PSTAT131_HW6

Tao Wang

2022-05-20

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.

Loading Packages

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 0.2.0 --

## v broom        0.7.12     v rsample      0.1.1
## v dials        0.1.0      v tune         0.2.0
## v infer        1.0.0      v workflows    0.2.6
## v modeldata    0.1.1      v workflowsets 0.2.1
## v parsnip      0.2.1      v yardstick    0.0.9
## v recipes      0.2.0

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Search for functions across packages at https://www.tidymodels.org/find/
```

```
library(ISLR)
library(rpart.plot)
```

```
## Loading required package: rpart
```

```
##
## Attaching package: 'rpart'
## The following object is masked from 'package:dials':
##
##     prune
```

```
library(vip)
```

```
##
## Attaching package: 'vip'
## The following object is masked from 'package:utils':
##
##     vi
```

```
library(janitor)
```

```
##
## Attaching package: 'janitor'
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
library(randomForest)
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:dplyr':
##
##     combine
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(ISLR2)
```

```
##
## Attaching package: 'ISLR2'
## The following objects are masked from 'package:ISLR':
##
##     Auto, Credit
```

```
library(glmnet)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-4
```

```
library(ranger)
```

```
##
## Attaching package: 'ranger'

## The following object is masked from 'package:randomForest':
##
##     importance
```

```
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.

```
# read the data
pokemon <- read.csv(file = "data/Pokemon.csv")

#  Use its `clean_names()` function on the Pokémon data
pokemon<-clean_names(pokemon)
head(pokemon)
```

```
##   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
# Filter out the rarer Pokémon types
pokemon <- pokemon %>%
  filter(type_1 %in% c('Bug', 'Fire', 'Grass' , 'Normal', 'Water', 'Psychic'))

# Convert `type_1` and `legendary` to factors
pokemon$type_1 <- factor(pokemon$type_1)
pokemon$legendary <- factor(pokemon$legendary)

# Do an initial split of the data; you can choose the percentage for splitting.
# Stratify on the outcome variable.
set.seed(1234)
pokemon_split <- initial_split(pokemon,prop = 0.80, strata = type_1)
pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)

# Fold the training set using *v*-fold cross-validation, with `v = 5`.
# Stratify on the outcome variable.
set.seed(1234)
pokemon_folds <- vfold_cv(pokemon_train, v = 5, strata = type_1)

# 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.
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk +
                           attack + speed + defense+ hp +sp_def, pokemon_train) %>%
  step_dummy(legendary,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?

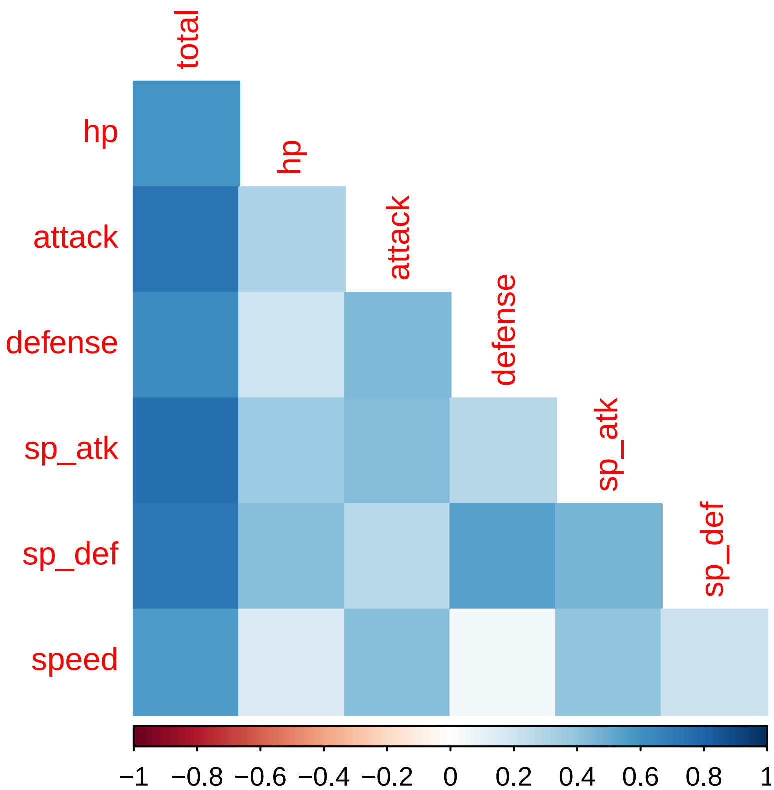
```r
head(pokemon_train)
```

```
##      x                   name type_1 type_2 total hp attack defense sp_atk sp_def
## 15 11                Metapod    Bug          205 50      20      55     25     25
## 16 12             Butterfree    Bug Flying   395 60      45      50     90     80
## 17 13                 Weedle    Bug Poison   195 40      35      30     20     20
## 18 14                 Kakuna    Bug Poison   205 45      25      50     25     25
## 19 15               Beedrill    Bug Poison   395 65      90      40     45     80
## 20 15 BeedrillMega Beedrill    Bug Poison   495 65     150      40     15     80
##    speed generation legendary
## 15    30          1     False
## 16    70          1     False
## 17    50          1     False
## 18    35          1     False
## 19    75          1     False
```

