HW6 Xilong Li new version after deleted

Xilong Li (3467966)

2022-06-02

```
library(tidymodels)
library(tidyverse)
library(MASS)
library(glmnet)
library(janitor)
library(discrim)
library(poissonreg)
library(klaR)
library(rpart.plot)
library(vip)
library(randomForest)
library(xgboost)
```

Note of citation: all of the codes and the use of codes in this HW are cited from labs and previous homework :-)

Question 1:

```
pokemon_original <- read.csv("Pokemon.csv")
pokemon <- janitor:: clean_names(dat = pokemon_original)
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
filtered_pokemon <- pokemon %>%
  filter(type_1 %in% c("Bug", "Fire", "Grass", "Normal", "Water", "Psychic"))
final_pokemon <- filtered_pokemon %>%
  mutate(type_1 = factor(type_1),
         legendary = factor(legendary))
dim(final_pokemon)
```

```
## [1] 458  13
```

```r
class(final_pokemon$type_1)
```

```
## [1] "factor"
```

```r
class(final_pokemon$legendary)
```

```
## [1] "factor"
```

```r
set.seed(2216)

poke_split <- initial_split(final_pokemon, prop = 0.80,
                            strata = type_1)

poke_train <- training(poke_split)
poke_test <- testing(poke_split)

poke_folds <- vfold_cv(poke_train, v = 5, strata = type_1)
class(poke_folds)
```

```
## [1] "vfold_cv"   "rset"       "tbl_df"     "tbl"        "data.frame"
```

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

poke_recipe
```

```
## Recipe
##
## Inputs:
##
```

```
##       role #variables
##    outcome          1
##  predictor          8
##
## Operations:
##
## Dummy variables from legendary, generation
## Centering for all_predictors()
## Scaling for all_predictors()
```

Question 2:

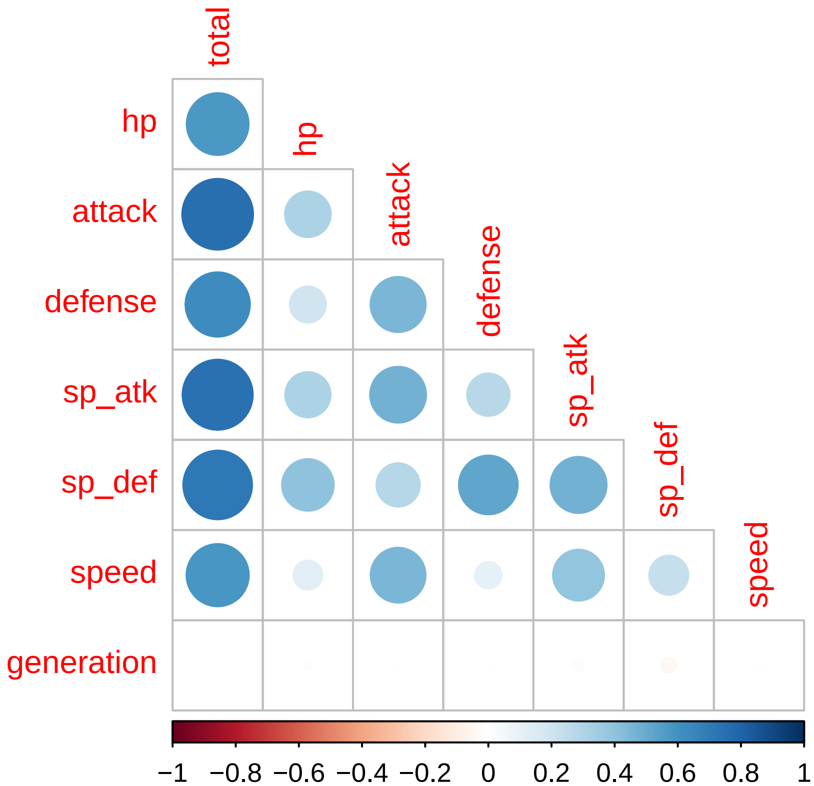
```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
head(poke_train)
```

