Homework 6

PSTAT 131/231

Contents

Tree-Based Models 1

Tree-Based Models

For this assignment, we will continue working with the file `"pokemon.csv"`.

Each Pokémon has a primary type. Based on their type, a Pokémon is strong against some types, and vulnerable to others. (Think rock, paper, scissors.) A Fire-type Pokémon, for example, is vulnerable to Water-type Pokémon, but strong against Grass-type.

```r
Pokemon <- read.csv("Pokemon.csv", header=TRUE)
head(Pokemon)
```

```
##   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
```

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.

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.

Answer :

```
library(tidyverse)
library(tidymodels)
library(rpart.plot)
library(vip)
library(janitor)


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

```
#str(Pokemon$type_1)
length(unique(Pokemon$type_1))
```

```
## [1] 18
```

```
filterdf <- Pokemon %>% filter(type_1%in% c("Bug", "Fire", "Grass", "Normal","Water","Psychic" )) %>%
            mutate(type_1 = factor(type_1),
                  legendary = factor(legendary))
```

```
filterdf_split <- filterdf %>%
  initial_split(strata = type_1, prop = 0.7)
filterdf_train <- training(filterdf_split)
filterdf_test <- testing(filterdf_split)
dim(filterdf_train)
```

```
## [1] 318  13
```

```
dim(filterdf_test)
```

```
## [1] 140  13
```

```
Auto_folds <-  vfold_cv(filterdf_train, strata = type_1, v = 5)
Auto_folds
```

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 x 2
##   splits           id
##   <list>           <chr>
## 1 <split [252/66]> Fold1
## 2 <split [253/65]> Fold2
## 3 <split [253/65]> Fold3
## 4 <split [256/62]> Fold4
## 5 <split [258/60]> Fold5
```

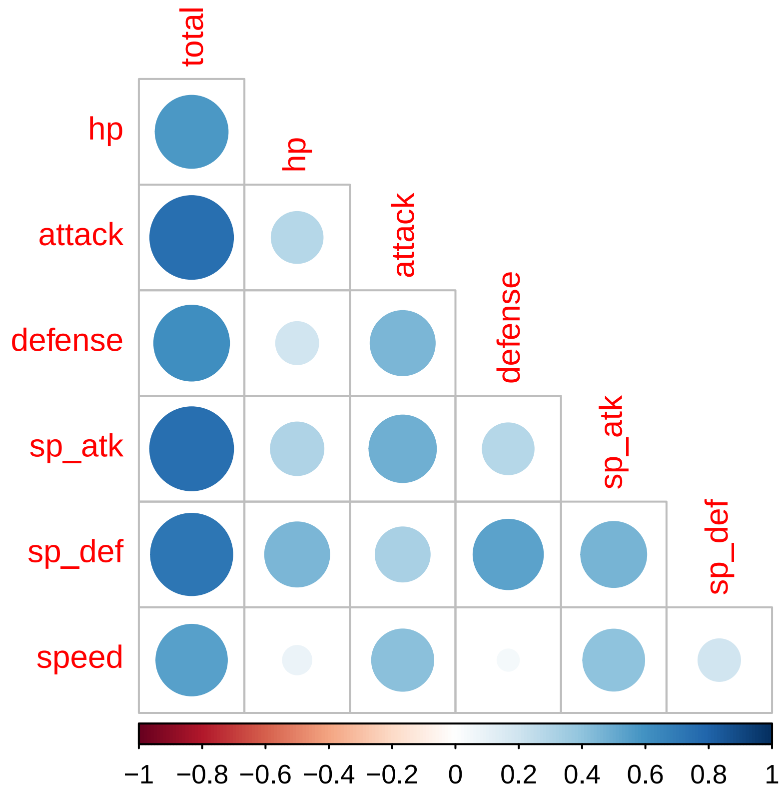
```
filterdf_recipe <- recipe(type_1 ~ legendary+ generation + sp_atk +
                           attack + speed+ hp+sp_def, filterdf_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())
```

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?

```
library(corrplot)
library(corrr)

filterdf_train %>%
  select(is.numeric, -x, -generation) %>%
  cor(use = "complete.obs") %>%
  corrplot(type = "lower", diag = FALSE)
```



special attack is positively correlated with damage resistance against special attacks and Speed is negativly correlated with defense and hit point

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`.