```
## 20   145          1      False
```

```r
# notice we should remove variable x and generation
# since x is the index and generation is categorical
pokemon_train%>%
  select(is.numeric,-x,-generation) %>%
  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.
```



- What relationships, if any, do you notice?
- Answer:
- 1. Variable total has positive relationships with variable hp,attack,defense,sp_atk,sp_def and speed.
- 2. Variable hp has positive relationships with variable attack,defense,sp_atk and sp_def. Variable hp also has a little positive relationship with variable speed.
- 3. Variable attack has positive relationships with variable defense,sp_atk,sp_def and speed.

- 4. Variable defense has positive relationships with variable sp__atk and sp__def. Variable defense also has a little positive relationship with variable speed.
- 5. Variable sp__atk has positive relationships with variable sp__def and speed.
- 6. Variable sp__def has a little positive relationship with variable speed.
- Do these relationships make sense to you?
- Answer:
- Yes, these relationships make sense to me. For example, variable total has positive relationships with variable hp,attack,defense,sp__atk,sp__def and speed. It is because variable total is a general guide to how strong a pokemon is, and how strong a pokemon is depends on variable hp,attack,defense,sp__atk,sp__def and speed. So, it is why the variable total has positive relationships with variable hp,attack,defense,sp__atk,sp__def and speed.

Exercise 3

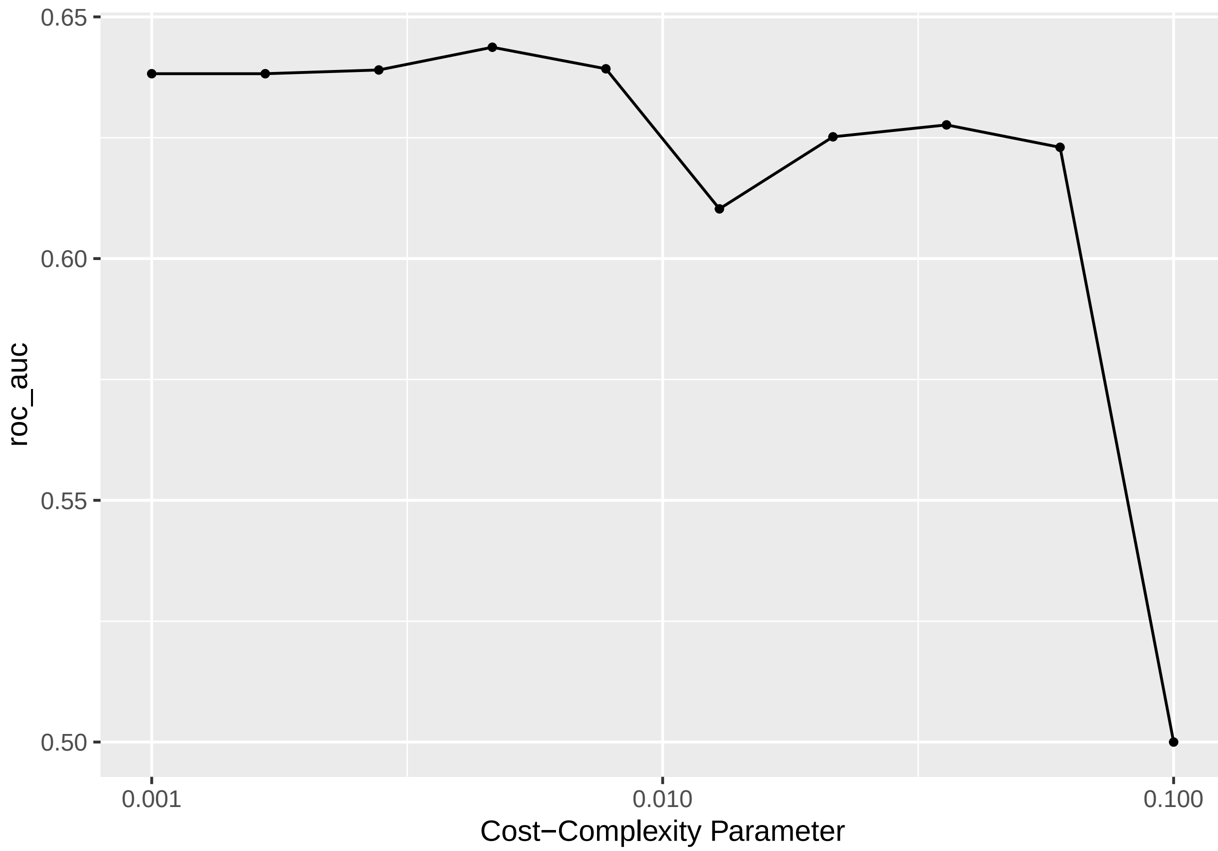
First, set up a decision tree model and workflow. Tune the `cost_complexity` hyperparameter. Use the same levels we used in Lab 7 – that is, `range = c(-3, -1)`. Specify that the metric we want to optimize is `roc_auc`.

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

```r
# set up a decision tree model
tree_spec <- decision_tree() %>%
  set_engine("rpart")

class_tree_spec <- tree_spec %>%
  set_mode("classification")%>%
  set_args(cost_complexity = tune()) # Tune the `cost_complexity` hyperparameter


# set up a decision tree workflow.
class_tree_wf <- workflow() %>%
  add_model(class_tree_spec)%>%
  add_recipe(pokemon_recipe)

# Use the same levels we used in Lab 7 -- that is, `range = c(-3, -1)`.
param_grid <- grid_regular(cost_complexity(range = c(-3, -1)), levels = 10)