```
##     x       name type_1 type_2 total hp attack defense sp_atk sp_def speed
## 14 10   Caterpie    Bug          195 45     30      35     20     20    45
## 15 11    Metapod    Bug          205 50     20      55     25     25    30
## 16 12 Butterfree    Bug Flying   395 60     45      50     90     80    70
## 18 14     Kakuna    Bug Poison   205 45     25      50     25     25    35
## 19 15   Beedrill    Bug Poison   395 65     90      40     45     80    75
## 36 46      Paras    Bug  Grass   285 35     70      55     45     55    25
##    generation legendary
## 14          1     False
## 15          1     False
## 16          1     False
## 18          1     False
## 19          1     False
## 36          1     False
```

```
cor_data <- poke_train %>%
  dplyr::select(-x) %>%
  dplyr::select(where(is.numeric))
corrplot(cor(cor_data), type = 'lower',diag = FALSE)
```



As it can be seen above, all attribute factors show positive correlation to each other, except for the attribute of "generation".
In particular, the attribute "total" shows strong positive correlation to other factors.
This makes sense to me since "total" measures the overall score of this pokemon, and thus the higher the scores of other factors, the higher the score of "total".

Question 3:

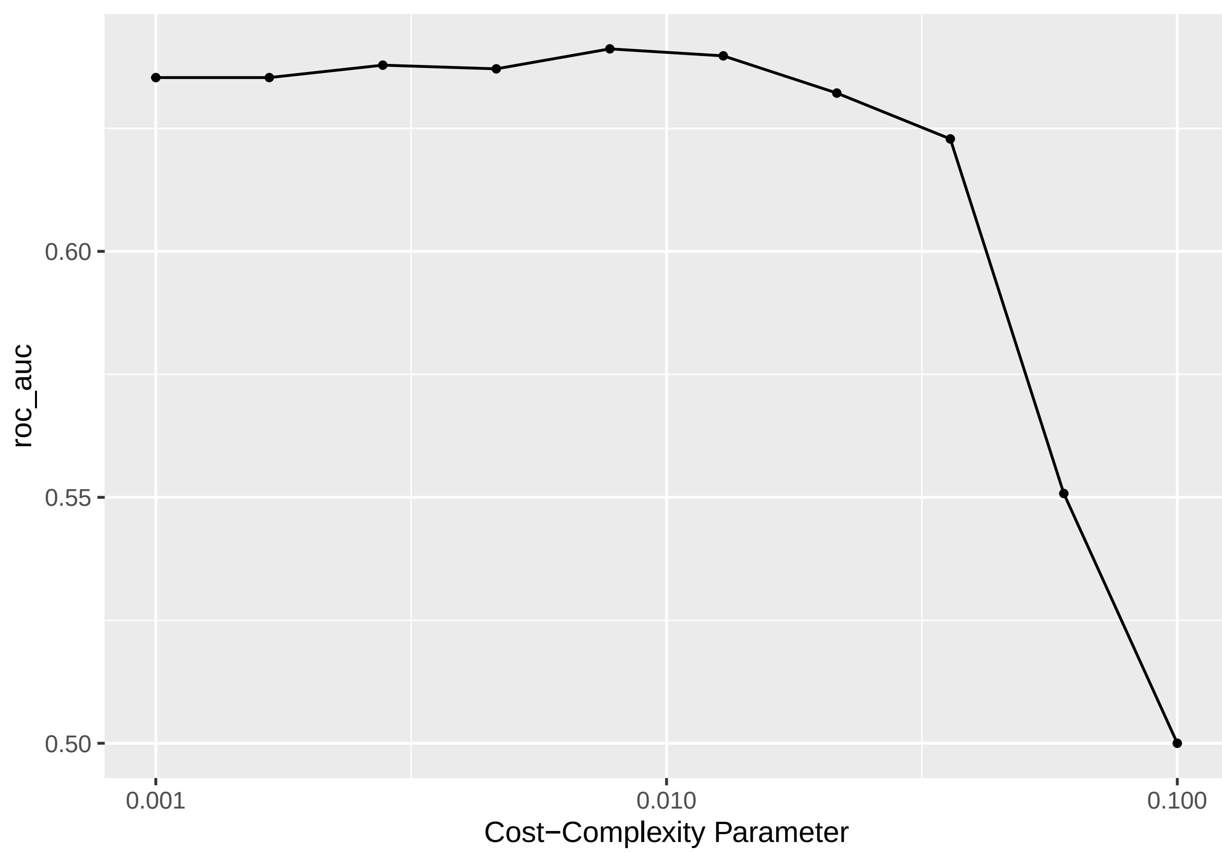
```
poke_spec <- decision_tree() %>%
  set_engine("rpart")

class_poke_spec <- poke_spec %>%
  set_mode("classification") %>%
  set_args(cost_complexity = tune())

class_poke_wf <- workflow() %>%
  add_model(class_poke_spec) %>%
  add_recipe(poke_recipe)

