PSTAT 231 Hw6 muxi

muxi

2022-11-26

Exercise 1

```
library(tidymodels)
```

```
## ── Attaching packages ──────────────────────────────────── tidymodels 1.0.0 ──
```

```
## ✓ broom        1.0.1      ✓ recipes      1.0.2
## ✓ dials        1.0.0      ✓ rsample      1.1.0
## ✓ dplyr        1.0.10     ✓ tibble       3.1.8
## ✓ ggplot2      3.3.6      ✓ tidyr        1.2.1
## ✓ infer        1.0.3      ✓ tune         1.0.1
## ✓ modeldata    1.0.1      ✓ workflows    1.1.0
## ✓ parsnip      1.0.2      ✓ workflowsets 1.0.0
## ✓ purrr        0.3.4      ✓ yardstick    1.1.0
```

```
## ── Conflicts ─────────────────────────────────────────── tidymodels_conflicts() ──
## ✕ purrr::discard() masks scales::discard()
## ✕ dplyr::filter()  masks stats::filter()
## ✕ dplyr::lag()     masks stats::lag()
## ✕ recipes::step()  masks stats::step()
## • Use suppressPackageStartupMessages() to eliminate package startup messages
```

```
library(ISLR)
library(ISLR2)
```

```
##
## 载入程辑包：'ISLR2'
```

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

```
library(tidyverse)
```

```
## ── Attaching packages
## ───────────────────────────────────────
## tidyverse 1.3.2 ──
```

```
## ✓ readr     2.1.3      ✓ forcats 0.5.2
## ✓ stringr 1.4.1
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ readr::col_factor() masks scales::col_factor()
## ✖ purrr::discard()    masks scales::discard()
## ✖ dplyr::filter()     masks stats::filter()
## ✖ stringr::fixed()    masks recipes::fixed()
## ✖ dplyr::lag()        masks stats::lag()
## ✖ readr::spec()       masks yardstick::spec()
```

```
tidymodels_prefer()
library(janitor)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

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

```
## 载入需要的程辑包：rpart
##
## 载入程辑包：'rpart'
##
## The following object is masked from 'package:dials':
##
##     prune
```

```
library(vip)
library(janitor)
library(randomForest)
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(corrr)
library(corrplot)
```

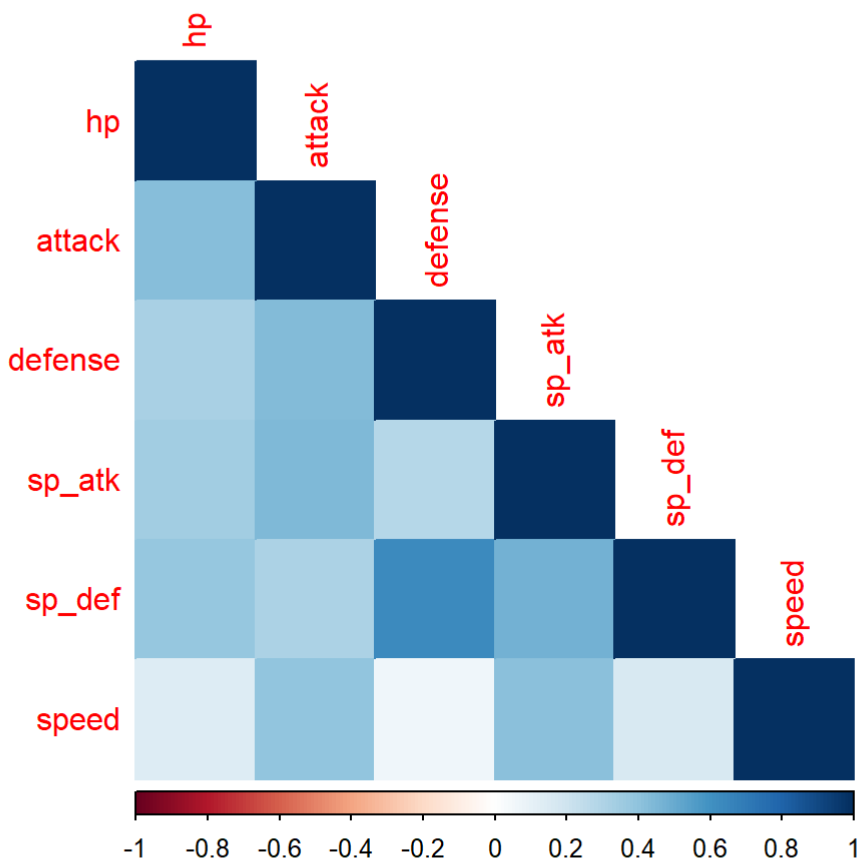
```
## corrplot 0.92 loaded
```