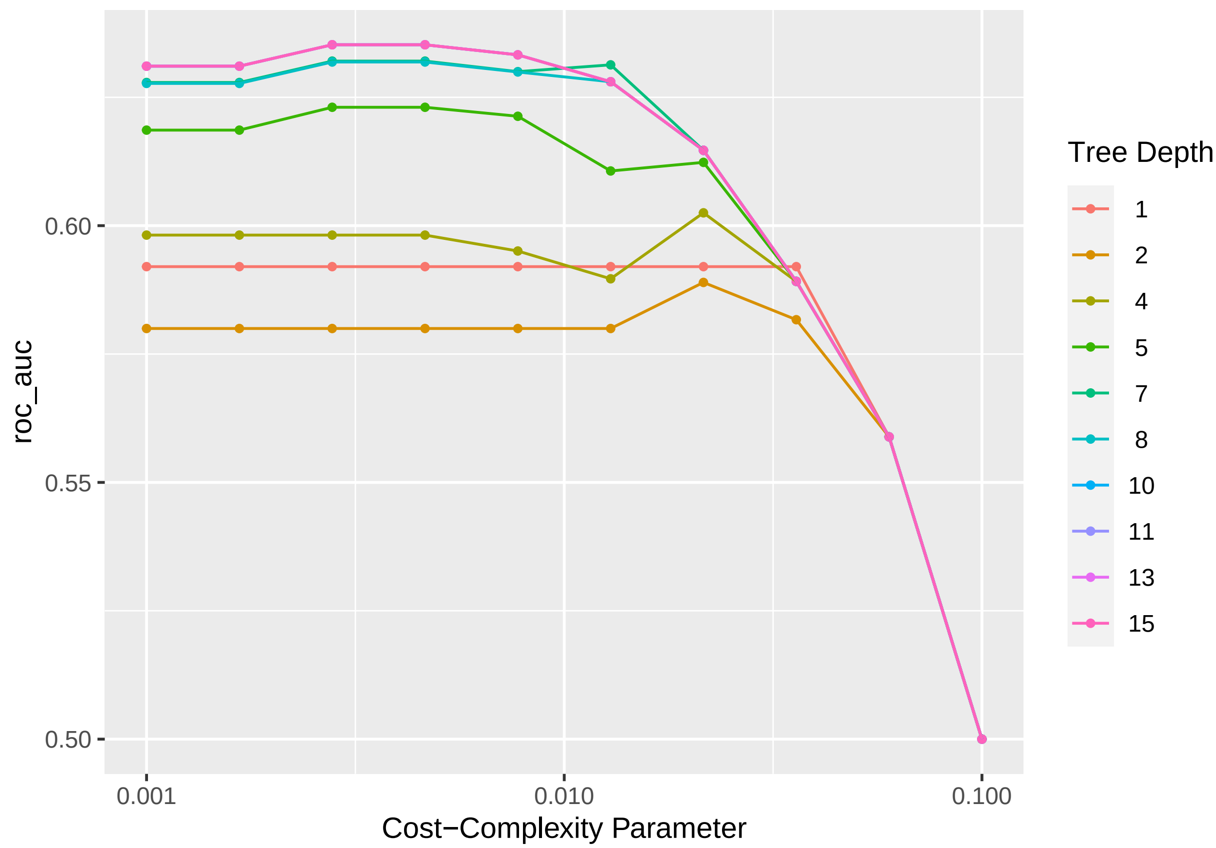
Print an `autoplot()` of the results. What do you observe? Does a single decision tree perform better with a smaller or larger complexity penalty?

Answer Decision tree performs better with small complexity penalty

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

```
class_tree_wf <- workflow() %>%
  add_model(tree_mod) %>%
  add_recipe(filterdf_recipe)
```

```r
set.seed(3435)

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

tune_res <- tune_grid(
  class_tree_wf,
  resamples = Auto_folds,
  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()`.*