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

```
tree_tune <- tune_grid(
  class_poke_wf,
  resamples = poke_folds,
```

```r
  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
autoplot(tree_tune)
```



As it is shown in the graph above, it might be better to have smaller complexity penalty, because as the complexity penalty increases to a very large level, the roc_auc decreases quickly.

Question 4:

```r
best_tree_roc_auc <- collect_metrics(tree_tune) %>%
  arrange(-mean) %>%
  head(1)
best_tree_roc_auc
```

```
## # A tibble: 1 x 7
##   cost_complexity .metric .estimator  mean     n std_err .config
##             <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1         0.00774 roc_auc hand_till  0.641     5  0.0208 Preprocessor1_Model05
```

As it can be shown the fold with 0.007742637 cost_complexity level has the highest roc_auc mean, which is 0.6411691.

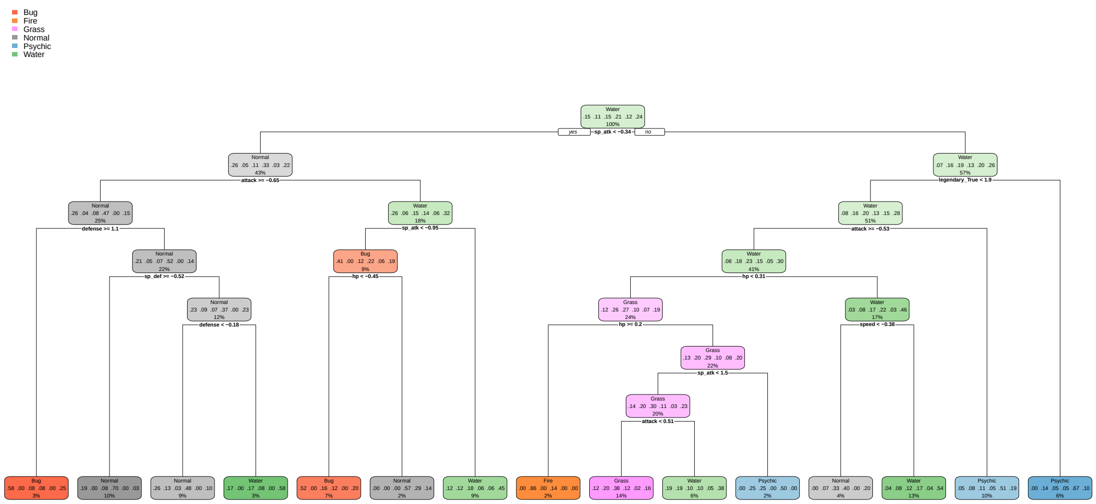
Question 5:

```r
best_complexity <- select_best(tree_tune)

class_poke_final <- finalize_workflow(class_poke_wf, best_complexity)

class_poke_final_fit <- fit(class_poke_final, data = poke_train)
```

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

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

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




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

rf_wf <- workflow() %>%
  add_recipe(poke_recipe) %>%
  add_model(rf_spec)