```
library(xgboost)
library(ranger)
```

```
#Use clean_names
data=read.csv("Pokemon.csv")%>%clean_names()
#Filter out the rarer Pokémon types
data=data %>% filter(type_1=="Bug"|type_1=="Fire"|type_1=="Grass"|type_1=="Normal"|type_1=="W
ater"|type_1=="Psychic")
#Convert type_1 and legendary to factors
data[,3]=as.factor(data[,3])
data[,12]=as.factor(data[,12])
data[,13]=as.factor(data[,13])
#Do an initial split of the data
set.seed(1215)
data_split=initial_split(data, strata = type_1, prop = 0.7)
data_train=training(data_split)
data_test=testing(data_split)
#Fold the training set using v-fold cross-validation
data_folds=vfold_cv(data_train, v = 5,strata = type_1)
#Set up a recipe
Pokemon_recipe=recipe(type_1 ~ legendary+generation+sp_atk+attack+speed+defense+hp+sp_def, da
ta = data_train) %>% step_dummy(legendary,generation)%>%step_center(all_predictors())%>%step_
scale(all_predictors())
```

Exercise 2

```
data_train %>% select(where(is.numeric)) %>% select(-total) %>% select(-x)%>% cor() %>% corrp
lot(type = 'lower', method = 'color')
```

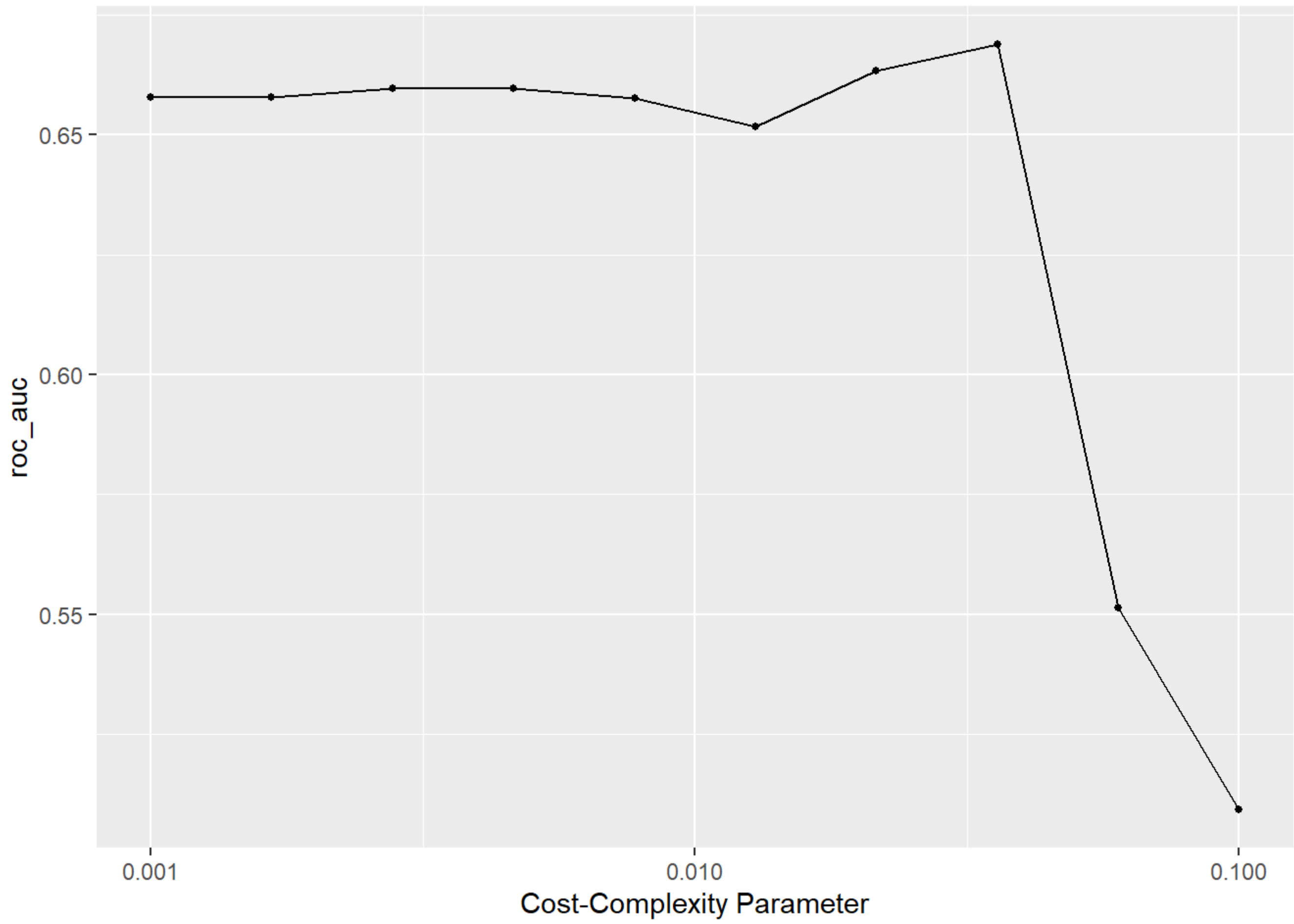


I removed the Total predictor. By the definition of Total: sum of all stats that come after this, a general guide to how strong a pokemon is, we know that the sum of all the other variables is a perfect predictor of Total.

We notice that SP Def has a strong positive correlation with defense. In my opinion, these two predictors both reveal Pokémon's defensive properties.

Exercise 3