```r
best_tree <- select_best(tune_res)
best_tree
```

```
## # A tibble: 1 x 3
##   cost_complexity tree_depth .config
##             <dbl>      <int> <chr>
## 1         0.00278         10 Preprocessor1_Model063
```

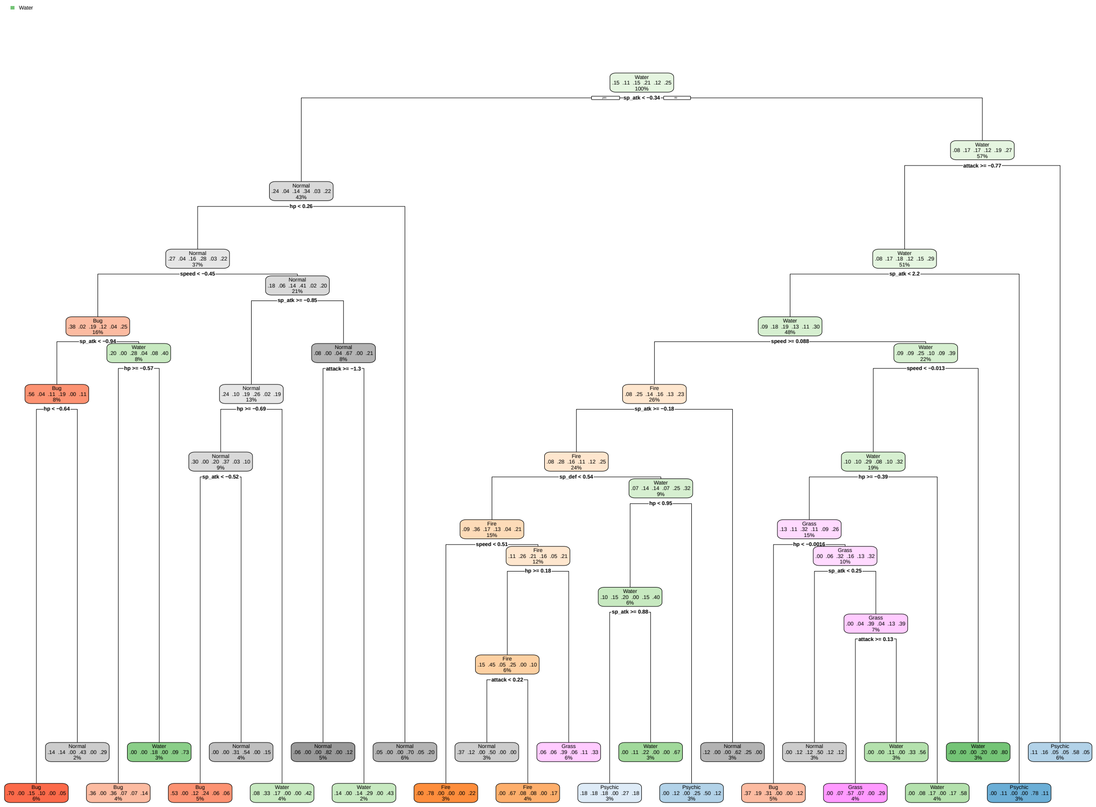
```r
collect_metrics(tune_res, summarize = TRUE) %>% arrange(select_best(tune_res), .by_group = TRUE)
```

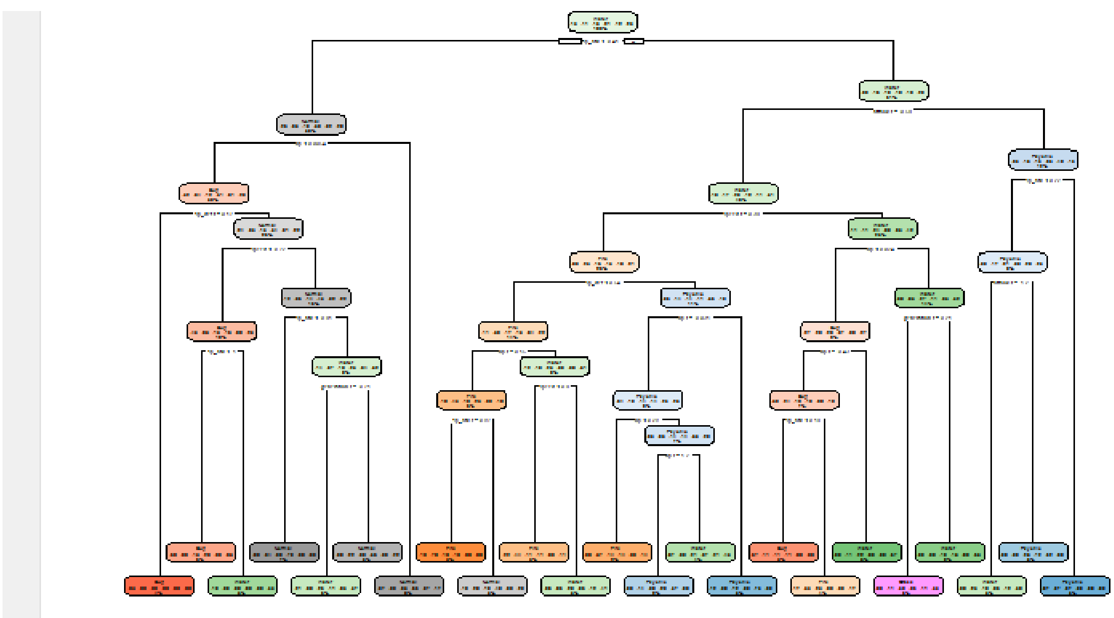
```
## # A tibble: 100 x 8
##    cost_complexity tree_depth .metric .estimator  mean     n std_err .config
##              <dbl>      <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         0.001            1 roc_auc hand_till  0.592     5 0.00976 Preprocess~
##  2         0.00167          1 roc_auc hand_till  0.592     5 0.00976 Preprocess~
##  3         0.00278          1 roc_auc hand_till  0.592     5 0.00976 Preprocess~
##  4         0.00464          1 roc_auc hand_till  0.592     5 0.00976 Preprocess~
##  5         0.00774          1 roc_auc hand_till  0.592     5 0.00976 Preprocess~
##  6         0.0129           1 roc_auc hand_till  0.592     5 0.00976 Preprocess~
##  7         0.0215           1 roc_auc hand_till  0.592     5 0.00976 Preprocess~
##  8         0.0359           1 roc_auc hand_till  0.592     5 0.00976 Preprocess~
##  9         0.0599           1 roc_auc hand_till  0.559     5 0.0255  Preprocess~
## 10         0.1              1 roc_auc hand_till  0.5       5 0       Preprocess~
## # ... with 90 more rows
```

Exercise 5

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

