Homework 6

PSTAT 131/231

Contents

Tree-Based Models

The goal of this assignment is to build a statistical learning model that can predict the **primary type** of a Pokémon based on its generation, legendary status, and six battle statistics.

Note: Fitting ensemble tree-based models can take a little while to run. Consider running your models outside of the .Rmd, storing the results, and loading them in your .Rmd to minimize time to knit.

```
library(tidymodels)
library(tidyverse)
library(ISLR) # For the Smarket data set
library(ISLR2) # For the Bikeshare data set
library(discrim)
library(poissonreg)
library(corrr)
library(klaR) # for naive bayes
library(forcats)
library(corrplot)
library(pROC)
library(recipes)
library(rsample)
library(parsnip)
library(workflows)
library(janitor)
library(glmnet)
library(rpart.plot)
library(vip)
library(janitor)
library(randomForest)
library(xgboost)
tidymodels_prefer()
```

Exercise 1

Read in the data and set things up as in Homework 5:

- Use `clean_names()`
- Filter out the rarer Pokémon types
- Convert `type_1` and `legendary` to factors

Do an initial split of the data; you can choose the percentage for splitting. Stratify on the outcome variable.

Fold the training set using *v*-fold cross-validation, with `v = 5`. Stratify on the outcome variable.

Set up a recipe to predict `type_1` with `legendary`, `generation`, `sp_atk`, `attack`, `speed`, `defense`, `hp`, and `sp_def`:

- Dummy-code `legendary` and `generation`;
- Center and scale all predictors.

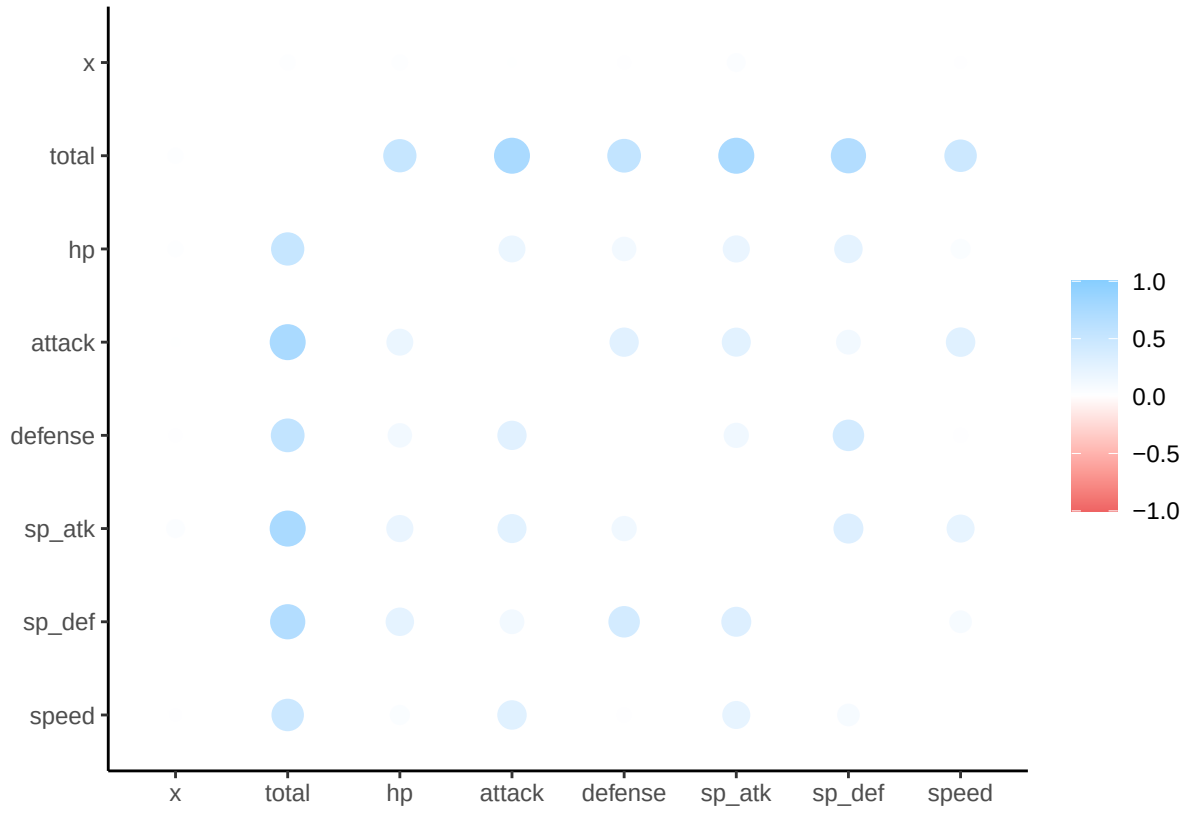
```
pk_data <- read.csv(file = "pokemon.csv")

pk_data <- clean_names(pk_data)

#filter
ft_data <- filter(pk_data, type_1 == "Bug" | type_1 == "Fire" | type_1 == "Grass" | type_1 == "Normal"
ft_data$type_1 = as.factor(ft_data$type_1)
ft_data$legendary = as.factor(ft_data$legendary)

set.seed(1000)

pk_split <- initial_split(ft_data, prop = 0.75, strata = type_1)
pk_train <- training(pk_split)
pk_test <- testing(pk_split)

pk_fold <- vfold_cv(pk_train, v = 5, strata = type_1)


recipe_pk <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_def,
  step_dummy(legendary) %>%
  step_dummy(generation) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())
```