rf_grid <- grid_regular(mtry(range = c(3,8)), trees(range = c(5,500)),
                        min_n(range = c(3,8)), levels = 5)
```

a) mtry: means the number of randomly selected predictors;

b) trees: means the number of trees that will be in this model;

c) min_n: means the minimal node size, which is the minimum number of data points in a node;

Since "mtry" means the randomly selected predictors, and also because there are only 8 predictors in our model, the "mtry" can only be from 1 to 8;
If "mtry" = 8, which is the maximum number of predictors in our model, then the model will become a bagging forest.

Question 6:

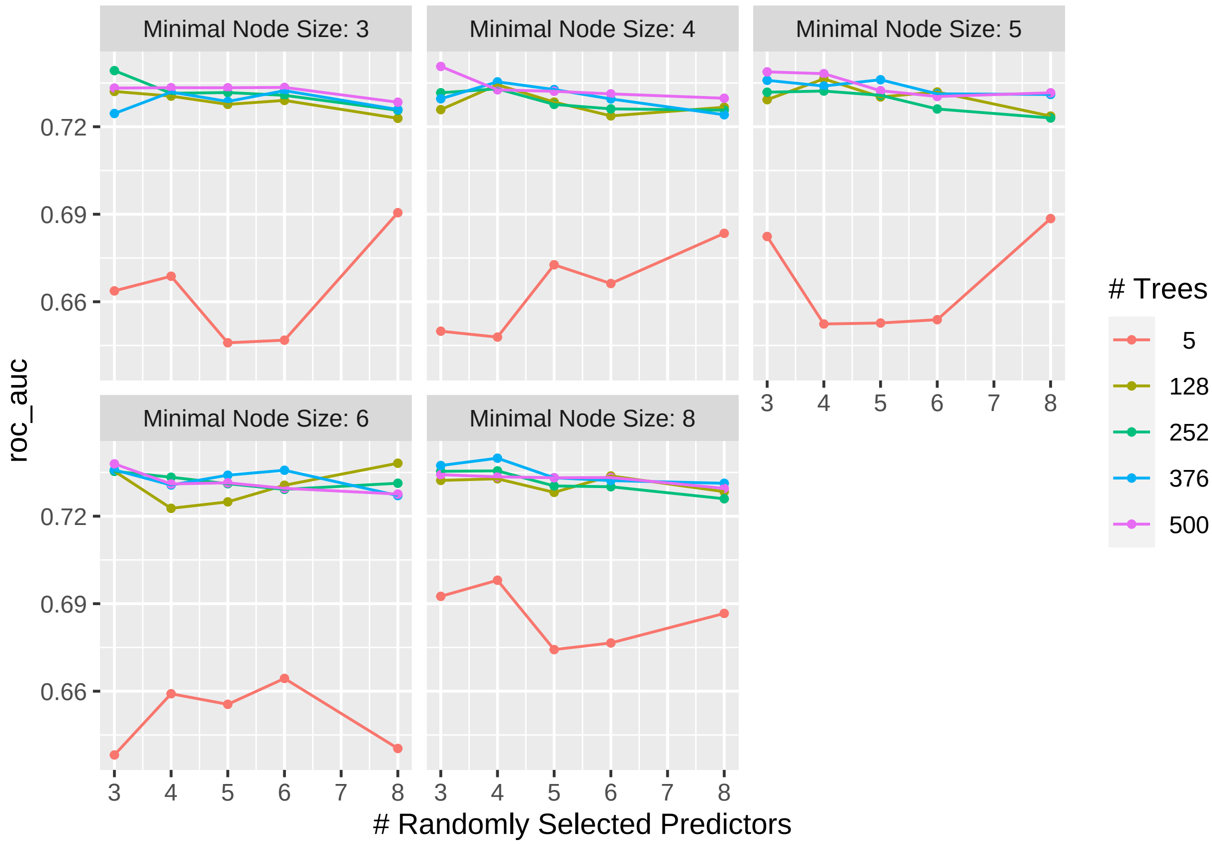
```r
library(ranger)
```

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

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

```r
rf_tune <- tune_grid(
  rf_wf,
  resamples = poke_folds,
  grid = rf_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
autoplot(rf_tune)
```



As the graph above has shown, it seems that when the number of trees is higher than 128 (chosed in the graph), the roc_auc is significantly higher than the case that has only 5 trees. However, when the number of trees gets even higher, it does not seem to make too many differences;
Also, it seems that the number of nodes does not have strong influence on the result; Furthermore, it is shown on the graph that has the number of randomly selected predictors increases, the roc_auc actually tends to decreases.

Question 7:

```
best_random_roc_auc <- collect_metrics(rf_tune) %>%
  arrange(-mean) %>%
  head(1)
best_random_roc_auc
```