```r
class_tree_final <- finalize_workflow(class_tree_wf, best_tree)

class_tree_final_fit <- fit(class_tree_final, data = filterdf_train)

class_tree_final_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.

```r
library(ranger)
cores <- parallel::detectCores()

rf_mod <-
  rand_forest(mtry = tune(), min_n = tune(), trees = 1000) %>%
  set_engine("ranger", importance = "impurity", num.threads = cores) %>%
  set_mode("classification")
```

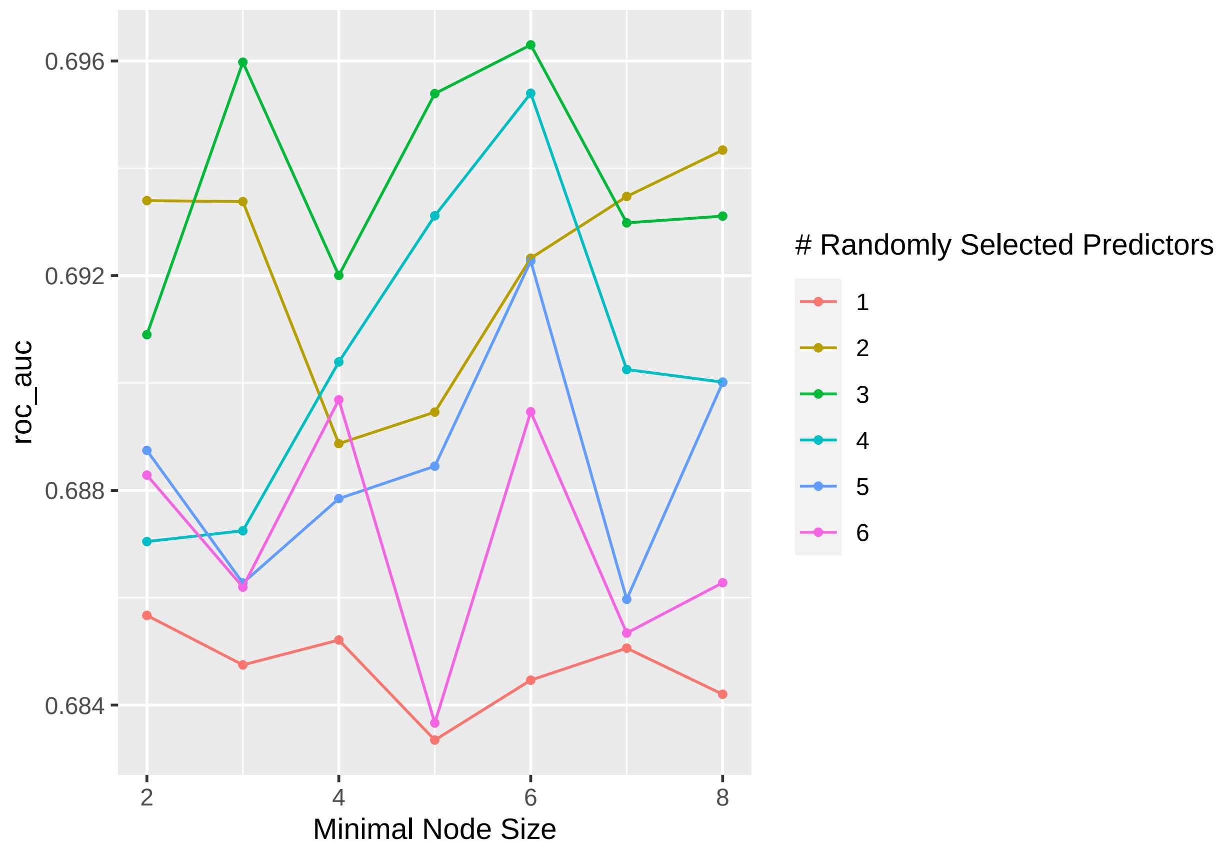
```r
random_tree_wf <- workflow() %>%
  add_model(rf_mod) %>%
  add_recipe(filterdf_recipe)
```

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?**

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?

```r
Auto_folds <-  vfold_cv(filterdf_train, strata = type_1, v = 5)

rf_grid <- grid_regular(
  mtry(range = c(1, 6)),
  min_n(range = c(2, 8)),
  levels = 8
)

set.seed(3456)
tune_res_rf <- tune_grid(
  random_tree_wf,
  resamples = Auto_folds,
  grid=rf_grid,
  metrics = metric_set(roc_auc)
)

autoplot(tune_res_rf)
```