Exercise 2

Create a correlation matrix of the training set, using the `corrplot` package. *Note: You can choose how to handle the continuous variables for this plot; justify your decision(s).*

What relationships, if any, do you notice? Do these relationships make sense to you?

```
cor_titanic_train <- pk_train %>%
  select(is.numeric) %>%
  correlate()
rplot(cor_titanic_train)
```



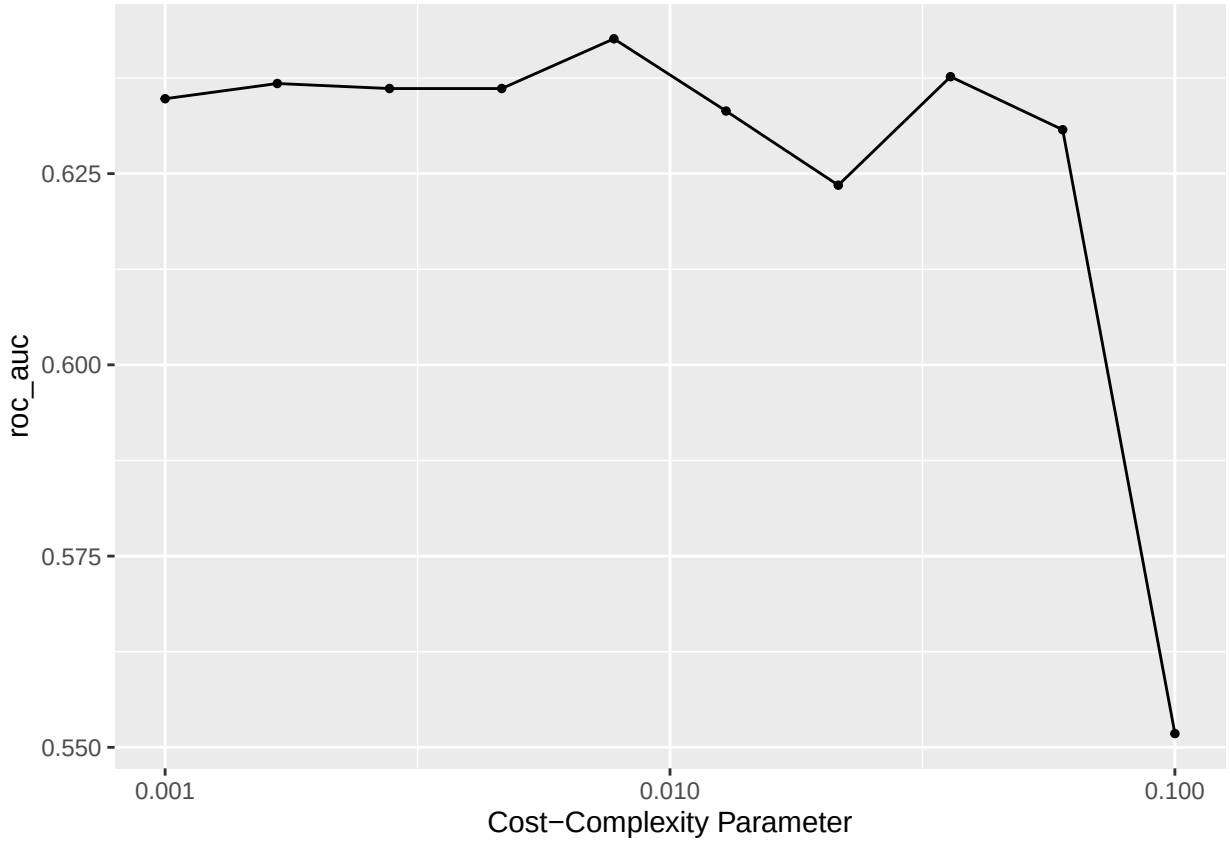
Exercise 3

First, set up a decision tree model and workflow. Tune the `cost_complexity` hyperparameter. Use the same levels we used in Lab 7 – that is, `range = c(-3, -1)`. Specify that the metric we want to optimize is `roc_auc`.