```
## # A tibble: 1 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     3   500     4 roc_auc hand_till  0.741     5  0.0139 Preprocessor1_Model0~
```

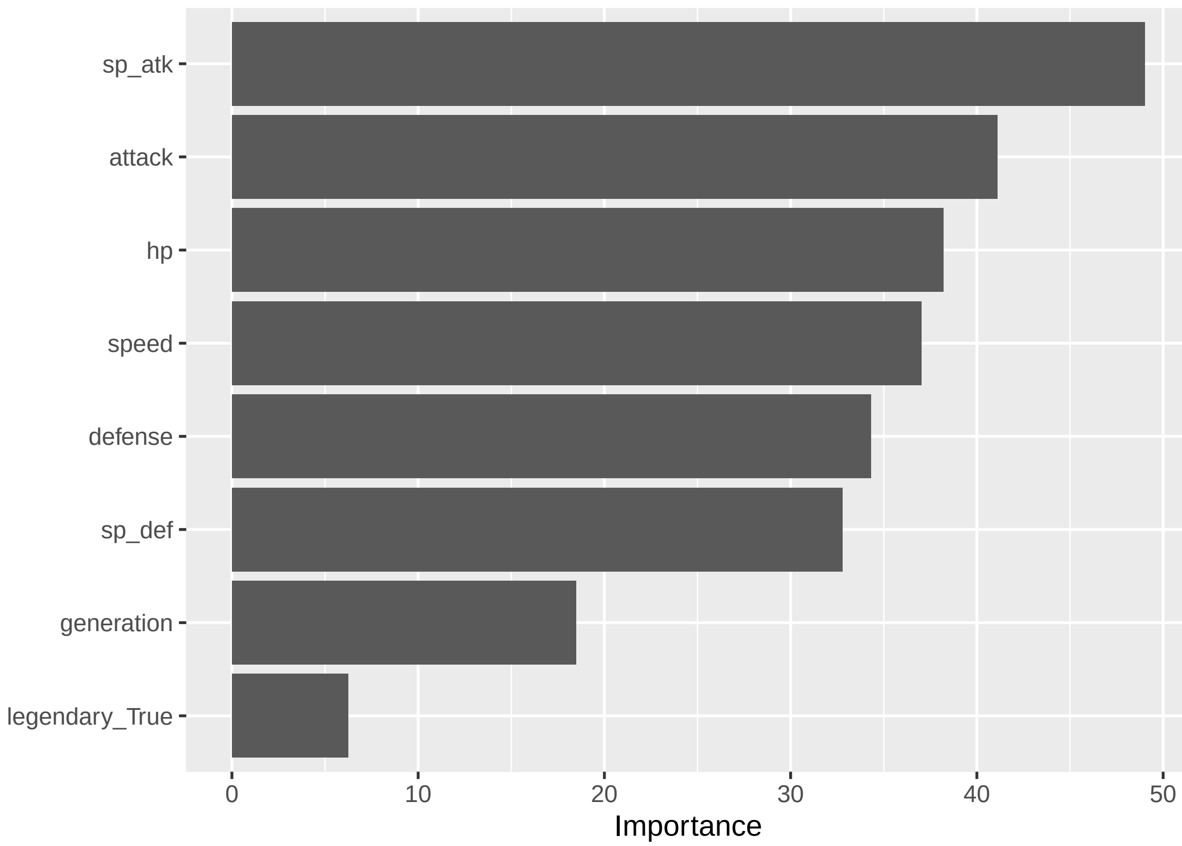
Thus, as it is shown above, the best roc_auc in this random forest model is 0.7420186;

Question 8:

```r
best_model <- select_best(rf_tune, metric = "roc_auc")
final_rf <- finalize_workflow(rf_wf, best_model)
final_fit <- fit(final_rf, data = poke_train)
```

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

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



As it is shown in the graph above, “sp_atk” has the greatest importance as predictor in this model; On the opposite, “legendary_True” is hast the least importance, which might be reasonably explained because the number of legendary pokemon is too small so that this predictor does not affect much to the overall model.

Question 9:

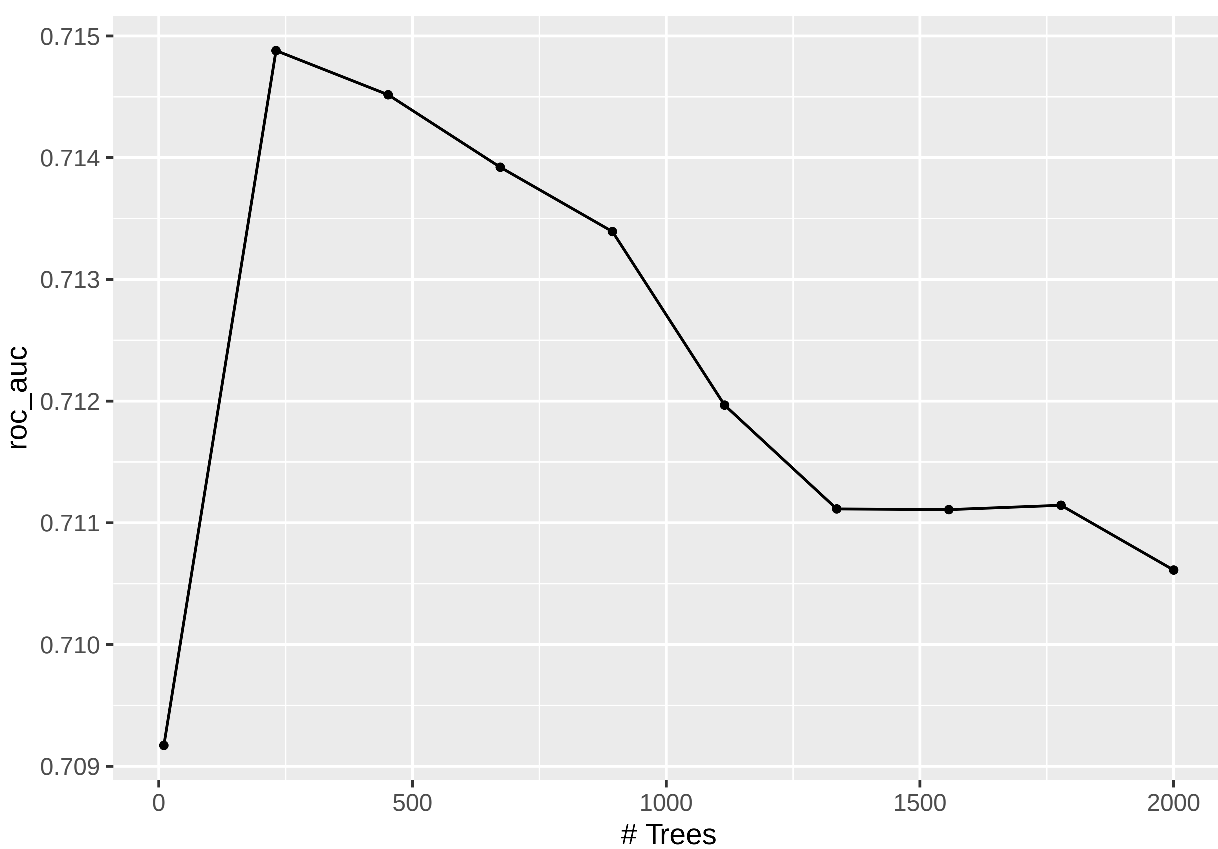
```r
boost_spec <- boost_tree(trees = tune()) %>%
  set_engine("xgboost") %>%
```

```
  set_mode("classification")

boost_wf <- workflow() %>%
  add_model(boost_spec) %>%
  add_recipe(poke_recipe)

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

```
boost_tune <- tune_grid(
  boost_wf,
  resamples = poke_folds,
  grid = boost_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...
```

```
autoplot(boost_tune)
```



```
best_boost_roc_auc <- collect_metrics(boost_tune) %>%
  arrange(-mean) %>%
  head(1)
best_boost_roc_auc
```

```
## # A tibble: 1 x 7
##   trees .metric .estimator  mean     n std_err .config
##   <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   231 roc_auc hand_till  0.715     5  0.0113 Preprocessor1_Model02
```

As it can be seen in the graph and data above: The roc_auc of my best performing model is 0.7148797, when trees = 231;

Question 10:

```
models_roc_auc_mean<- c(best_tree_roc_auc$mean,
                    best_random_roc_auc$mean,
                    best_boost_roc_auc$mean)
names <- c("pruned tree", "random forest", "boosted tree")

compare <- tibble(models_roc_auc_mean,names) %>%
  arrange(-models_roc_auc_mean)

compare
```

```
## # A tibble: 3 x 2
##   models_roc_auc_mean names
##                 <dbl> <chr>
## 1               0.741 random forest
## 2               0.715 boosted tree
## 3               0.641 pruned tree
```