Exercise 7

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

```
best_rf <- select_best(tune_res_rf)
best_rf
```

```
## # A tibble: 1 x 3
##    mtry min_n .config
##   <int> <int> <chr>
## 1     3     6 Preprocessor1_Model27
```

```
collect_metrics(tune_res_rf, summarize = TRUE) %>% arrange(desc(mean), .by_group = TRUE)
```

```
## # A tibble: 42 x 8
##     mtry min_n .metric .estimator  mean     n std_err .config
##    <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     3     6 roc_auc hand_till  0.696     5  0.0120 Preprocessor1_Model27
##  2     3     3 roc_auc hand_till  0.696     5  0.0120 Preprocessor1_Model09
##  3     4     6 roc_auc hand_till  0.695     5  0.0107 Preprocessor1_Model28
##  4     3     5 roc_auc hand_till  0.695     5  0.0116 Preprocessor1_Model21
##  5     2     8 roc_auc hand_till  0.694     5  0.0140 Preprocessor1_Model38
##  6     2     7 roc_auc hand_till  0.693     5  0.0139 Preprocessor1_Model32
```

```
## 7     2     2 roc_auc hand_till  0.693     5  0.0112 Preprocessor1_Model02
## 8     2     3 roc_auc hand_till  0.693     5  0.0119 Preprocessor1_Model08
## 9     4     5 roc_auc hand_till  0.693     5  0.0128 Preprocessor1_Model22
## 10    3     8 roc_auc hand_till  0.693     5  0.0135 Preprocessor1_Model39
## # ... with 32 more rows
```

Exercise 8

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

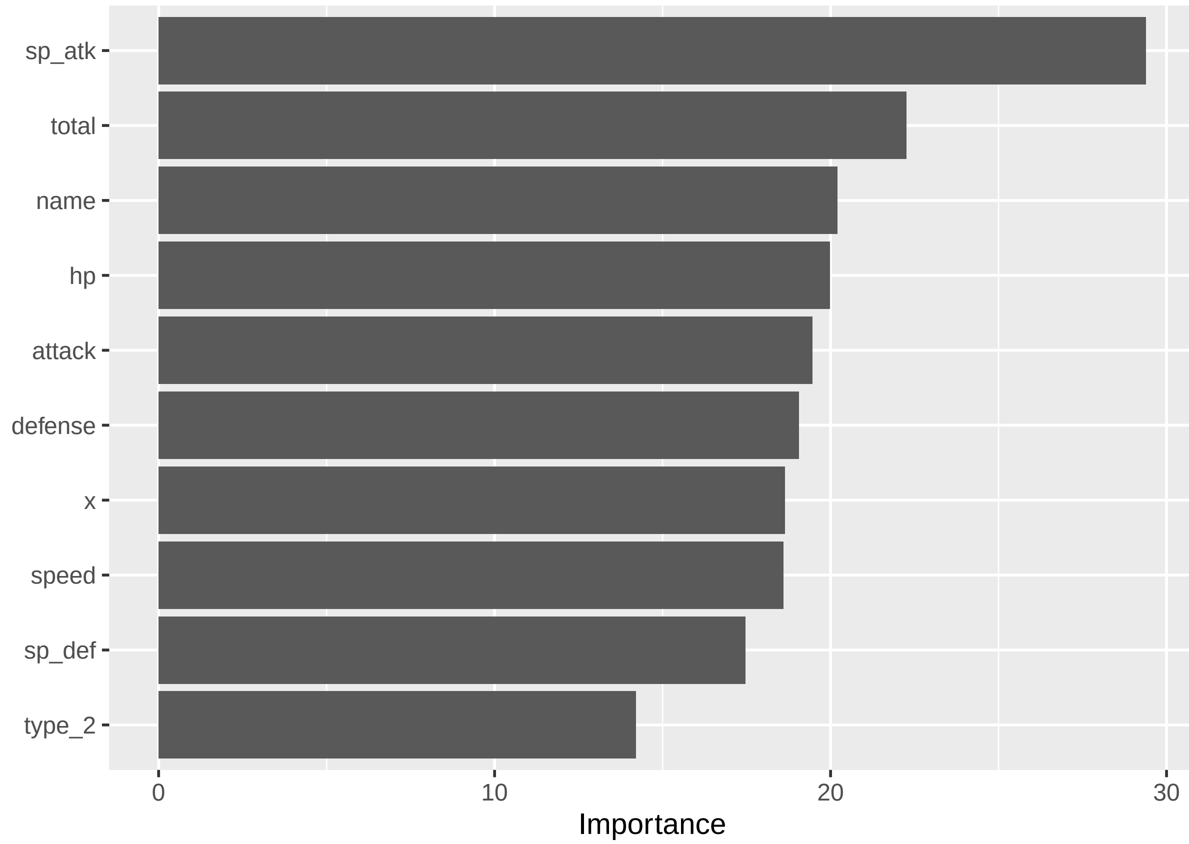
Which variables were most useful? Which were least useful? Are these results what you expected, or not?

```
rf_tree_final <- finalize_workflow(random_tree_wf, best_tree)

rf_tree_final_fit <- fit(rf_tree_final, data = filterdf_train)
```

```
library(vip)

best_auc <- select_best(tune_res_rf, "roc_auc")

final_rf <- finalize_model(
  rf_mod,
  best_auc
)
final_rf
```

```
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = 3
##   trees = 1000
##   min_n = 6
##
## Engine-Specific Arguments:
##   importance = impurity
##   num.threads = cores
##
## Computational engine: ranger
```

```
rf_fit <- fit(final_rf, type_1 ~ ., data = filterdf_train)

vip(rf_fit)
```



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.