# Specify that the metric we want to optimize is `roc_auc`
tune_res <- tune_grid(
  class_tree_wf,
  resamples = pokemon_folds,
  grid = param_grid,
  metrics = metric_set(roc_auc)
)
```

```
## ! Fold1: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold2: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold3: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold4: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold5: preprocessor 1/1: The following variables are not factor vectors and wil...
```

```r
# Print an `autoplot()` of the results.
autoplot(tune_res)
```



*

What do you observe? * Answer: * We can see that as the value of the cost-complexity parameter increases, the ROC__AUC value will finally decrease.

- To be more specific,
- when the cost-complexity parameter less than 0.00464, as the the cost-complexity parameter increases, the ROC__AUC value increases (a little bit);
- when the cost-complexity parameter is around 0.00464 to 0.0129, as the the cost-complexity parameter increases, the ROC__AUC value decreases (a little bit);
- when the cost-complexity parameter is around 0.0129 to 0.0599, as the the cost-complexity parameter increases, the ROC__AUC value increases (a little bit)
- when the cost-complexity parameter is greater than 0.0599, as the the cost-complexity parameter increases, the ROC__AUC value decreases (very fast).
- Does a single decision tree perform better with a smaller or larger complexity penalty?
- Answer:
- According to the graph, we know that a single decision tree perform better with a smaller complexity penalty.

Exercise 4

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

```r
# What is the `roc_auc` of your best-performing boosted tree model on the folds?
arrange(collect_metrics(tune_res), desc(mean))
```

```
## # A tibble: 10 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  0.644     5 0.0109  Preprocessor1_Model04
##  2         0.00774 roc_auc hand_till  0.639     5 0.0124  Preprocessor1_Model05
##  3         0.00278 roc_auc hand_till  0.639     5 0.00891 Preprocessor1_Model03
##  4         0.001   roc_auc hand_till  0.638     5 0.00916 Preprocessor1_Model01
##  5         0.00167 roc_auc hand_till  0.638     5 0.00916 Preprocessor1_Model02
##  6         0.0359  roc_auc hand_till  0.628     5 0.0139  Preprocessor1_Model08
##  7         0.0215  roc_auc hand_till  0.625     5 0.0141  Preprocessor1_Model07
##  8         0.0599  roc_auc hand_till  0.623     5 0.0100  Preprocessor1_Model09
##  9         0.0129  roc_auc hand_till  0.610     5 0.0124  Preprocessor1_Model06
## 10         0.1     roc_auc hand_till  0.5       5 0       Preprocessor1_Model10
```

- What is the `roc_auc` of your best-performing pruned decision tree on the folds?
- Answer: The `roc_auc` of my best-performing pruned decision tree on the folds is 0.644.

Exercise 5

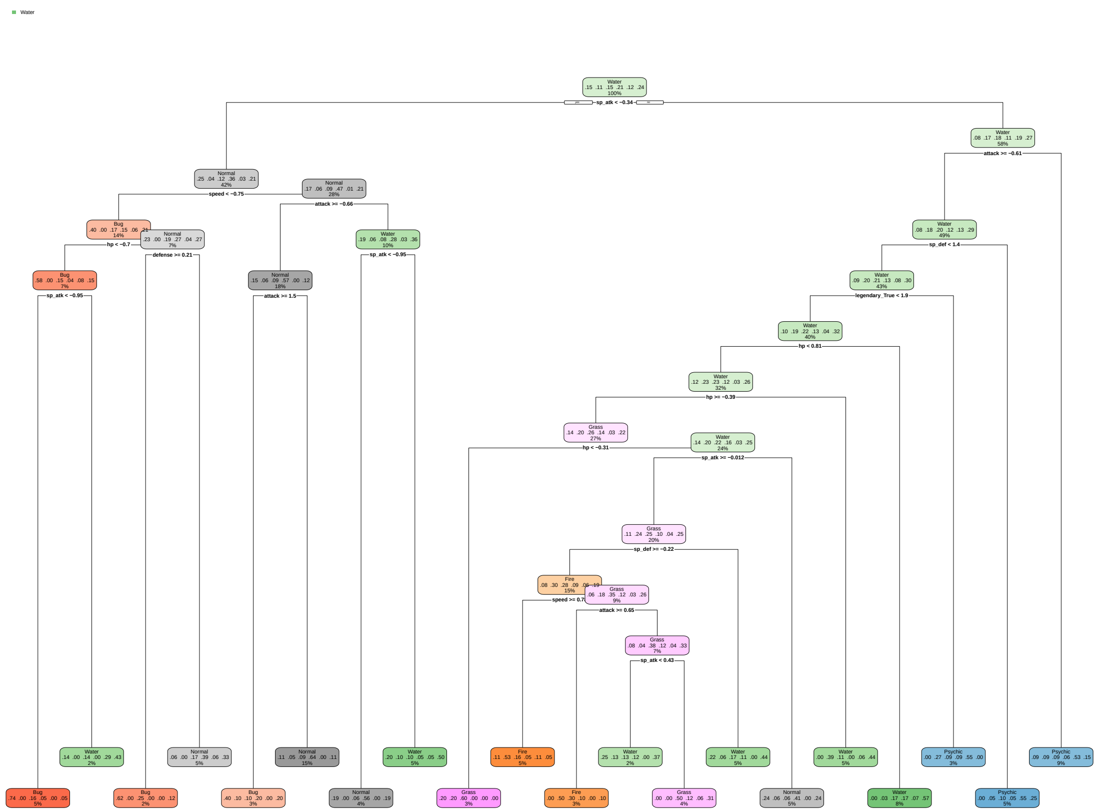
Using `rpart.plot`, fit and visualize your best-performing pruned decision tree with the *training* set.

