon_train)

augm <- augment(final_fit, new_data = pokemon_test)
augm %>% roc_auc(truth = type_1, estimate = c(
  .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 0.715
```

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



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

Prediction	Bug -	8	0	3	2	1	2
	Fire -	0	4	2	5	0	3
	Grass -	0	2	1	0	3	5
	Normal -	3	1	2	10	1	6
	Psychic -	0	0	3	1	5	1
	Water -	3	4	3	2	2	6
		Bug	Fire	Grass	Normal	Psychic	Water
		Truth					