Print an `autoplot()` of the results. What do you observe? Does a single decision tree perform better with a smaller or larger complexity penalty?

```
tree_pk <- decision_tree() %>%
  set_engine("rpart")

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

class_tree_wkflow <- workflow() %>%
  add_model(class_tree_spec %>%
              set_args(cost_complexity = tune())) %>%
  add_recipe(recipe_pk)

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

# optimize roc_auc
tune_res <- tune_grid(
  class_tree_wkflow,
```

```
  resamples = pk_fold,
  grid = param_grid,
  metrics = metric_set(roc_auc)
)

autoplot(tune_res)
```



Exercise 4

What is the `roc_auc` of your best-performing pruned decision tree on the folds? *Hint: Use `collect_metrics()` and `arrange()`.*

```
collect_metrics(tune_res) %>%
  arrange(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  0.635     5 0.0171  Preprocessor1_Model01
##  2         0.00167 roc_auc hand_till  0.637     5 0.0178  Preprocessor1_Model02
##  3         0.00278 roc_auc hand_till  0.636     5 0.0178  Preprocessor1_Model03
##  4         0.00464 roc_auc hand_till  0.636     5 0.0178  Preprocessor1_Model04
##  5         0.00774 roc_auc hand_till  0.643     5 0.0191  Preprocessor1_Model05
##  6         0.0129  roc_auc hand_till  0.633     5 0.0186  Preprocessor1_Model06
```

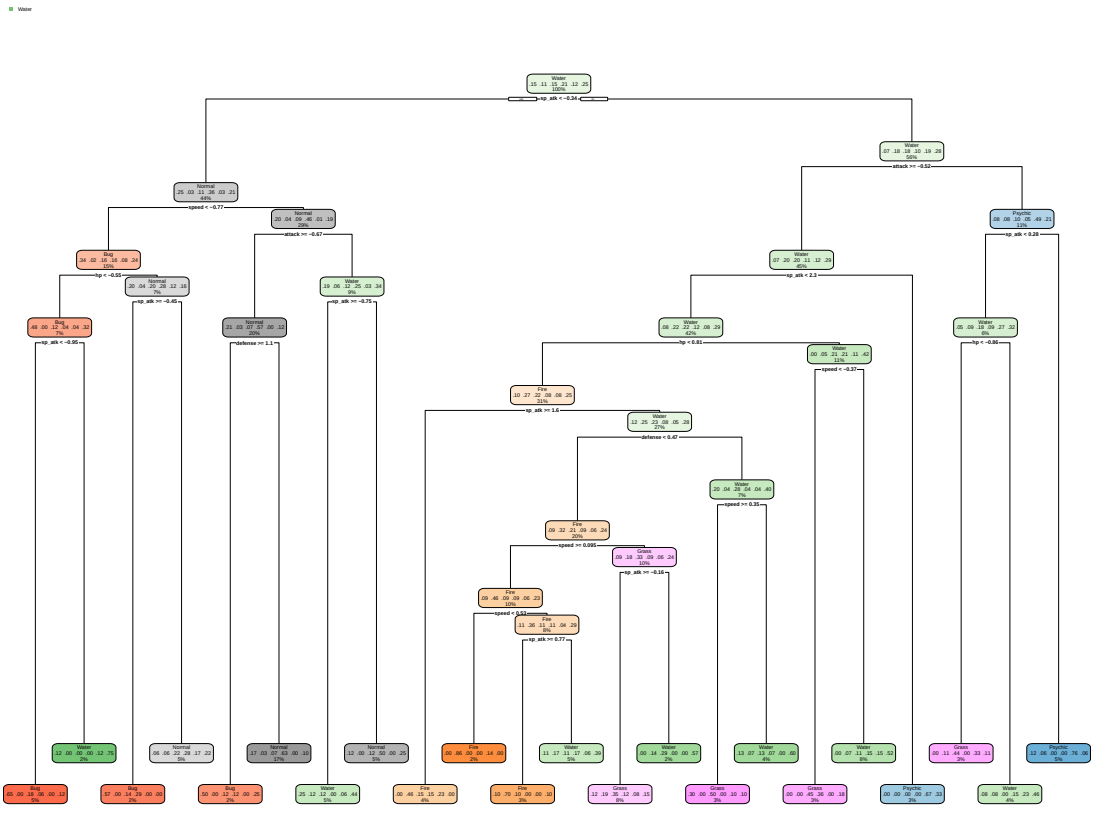
```
##  7         0.0215  roc_auc hand_till  0.623     5 0.00810 Preprocessor1_Model07
##  8         0.0359  roc_auc hand_till  0.638     5 0.0147  Preprocessor1_Model08
##  9         0.0599  roc_auc hand_till  0.631     5 0.00977 Preprocessor1_Model09
## 10         0.1     roc_auc hand_till  0.552     5 0.0317  Preprocessor1_Model10
```

```
best_complexity <- select_best(tune_res)
```