```r
# according to the professor, we can just use select_best() method
best_complexity<- select_best(tune_res)



class_tree_final <- finalize_workflow(class_tree_wf, best_complexity)

class_tree_final_fit <- fit(class_tree_final, data = pokemon_train)
```

```
## Warning: The following variables are not factor vectors and will be ignored:
## `generation`
```

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

```
## Warning: Cannot retrieve the data used to build the model (so cannot determine roundint and is.binary
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.

## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



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?**

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

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

pgram_grid<- grid_regular(mtry(range= c(1,8)),
                          trees(range = c(100,1000)),
                           min_n(range = c(1,10)),
                          levels = 8)
```

- Explain in your own words what each of these hyperparameters represent.
- Answer:

- mtry: The number of predictors that will be randomly sampled at each split when we are creating the tree models.
- trees: The number of trees contained in the ensemble.
- min_n: The minimum number of data points in a node that are required for the node to be split further.
- Note that `mtry` should not be smaller than 1 or larger than 8. Explain why not.
- Answer: Notice that `mtry` is the number of predictors that will be randomly sampled at each split when we are creating the tree models. Since we only have 8 predictors, then we cannot make `mtry` greater than 8 or smaller than 1. According to the professor, if we do that, it won't be meaningful.
- What type of model would `mtry = 8` represent??
- Answer: `mtry = 8` represents the bagging model.

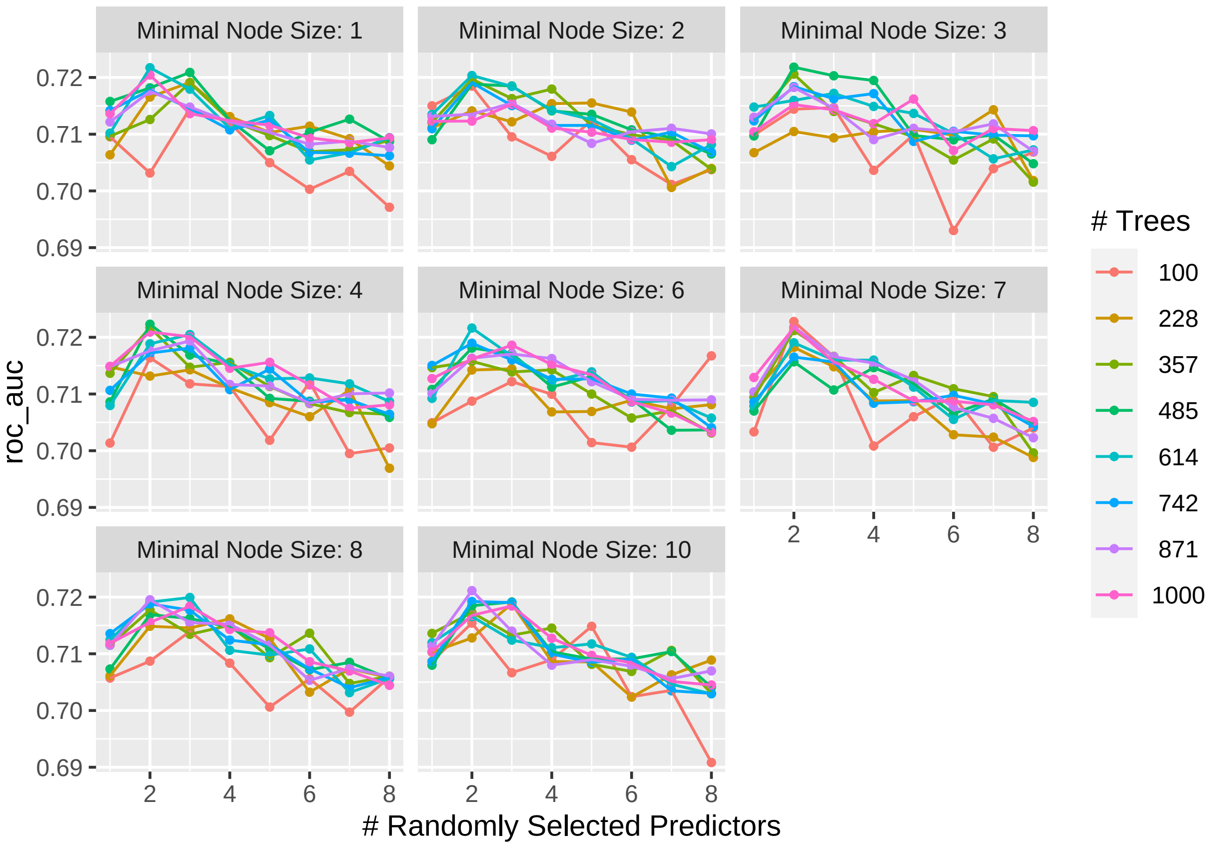
Exercise 6

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?