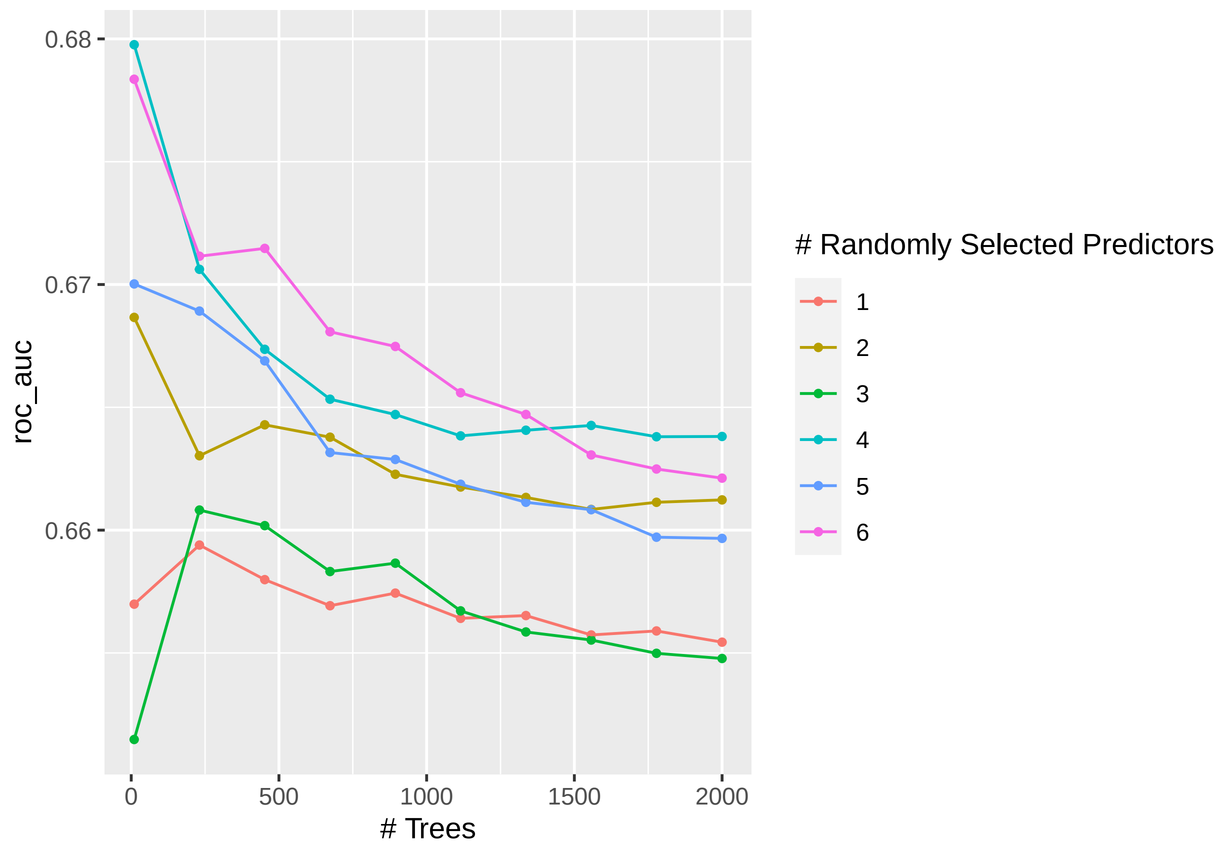
What do you observe?

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

```
library(xgboost)

xgb_mod <-
  boost_tree(
  trees = tune(),
  #min_n = tune(),
  mtry = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

```
xgb_tree_wf <- workflow() %>%
  add_model(xgb_mod) %>%
  add_recipe(filterdf_recipe)
```

```
Auto_folds <-  vfold_cv(filterdf_train, strata = type_1, v = 5)

xgb_grid <- grid_regular(
  mtry(range = c(1, 6)),
  #min_n(range = c(2,4)),
  trees(range = c(10,2000)),
  levels = 10
)

set.seed(3456)
tune_res_xgb <- tune_grid(
  xgb_tree_wf,
  resamples = Auto_folds,
  grid=xgb_grid,
  metrics = metric_set(roc_auc)
)

autoplot(tune_res_xgb)
```




```
best_xgb <- select_best(tune_res_xgb)
best_xgb
```

```
## # A tibble: 1 x 3
##    mtry trees .config
##   <int> <int> <chr>
```

```
## 1     4    10 Preprocessor1_Model31
```

```r
collect_metrics(tune_res_xgb, summarize = TRUE) %>% arrange(desc(mean), .by_group = TRUE)
```