Exercise 5

Using `rpart.plot`, fit and visualize your best-performing pruned decision tree with the *training* set.

```
best_pk <- finalize_workflow(class_tree_wkflow, best_complexity)
best_pk_fit <- fit(best_pk, pk_train)


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



Exercise 5

Now set up a random forest model and workflow. Use the `ranger` engine and set `importance = "impurity"`. Tune `mtry`, `trees`, and `min_n`. Using the documentation for `rand_forest()`, explain in your own words what each of these hyperparameters represent.

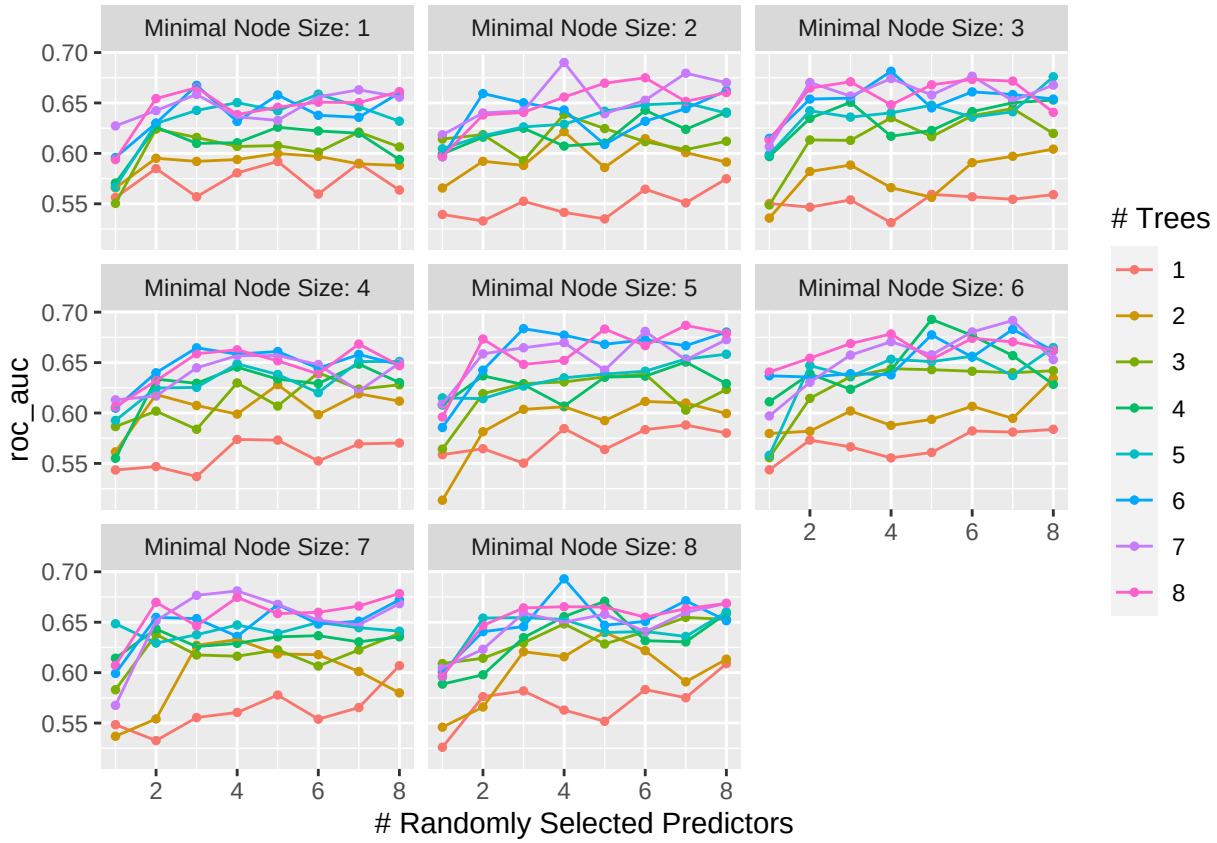
Create a regular grid with 8 levels each. You can choose plausible ranges for each hyperparameter. Note that `mtry` should not be smaller than 1 or larger than 8. **Explain why not. What type of model would `mtry = 8` represent?**

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

rfm_wkflow <- workflow() %>%
  add_model(rfm_spec) %>%
  add_recipe(recipe_pk)

param_grid2 <- grid_regular(mtry(range = c(1, 8)),
                            trees(range = c(1, 8)),
                            min_n(range = c(1, 8)),
                            levels = 8)
```

Exercise 6

Specify `roc_auc` as a metric. Tune the model and print an `autoplot()` of the results. What do you observe? What values of the hyperparameters seem to yield the best performance?

```
tune_res <- tune_grid(
  rfm_wkflow,
  resamples = pk_fold,
  grid = param_grid2,
  metrics = metric_set(roc_auc)
)

# print results
autoplot(tune_res)
```



Exercise 7

What is the `roc_auc` of your best-performing random forest model on the folds? *Hint: Use `collect_metrics()` and `arrange()`.*

```
arrange(collect_metrics(tune_res),desc(mean))
```