```
# Specify that the metric we want to optimize is `roc_auc`
forest_tune_res <- tune_grid(
  class_forest_rf,
  resamples = pokemon_folds,
  grid = pgram_grid,
  metrics = metric_set(roc_auc)
)
```

```
## ! Fold1: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold2: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold3: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold4: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold5: preprocessor 1/1: The following variables are not factor vectors and wil...
```

```
# Print an `autoplot()` of the results.
autoplot(forest_tune_res)
```



- What do you observe?
- Answer: According to the graph, if we fix the value of min_n (which is the minimal node size), we can saw that as the number of randomly selected predictors (variable mtry) increases, the trend of roc_auc values of most of these models will finally decrease.
- What values of the hyperparameters seem to yield the best performance?
- Answer: According to the graph, we saw that when the number of randomly selected predictors (variable mtry) = 2, the number of tree (variable trees) = 100, and the minimal node size (variable min_n) = 7, we seem to yield the best performance.

Exercise 7

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

```
# What is the `roc_auc` of your best-performing boosted tree model on the folds?
arrange(collect_metrics(forest_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     2   100     7 roc_auc hand_till  0.723     5 0.00925 Preprocessor1_Model~
##  2     2   485     4 roc_auc hand_till  0.722     5 0.00647 Preprocessor1_Model~
##  3     2   485     3 roc_auc hand_till  0.722     5 0.0112  Preprocessor1_Model~
##  4     2   871     7 roc_auc hand_till  0.722     5 0.00943 Preprocessor1_Model~
##  5     2   357     4 roc_auc hand_till  0.722     5 0.00968 Preprocessor1_Model~
##  6     2   614     1 roc_auc hand_till  0.722     5 0.0102  Preprocessor1_Model~
##  7     2  1000     7 roc_auc hand_till  0.722     5 0.0102  Preprocessor1_Model~
```

```
## 8      2   614      6 roc_auc hand_till  0.722     5 0.0109  Preprocessor1_Model~
## 9      2   357      7 roc_auc hand_till  0.721     5 0.00954 Preprocessor1_Model~
## 10     2   871     10 roc_auc hand_till  0.721     5 0.00820 Preprocessor1_Model~
## # ... with 502 more rows
```

```
best_complexity<- select_best(forest_tune_res, metric= 'roc_auc')

class_forest_final <- finalize_workflow(class_forest_rf, best_complexity)
```

- What is the `roc_auc` of your best-performing random forest model on the folds?
- Answer: The `roc_auc` of your best-performing random forest model on the folds is 0.723.

Exercise 8

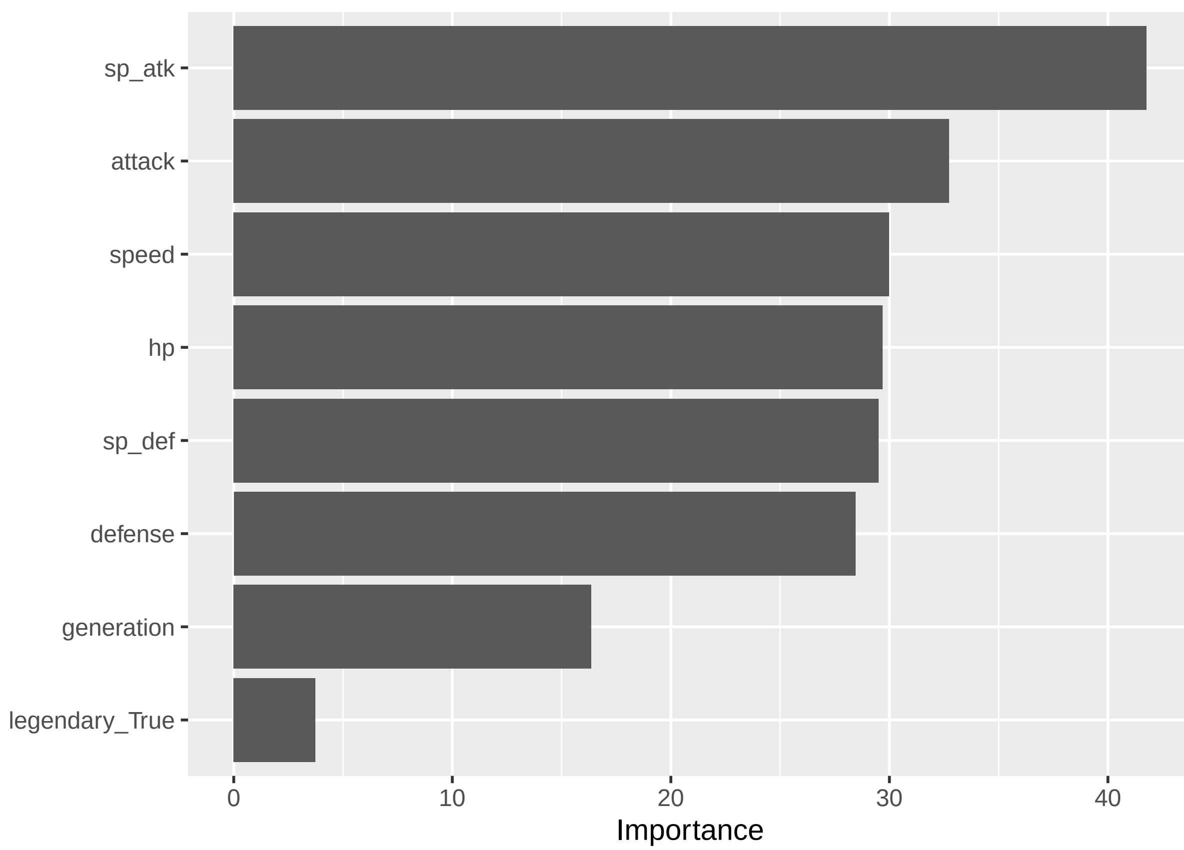
Create a variable importance plot, using `vip()`, with your best-performing random forest model fit on the *training* set.

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