```
## # A tibble: 60 x 8
##     mtry trees .metric .estimator  mean     n std_err .config
##    <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     4    10 roc_auc hand_till  0.680     5 0.0122  Preprocessor1_Model31
##  2     6    10 roc_auc hand_till  0.678     5 0.00879 Preprocessor1_Model51
##  3     6   452 roc_auc hand_till  0.671     5 0.0104  Preprocessor1_Model53
##  4     6   231 roc_auc hand_till  0.671     5 0.0111  Preprocessor1_Model52
##  5     4   231 roc_auc hand_till  0.671     5 0.0141  Preprocessor1_Model32
##  6     5    10 roc_auc hand_till  0.670     5 0.0147  Preprocessor1_Model41
##  7     5   231 roc_auc hand_till  0.669     5 0.0110  Preprocessor1_Model42
##  8     2    10 roc_auc hand_till  0.669     5 0.00839 Preprocessor1_Model11
##  9     6   673 roc_auc hand_till  0.668     5 0.00954 Preprocessor1_Model54
## 10     6   894 roc_auc hand_till  0.667     5 0.00929 Preprocessor1_Model55
## # ... with 50 more rows
```

```r
xgb_tree_final <- finalize_workflow(xgb_tree_wf, best_xgb)

xgb_tree_final_fit <- fit(xgb_tree_final, data = filterdf_train)
```

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?

```r
tree_test_fit <- fit(class_tree_final, data = filterdf_train)
rf_tree_final_fit <- fit(rf_tree_final, data = filterdf_train)
xgb_tree_final_fit <- fit(xgb_tree_final, data = filterdf_train)


p1 <- augment(tree_test_fit, new_data = filterdf_train) %>%
  select(type_1, starts_with(".pred")) %>%
  roc_auc(type_1, .pred_Bug:.pred_Water)

p2<-augment(rf_tree_final_fit, new_data = filterdf_train) %>%
    select(type_1, starts_with(".pred"))%>%
    roc_auc(type_1, .pred_Bug:.pred_Water)

p3<-augment(xgb_tree_final_fit, new_data = filterdf_train) %>%
  select(type_1, starts_with(".pred")) %>%
  roc_auc(type_1, .pred_Bug:.pred_Water)

accuracies <- c(p1$estimate,p2$estimate,p3$estimate)
models <- c("Decision Tree", "Random Forrest", "BoostedTree")
```

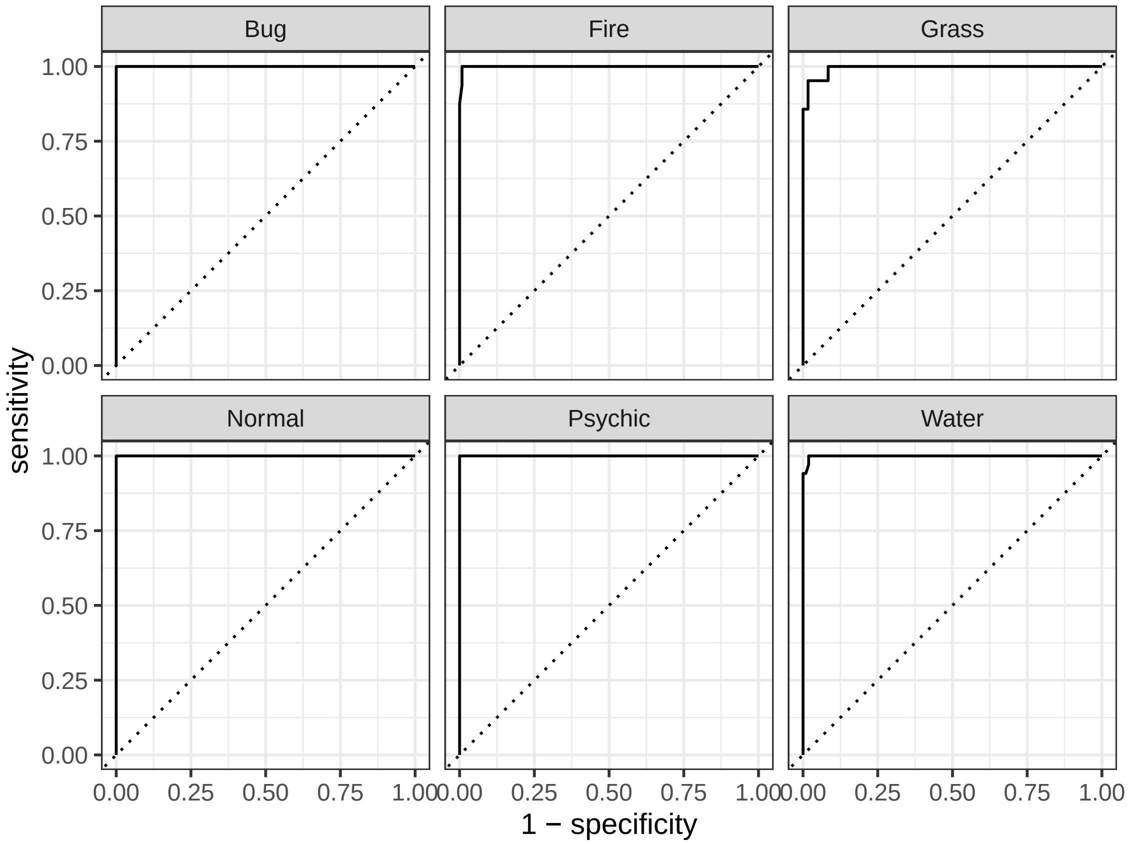
```r
results <- tibble(roc = accuracies, models = models)
results %>%
  arrange(-accuracies)
```