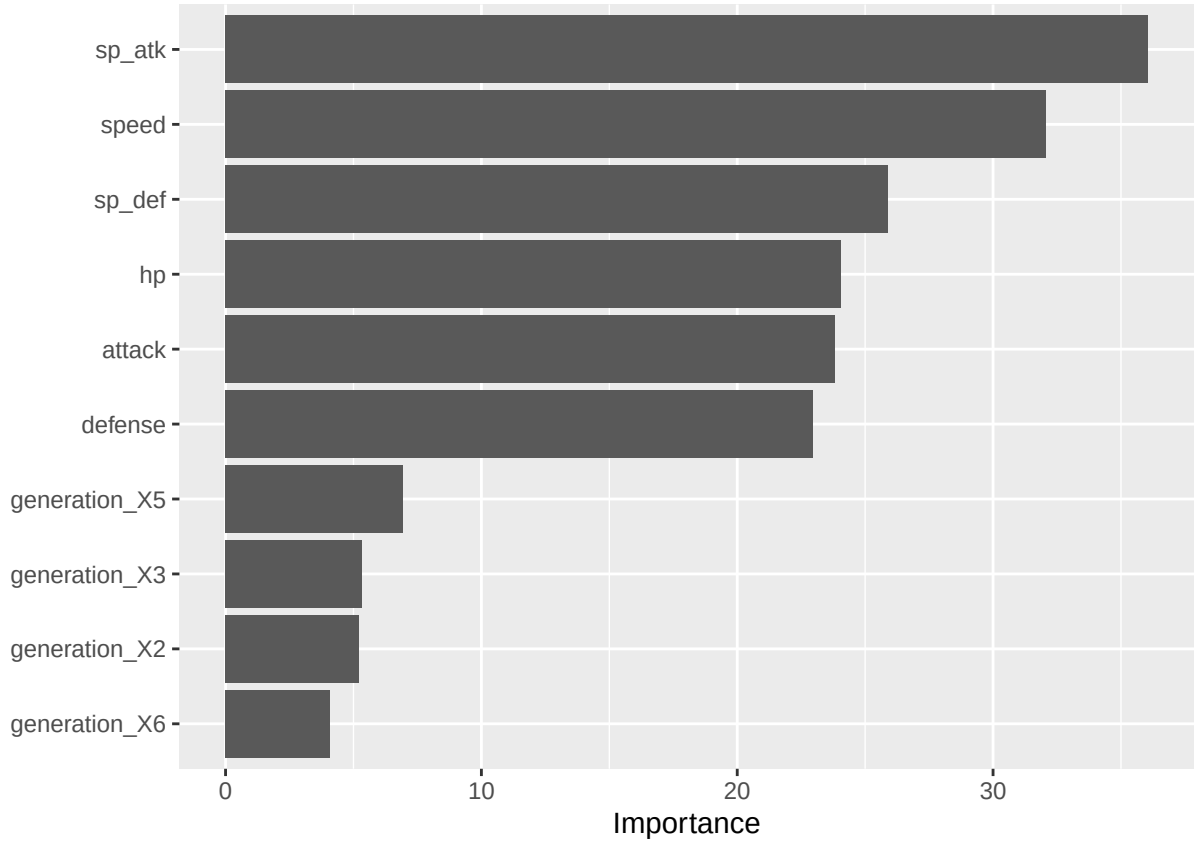
```
## # A tibble: 512 x 9
##     mtry trees min_n .metric .estimator  mean     n std_err .config
##    <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     4     6     8 roc_auc hand_till  0.693     5 0.0195  Preprocessor1_Model~
##  2     5     4     6 roc_auc hand_till  0.693     5 0.0191  Preprocessor1_Model~
##  3     7     7     6 roc_auc hand_till  0.692     5 0.00990 Preprocessor1_Model~
##  4     4     7     2 roc_auc hand_till  0.690     5 0.0120  Preprocessor1_Model~
##  5     7     8     5 roc_auc hand_till  0.687     5 0.0111  Preprocessor1_Model~
##  6     3     6     5 roc_auc hand_till  0.684     5 0.0126  Preprocessor1_Model~
##  7     5     8     5 roc_auc hand_till  0.683     5 0.00777 Preprocessor1_Model~
##  8     7     6     6 roc_auc hand_till  0.683     5 0.0162  Preprocessor1_Model~
##  9     4     6     3 roc_auc hand_till  0.681     5 0.0103  Preprocessor1_Model~
## 10     4     7     7 roc_auc hand_till  0.681     5 0.0128  Preprocessor1_Model~
## # ... with 502 more rows
```

Exercise 8

Create a variable importance plot, using `vip()`, with your best-performing random forest model fit on the *training* set.