```
# model
class_tree_spec=decision_tree() %>%
  set_engine("rpart") %>%
  set_mode("classification")

# workflow
class_tree_wf <- workflow() %>%
  add_model(class_tree_spec %>% set_args(cost_complexity = tune())) %>%
  add_recipe(Pokemon_recipe)
#tune
param_grid <- grid_regular(cost_complexity(range = c(-3, -1)), levels = 10)
#fit
tune_res <- tune_grid(
  class_tree_wf,
  resamples = data_folds,
  grid = param_grid,
  metrics = metric_set(roc_auc)
)
autoplot(tune_res)
```



From the graph, we could see that the roc_auc keeps stable and then decreasing as Cost-Complexity Parameter increases. It is clear that a single decision tree don't perform better with a smaller or larger complexity penalty. The roc_auc reaches the highest value at Cost-Complexity Parameter = 0.03 approximately.

Exercise 4

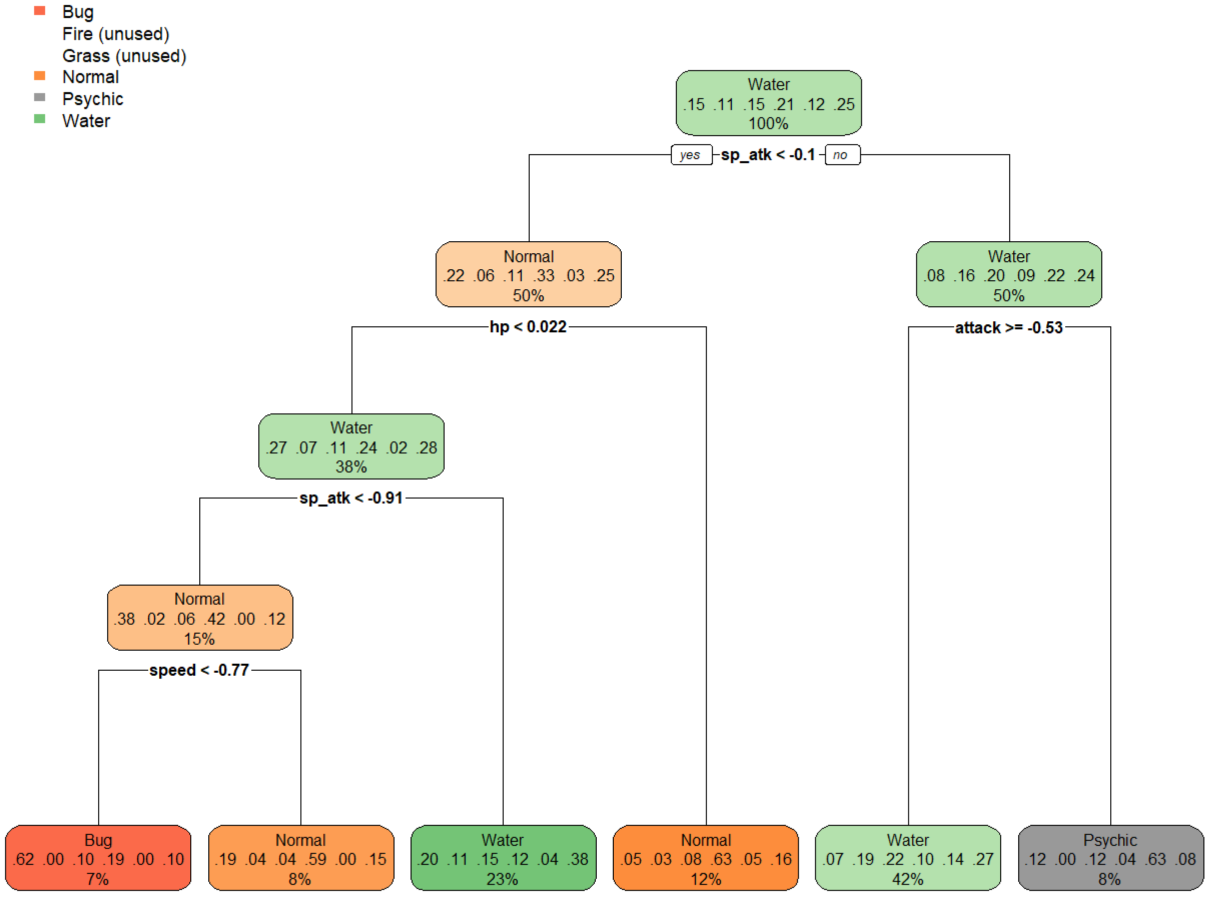
```
#Best roc_auc
collect_metrics(tune_res)%>% arrange(-mean)
```