```
class_forest_final_fit <- fit(class_forest_final, data = pokemon_train)
```

```
## Warning: The following variables are not factor vectors and will be ignored:
## `generation`
```

```
class_forest_final_fit%>%
  extract_fit_engine() %>%
  vip()
```



- Which variables were most useful?

- Answer: sp_atk is the most useful variable.

- Which were least useful?
- Answer: legendary_True (or legendary) is the least useful variable.
- Are these results what you expected, or not?
- Answer: Yes, these results are what I expected. Anyone who has watched Pokemon videos or played Pokemon games knows that sp_atk is very important to a Pokemon's strength, and the strength of a Pokemon has nothing to do with whether he/she is a legendary Pokemon.

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.

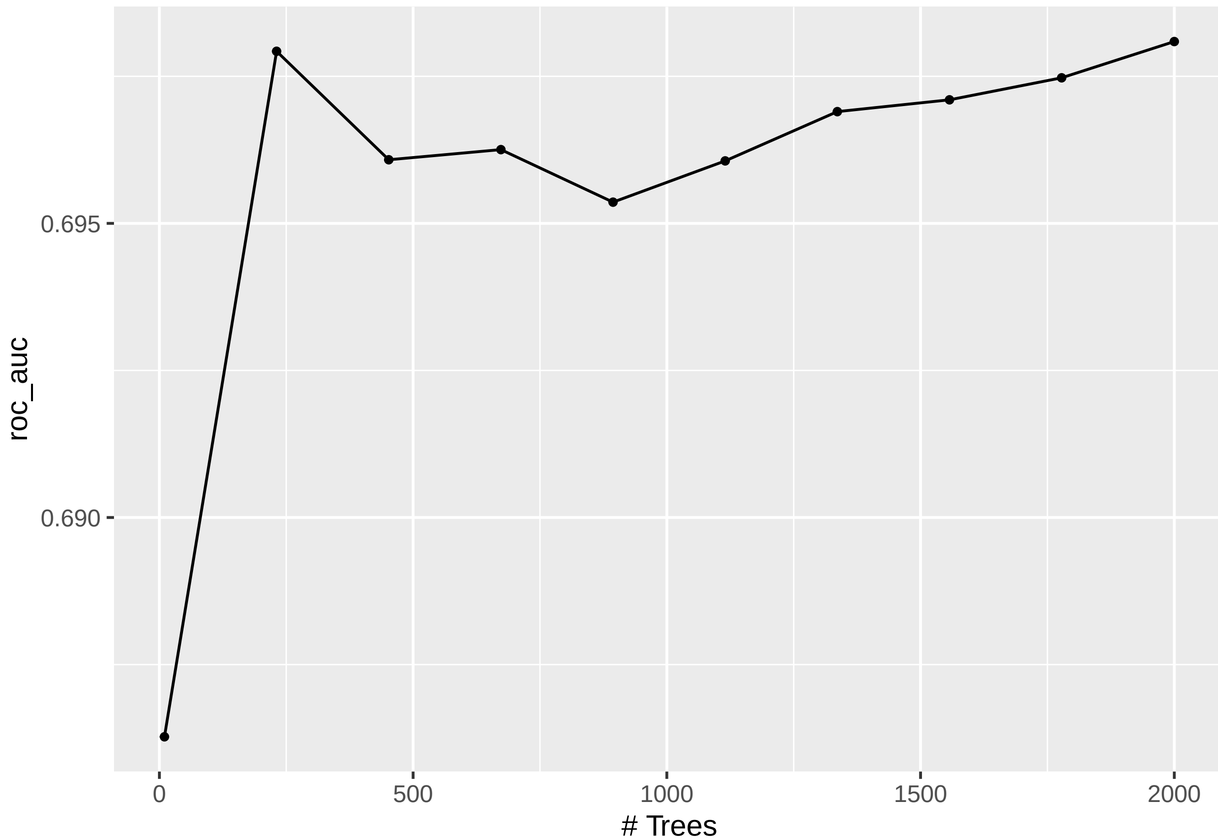
What do you observe?

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

```
boost_spec <- boost_tree(trees = tune(), tree_depth = 4) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

boost_rf <- workflow()%>%
  add_model(boost_spec)%>%
  add_recipe(pokemon_recipe)

pgram3_grid<- grid_regular(trees(range = c(10,2000) ),
                           levels = 10)


# Specify that the metric we want to optimize is `roc_auc`
boost_tune_res <- tune_grid(
  boost_rf,
  resamples = pokemon_folds,
  grid = pgram3_grid,
  metrics = metric_set(roc_auc)
)
```

```
## ! Fold1: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold2: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold3: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold4: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold5: preprocessor 1/1: The following variables are not factor vectors and wil...
```