Which variables were most useful? Which were least useful? Are these results what you expected, or not?

```
best_complexity <- select_best(tune_res, metric = "roc_auc")
pk_final <- finalize_workflow(rfm_wkflow, best_complexity)

rfm_fit <- fit(pk_final,data = pk_train)
rfm_fit %>%
  extract_fit_engine() %>%
  vip()
```



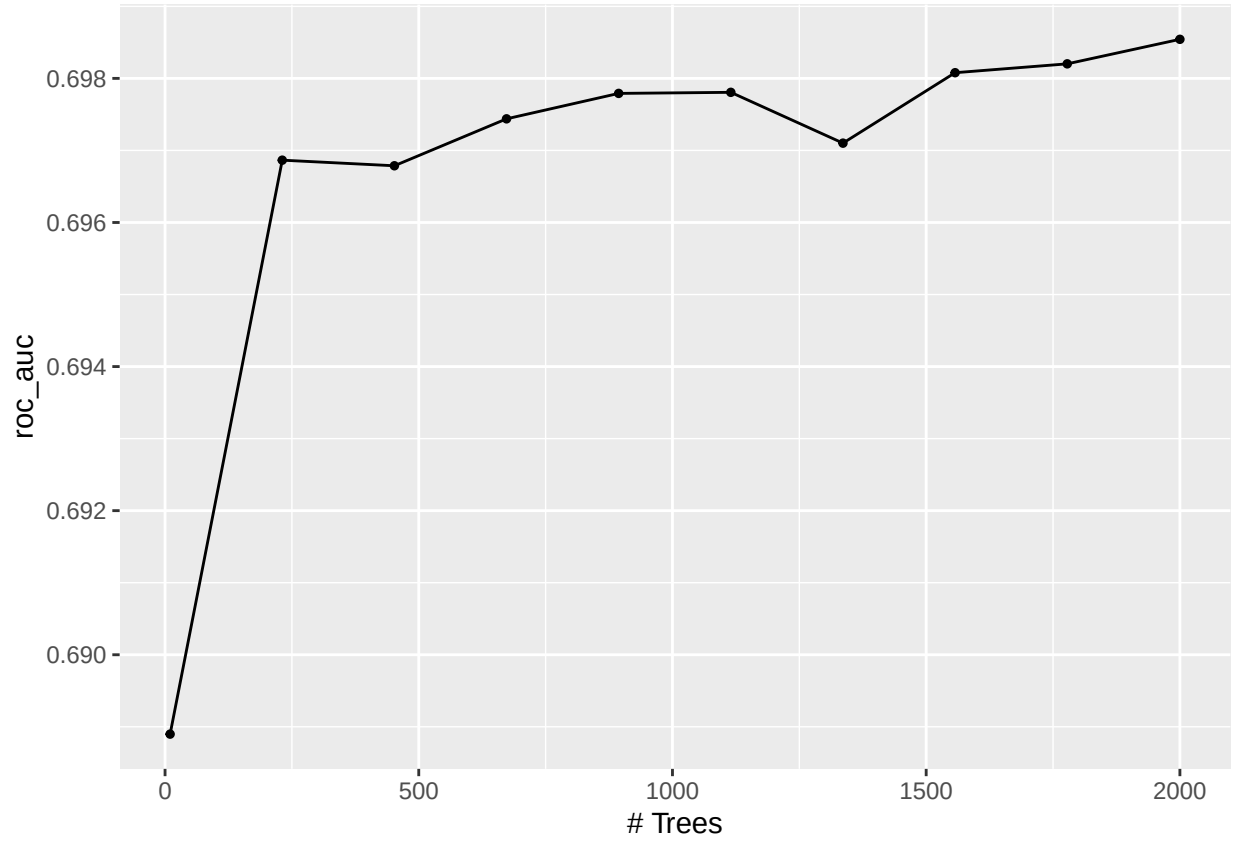
Exercise 9

Finally, set up a boosted tree model and workflow. Use the `xgboost` engine. Tune `trees`. Create a regular grid with 10 levels; let `trees` range from 10 to 2000. Specify `roc_auc` and again print an `autoplot()` of the results.

What do you observe?

What is the `roc_auc` of your best-performing boosted tree model on the folds? *Hint: Use `collect_metrics()` and `arrange()`.*

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


boost_tree_grid <- grid_regular(trees(c(10,2000)),levels = 10)


boost_tree_wkflow <- workflow() %>%
  add_model(boost_tree_pk) %>%
  add_formula(type_1 ~ sp_atk + attack + speed +
                defense + hp + sp_def + legendary + generation)

boost_tune_res <- tune_grid(
  boost_tree_wkflow,
  resamples = pk_fold,
  grid = boost_tree_grid,
  metrics = metric_set(roc_auc),
)