```
## # A tibble: 10 × 7
##    cost_complexity .metric .estimator  mean     n std_err .config
##              <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         0.0359  roc_auc hand_till  0.669     5 0.0179  Preprocessor1_Model08
##  2         0.0215  roc_auc hand_till  0.663     5 0.0174  Preprocessor1_Model07
##  3         0.00278 roc_auc hand_till  0.660     5 0.0213  Preprocessor1_Model03
##  4         0.00464 roc_auc hand_till  0.660     5 0.0213  Preprocessor1_Model04
##  5         0.001   roc_auc hand_till  0.658     5 0.0221  Preprocessor1_Model01
##  6         0.00167 roc_auc hand_till  0.658     5 0.0221  Preprocessor1_Model02
##  7         0.00774 roc_auc hand_till  0.658     5 0.0216  Preprocessor1_Model05
##  8         0.0129  roc_auc hand_till  0.652     5 0.0203  Preprocessor1_Model06
##  9         0.0599  roc_auc hand_till  0.551     5 0.0237  Preprocessor1_Model09
## 10         0.1     roc_auc hand_till  0.509     5 0.00921 Preprocessor1_Model10
```

From the graph, we could see that the roc_auc of my best-performing pruned decision tree on the folds is 0.6688242.

Exercise 5

```
#fit
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 = data_train)
#visualize
class_tree_final_fit %>%
  extract_fit_engine() %>%
  rpart.plot(roundint=FALSE)
```



Exercise 5

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

# workflow
rf_wf=workflow() %>%
  add_model(rf_spec) %>%
  add_recipe(Pokemon_recipe)
```

mtry: An integer indicating how many predictors will be sampled at random for the tree models at each split.

trees: An integer representing how many trees are included in the ensemble.