```
# Print an `autoplot()` of the results.
autoplot(boost_tune_res)
```




```r
# What is the `roc_auc` of your best-performing boosted tree model on the folds?
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  0.698     5 0.00846 Preprocessor1_Model10
##  2   231 roc_auc hand_till  0.698     5 0.00700 Preprocessor1_Model02
##  3  1778 roc_auc hand_till  0.697     5 0.00864 Preprocessor1_Model09
##  4  1557 roc_auc hand_till  0.697     5 0.00838 Preprocessor1_Model08
##  5  1336 roc_auc hand_till  0.697     5 0.00813 Preprocessor1_Model07
##  6   673 roc_auc hand_till  0.696     5 0.00779 Preprocessor1_Model04
##  7   452 roc_auc hand_till  0.696     5 0.00708 Preprocessor1_Model03
##  8  1115 roc_auc hand_till  0.696     5 0.00789 Preprocessor1_Model06
##  9   894 roc_auc hand_till  0.695     5 0.00742 Preprocessor1_Model05
## 10    10 roc_auc hand_till  0.686     5 0.0171  Preprocessor1_Model01
```

- What do you observe?
- Answer: Based on the graph we get, we can see that:
- when the number of trees is less than 231, as the number of trees increases, the value of roc_auc will increase;
- when the number of trees is greater than 231 but less than 452, as the number of trees increases, the value of roc_auc will decrease;
- when the number of trees is greater than 452 but less than 894, as the number of trees increases, the value of roc_auc will increase a little first and then decrease;

- when the number of trees is greater than 894, as the number of trees increases, the value of roc_auc will increase.
- What is the `roc_auc` of your best-performing boosted tree model on the folds?
- Answer: The `roc_auc` of your best-performing boosted tree model on the folds is 0.698.

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?

```
# Display a table of the three ROC AUC values for your
# best-performing pruned tree, random forest, and boosted tree models.
value <- c(arrange(collect_metrics(tune_res), desc(mean))[1,4],
                 arrange(collect_metrics(forest_tune_res), desc(mean))[1,6],
                 arrange(collect_metrics(boost_tune_res), desc(mean))[1,4])
cnames <- c('ROC AUC values of pruned tree', 'ROC AUC values of random forest',
            'ROC AUC values of boosted tree')
rnames <- 'values'
table<- matrix(value, nrow = 1, ncol = 3, byrow = TRUE, dimnames = list(rnames,cnames))
table
```

```
##        ROC AUC values of pruned tree ROC AUC values of random forest
## values 0.6437168                     0.7227707
##        ROC AUC values of boosted tree
## values 0.698091
```

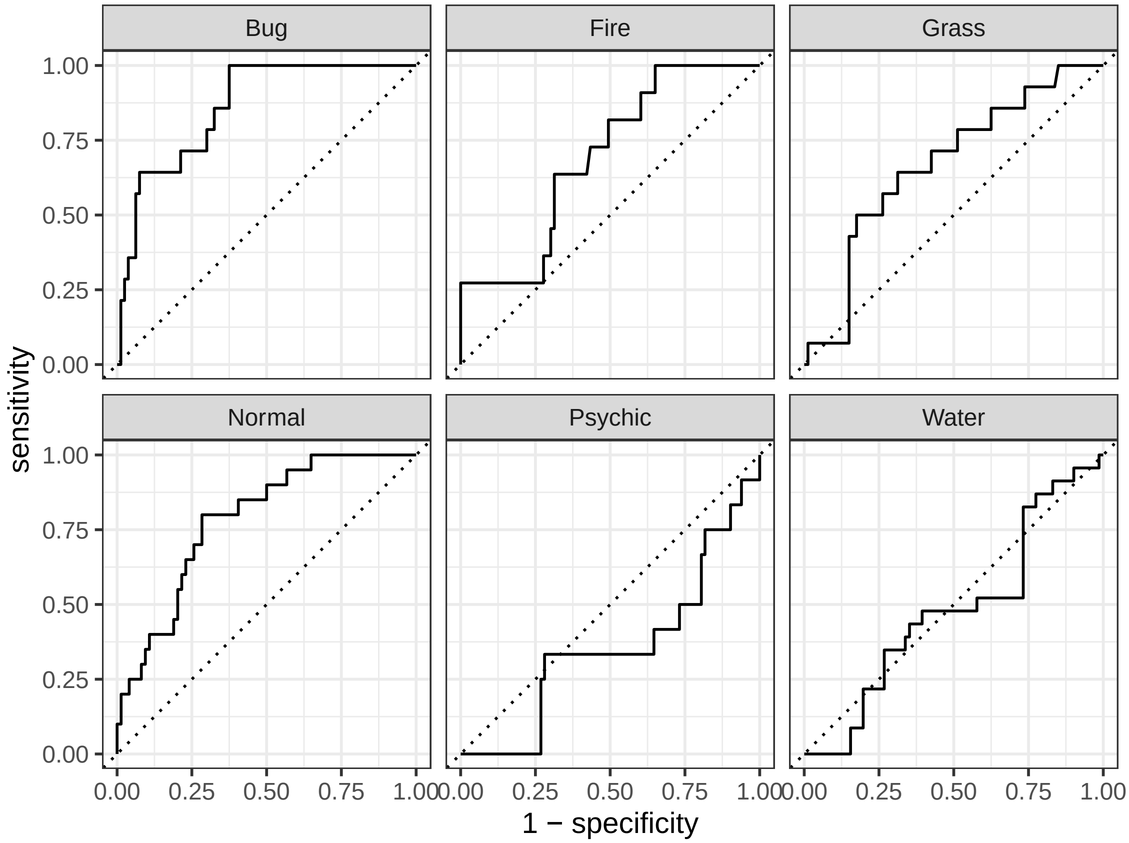
```
# Which performed best on the folds?
# random forest model performed best on the folds
# Select the best of the three and use `select_best()`, `finalize_workflow()`,
# and `fit()` to fit it to the *training* set.
best_complexity<- select_best(forest_tune_res, metric= 'roc_auc')