autoplot(boost_tune_res)
```




```
arrange(collect_metrics(boost_tune_res),desc(mean))
```

```
## # A tibble: 10 x 7
```

```
##    trees .metric .estimator  mean     n std_err .config
##    <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1  2000 roc_auc hand_till  0.699     5 0.00451 Preprocessor1_Model10
##  2  1778 roc_auc hand_till  0.698     5 0.00427 Preprocessor1_Model09
##  3  1557 roc_auc hand_till  0.698     5 0.00457 Preprocessor1_Model08
##  4  1115 roc_auc hand_till  0.698     5 0.00515 Preprocessor1_Model06
##  5   894 roc_auc hand_till  0.698     5 0.00580 Preprocessor1_Model05
##  6   673 roc_auc hand_till  0.697     5 0.00641 Preprocessor1_Model04
##  7  1336 roc_auc hand_till  0.697     5 0.00484 Preprocessor1_Model07
##  8   231 roc_auc hand_till  0.697     5 0.00818 Preprocessor1_Model02
##  9   452 roc_auc hand_till  0.697     5 0.00712 Preprocessor1_Model03
## 10    10 roc_auc hand_till  0.689     5 0.0107  Preprocessor1_Model01
```

Exercise 10

Display a table of the three ROC AUC values for your best-performing pruned tree, random forest, and boosted tree models. Which performed best on the folds? Select the best of the three and use `select_best()`, `finalize_workflow()`, and `fit()` to fit it to the *testing* set.

Print the AUC value of your best-performing model on the testing set. Print the ROC curves. Finally, create and visualize a confusion matrix heat map.

Which classes was your model most accurate at predicting? Which was it worst at?

```
best_boost_tree <- select_best(boost_tune_res)
boost_tree_final <- finalize_workflow(boost_tree_wkflow, best_boost_tree)
boost_tree_final_fit <- fit(boost_tree_final, data = pk_train)


final_class_model = augment(best_pk_fit, new_data = pk_train)
final_rand_model = augment(rfm_fit, new_data = pk_train)
final_boost_model = augment(boost_tree_final_fit, new_data = pk_train)
results<- bind_rows(
  roc_auc(final_class_model, truth = type_1, .pred_Bug, .pred_Fire, .pred_Grass,
          .pred_Normal, .pred_Water, .pred_Psychic),
  roc_auc(final_rand_model, truth = type_1, .pred_Bug, .pred_Fire, .pred_Grass,
          .pred_Normal, .pred_Water, .pred_Psychic),
  roc_auc(final_boost_model, truth = type_1, .pred_Bug, .pred_Fire, .pred_Grass,
          .pred_Normal, .pred_Water, .pred_Psychic)
)
results
```

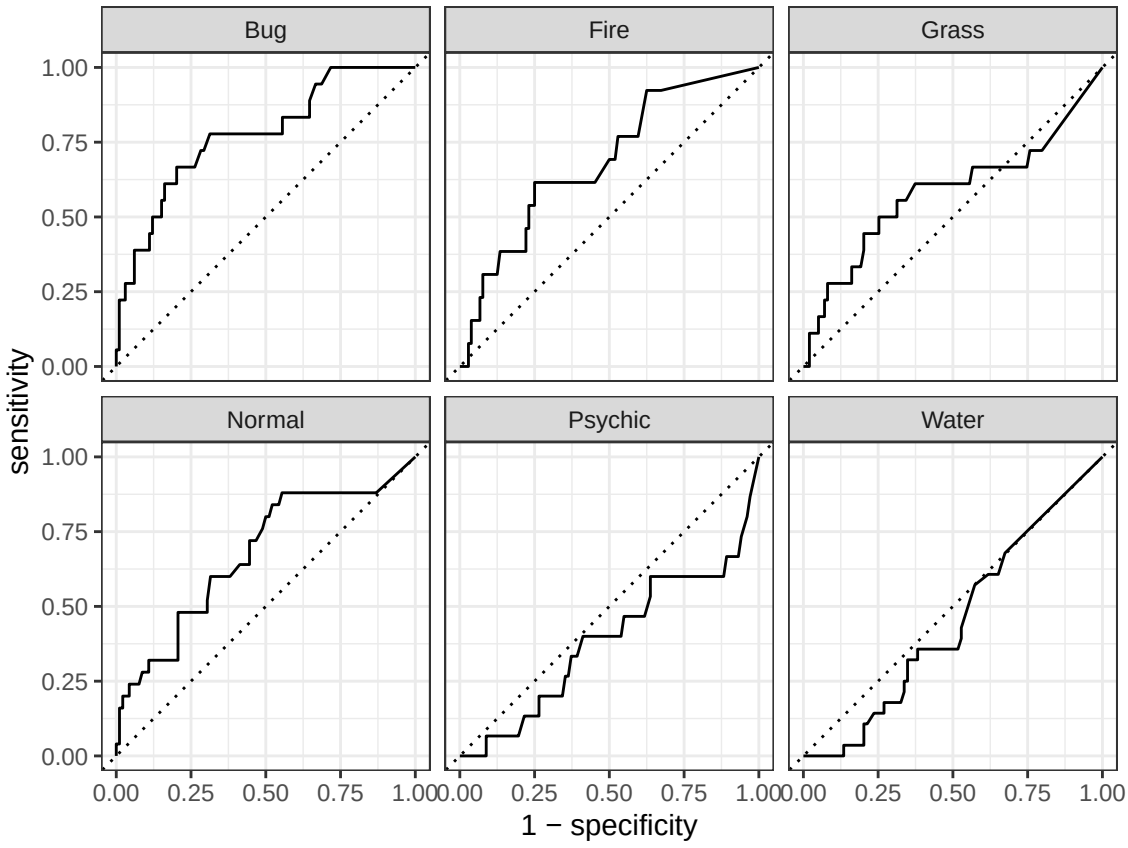
```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.708
## 2 roc_auc hand_till      0.795
## 3 roc_auc hand_till      0.809
```