As it is shown, the random forest model has the highest roc_auc and thus has the best performance. And thus we use the random forest model then.

```
best_model <- select_best(rf_tune)

rf_final <- finalize_workflow(rf_wf, best_model)

rf_final_fit <- fit(rf_final, data = poke_train)
```

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

```
augmented_result <- augment(rf_final_fit, new_data = poke_test)

augment(rf_final_fit, new_data = poke_test) %>%
  accuracy(truth = type_1, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.394
```

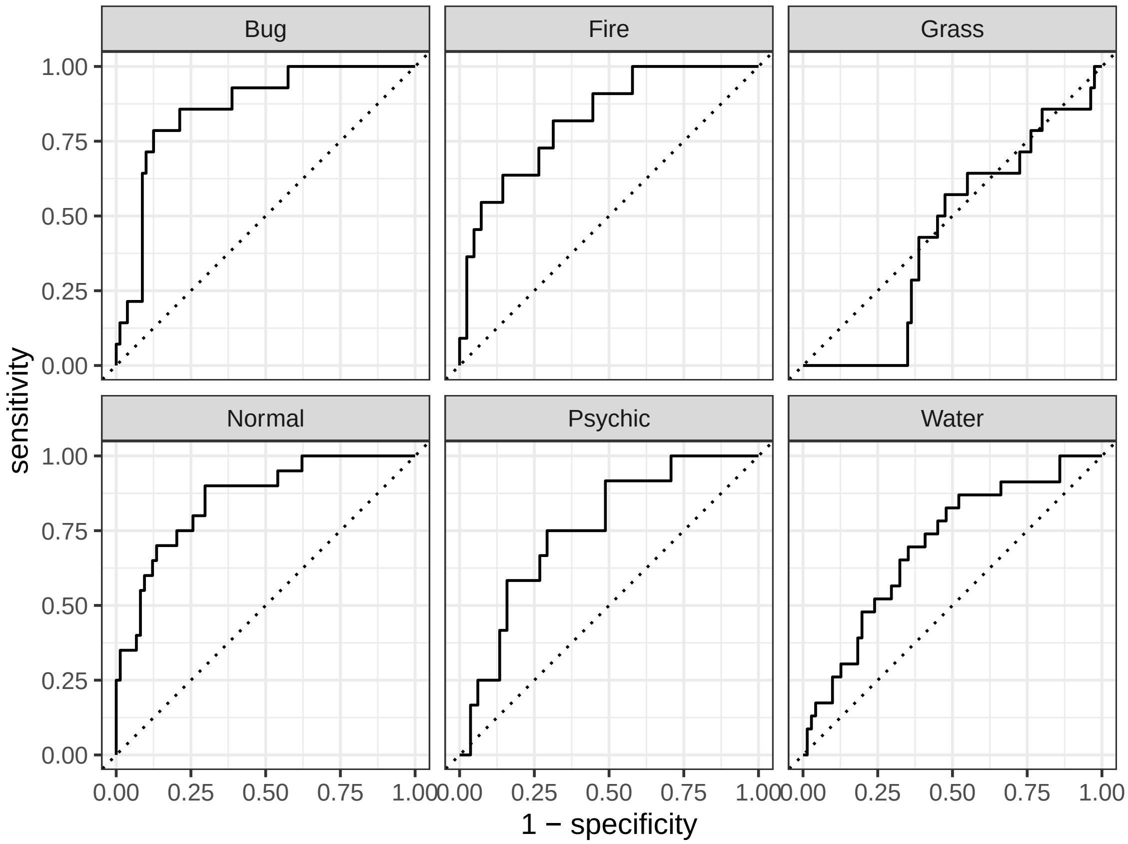
```
predicted_result <- augmented_result[c('type_1',
                                       '.pred_class',
                                       '.pred_Bug',
                                       '.pred_Fire',
                                       '.pred_Grass',
                                       '.pred_Normal',
                                       '.pred_Psychic',
                                       '.pred_Water')]
head(predicted_result)
```

```
## # A tibble: 6 x 8
##   type_1 .pred_class .pred_Bug .pred_Fire .pred_Grass .pred_Normal .pred_Psychic
##   <fct>  <fct>           <dbl>      <dbl>       <dbl>        <dbl>         <dbl>
## 1 Grass  Water          0.0603     0.121       0.16         0.0542        0.202
## 2 Grass  Water          0.0095     0.111       0.177        0.0302        0.124
## 3 Water  Grass          0.103      0.0552      0.517        0.0843        0.0147
## 4 Bug    Bug            0.505      0.0108      0.114        0.168         0.0248
## 5 Bug    Bug            0.398      0.182       0.0565       0.252         0.0377
## 6 Normal Normal         0.197      0.0487      0.123        0.495         0.015
## # ... with 1 more variable: .pred_Water <dbl>
```