class_forest_final2 <- finalize_workflow(class_forest_rf, best_complexity)


class_forest_final_fit2 <- fit(class_forest_final, data = pokemon_train)
```

```
## Warning: The following variables are not factor vectors and will be ignored:
## `generation`
```

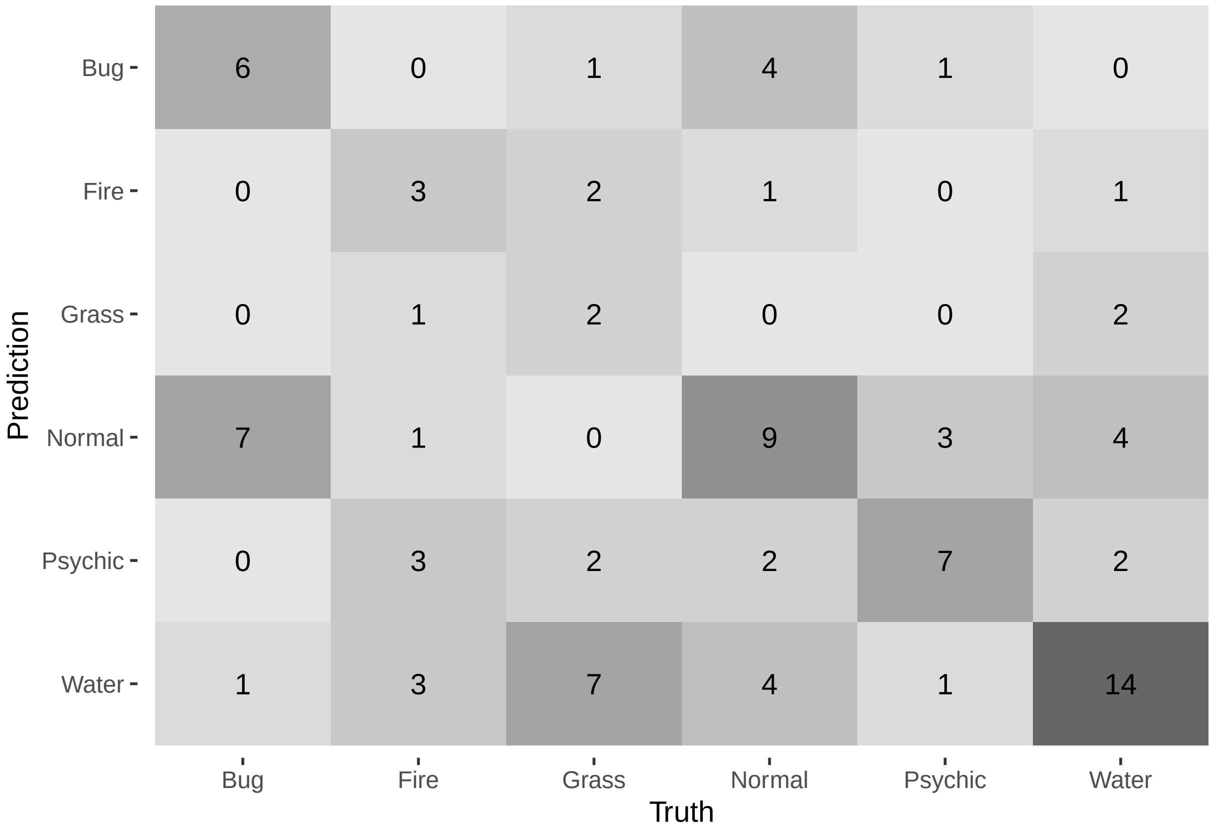
```
# Print the AUC value of your best-performing model on the testing set.

augment(class_forest_final_fit2, new_data = pokemon_test) %>%
   roc_auc(type_1,.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Water, .pred_Psychic)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.639
```

```
# Print the ROC curves.
augment(class_forest_final_fit2, new_data = pokemon_test) %>% roc_curve(type_1, .pred_Bug, .pred_Fire,
```




```
# Finally, create and visualize a confusion matrix heat map.
augment(class_forest_final_fit2, new_data = pokemon_test) %>%
  conf_mat(truth = type_1, estimate =.pred_class)%>%
  autoplot("heatmap")
```



*

Print the AUC value of your best-performing model on the testing set. * Answer:0.639

- Which classes was your model most accurate at predicting? Which was it worst at?
- Answer:
- In order to answer this question, we need to calculate # the number of a specific class which be predicted correctly / # the total number of predictions.
- Bug: $6/(6+0+1+4+1+0) = 0.5$
- Fire: $3/(0+3+2+1+0+1) = 0.4286$
- Grass: $2/(0+1+2+0+0+2) = 0.4$
- Normal: $9/(7+1+0+9+3+4) = 0.375$
- Psychic: $7/(0+3+2+2+7+2) = 0.4375$
- Water: $14/(1+3+7+4+1+14) = 0.46667$
- Based on the calculations, we know that my model most accurate at predicting at Class Bug, and my model worst accurate at predicting at Class Normal.