```
final_rand_model_test = augment(rfm_fit, new_data = pk_test)
roc_auc(final_rand_model_test, truth = type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_
```

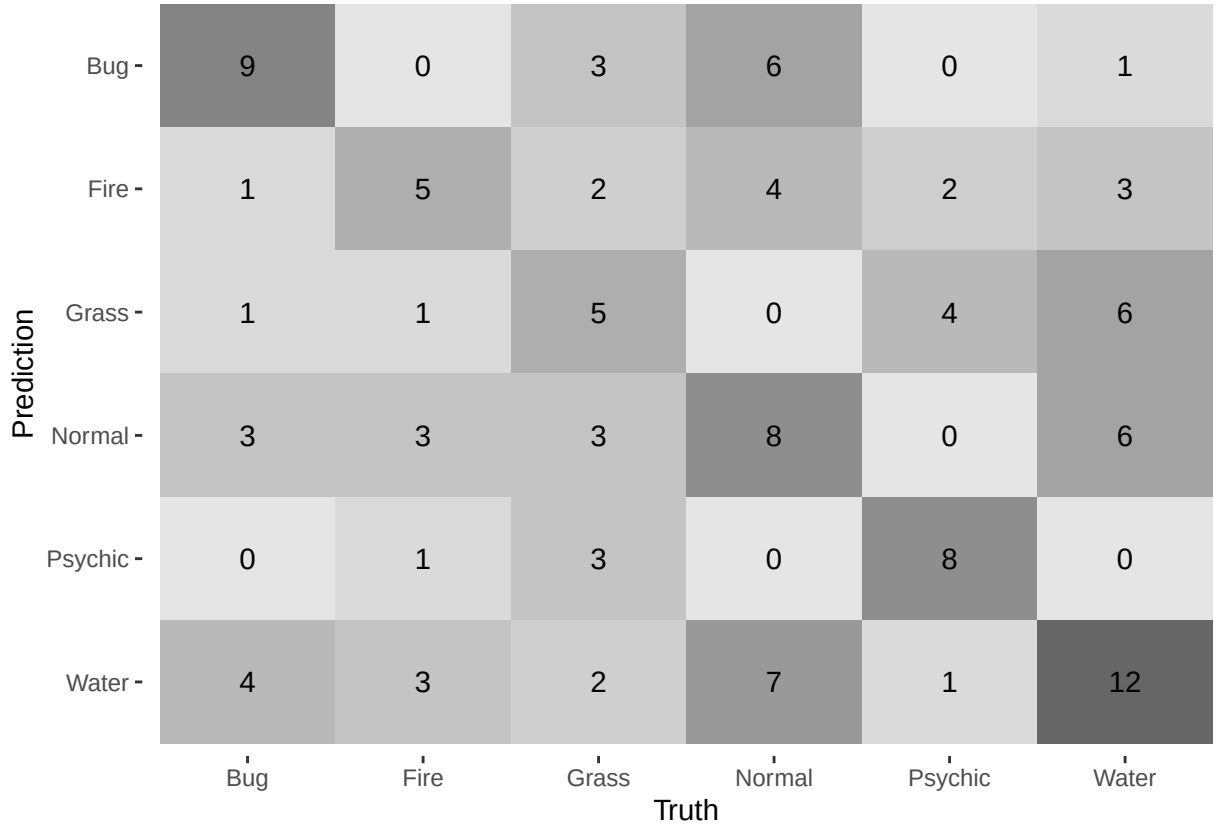
```
## # A tibble: 1 x 3
```

```
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.590
```

```
autoplot(roc_curve(final_rand_model_test, truth = type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Nor
```




```
conf_mat(final_rand_model_test, truth = type_1, estimate = .pred_class) %>% #calclate confusion matri
  autoplot(type = "heatmap")
```



For 231 Students

Exercise 11

Using the `abalone.txt` data from previous assignments, fit and tune a random forest model to predict `age`. Use stratified cross-validation and select ranges for `mtry`, `min_n`, and `trees`. Present your results. What was the model's RMSE on your testing set?