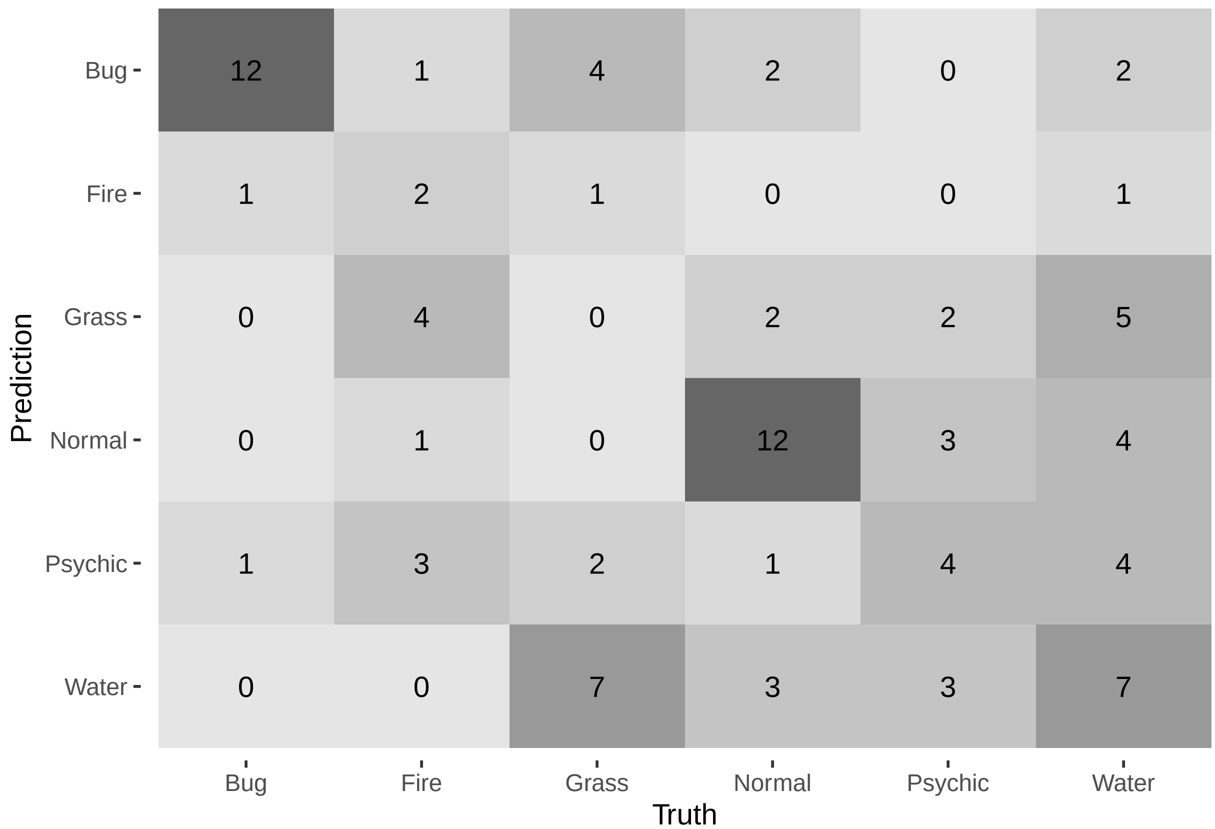
```
roc_auc(predicted_result, type_1, c(.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic, .p
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.736
```

```
roc_curve(predicted_result, type_1, c(.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic,.
```




```
predicted_result %>%
  conf_mat(type_1, .pred_class) %>%
  autoplot(type = "heatmap")
```



As it is shown above, Bug, Normal, and Water are predicted most accurately, while Grass is worst predicted.