```
## # A tibble: 3 x 2
##     roc models
##   <dbl> <chr>
## 1 0.998 BoostedTree
## 2 0.871 Decision Tree
## 3 0.5   Random Forrest
```

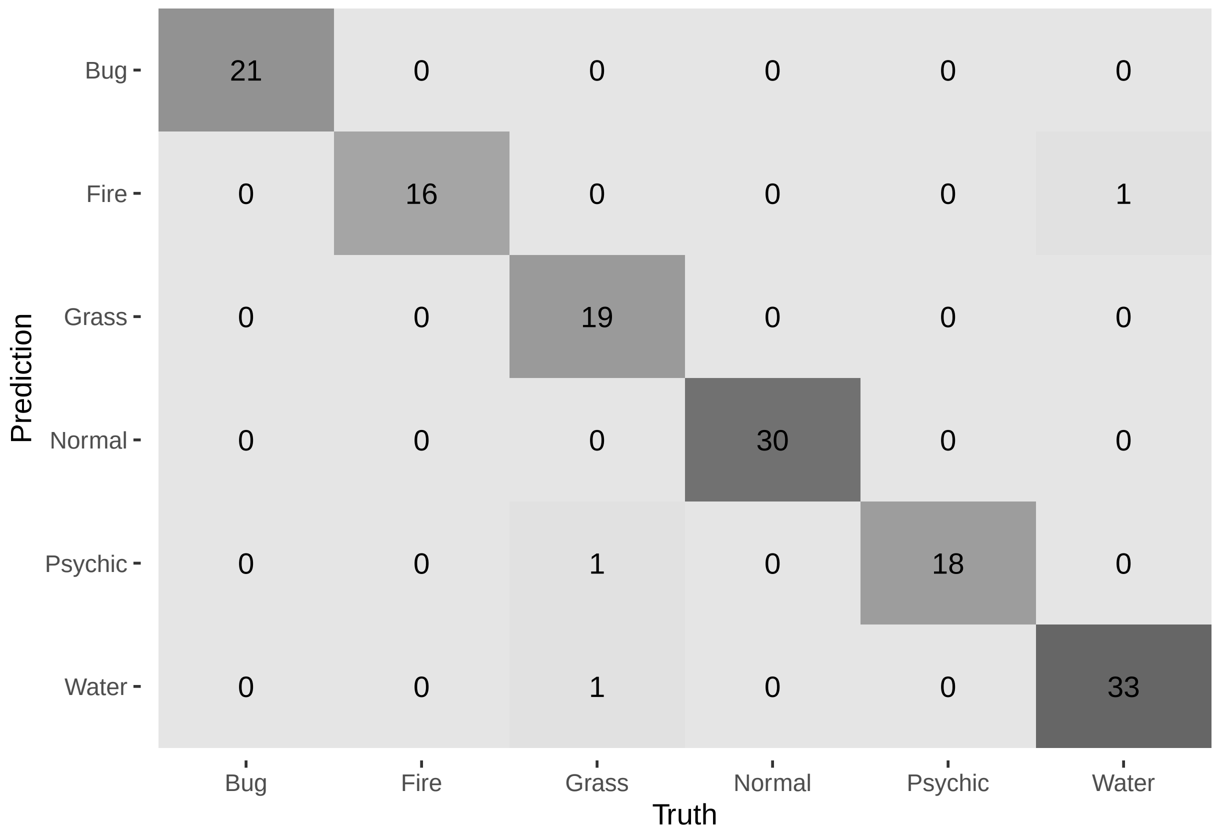
```r
library(vip)

best_fit <- select_best(tune_res_xgb,tune_res_rf,tune_res, metric = "roc_auc" )

final_fit <- finalize_workflow(
 xgb_tree_final,
 best_fit
)

final_fit <- fit(final_fit, data = filterdf_test)

predicted_data <- augment(final_fit, new_data = filterdf_test) %>%
  select(type_1, starts_with(".pred"))

predicted_data %>% roc_auc(type_1, .pred_Bug:.pred_Water)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.999
```

```r
predicted_data %>% roc_curve(type_1, .pred_Bug:.pred_Water) %>%
  autoplot()
```




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


Prediction
Truth
Bug
Fire
Grass
Normal
Psychic
Water
21
16
19
30
18
33
0
1