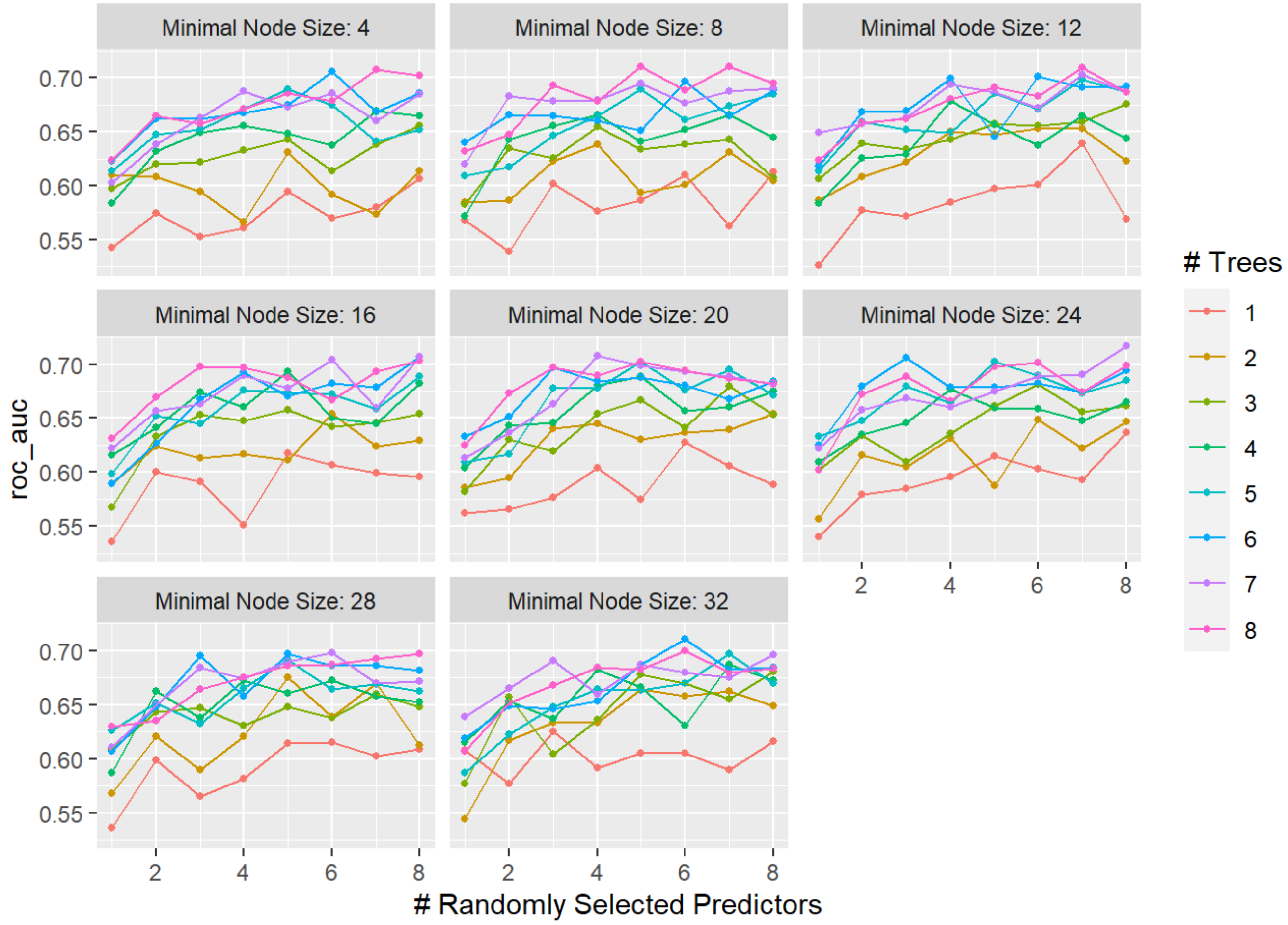
min_n: An integer representing the minimal quantity of data points in a node necessary for the node to be further divided.

```
#Create the regular grid
para_grid=grid_regular(mtry(range = c(1, 8)),trees(range = c(1, 8)),min_n(range = c(4, 32)),
 levels = c(mtry = 8, trees = 8,min_n=8))
```

As mtry is equal to the number of predictors. We have to fit a model type1 ~ legendary, generation, sp_atk, attack, speed, defense, hp, and sp_def. Hence, we totally have 8 predictors and the maximum value of mtry is 8.

Exercise 6

```
#fit
rf_res=tune_grid(
  rf_wf,
  resamples = data_folds,
  grid = para_grid,
  metrics = metric_set(roc_auc)
)
autoplot(rf_res)
```



From all graphs we could see that models with higher trees preform better. Moreover, the curves become more smooth as min_n increases. What's more, the roc_aucs gradually increase with the value of mtry in all graphs.

mtry=8,min_n=24,trees=7 seem to yield the best performance.

Exercise 7

```
# Best roc_auc
collect_metrics(rf_res)%>% arrange(-mean)
```

```
## # A tibble: 512 × 9
##     mtry trees min_n .metric .estimator  mean     n std_err .config
##    <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     8     7    24 roc_auc hand_till  0.717     5 0.0225  Preprocessor1_Model…
##  2     6     6    32 roc_auc hand_till  0.711     5 0.0149  Preprocessor1_Model…
##  3     5     8     8 roc_auc hand_till  0.710     5 0.00815 Preprocessor1_Model…
##  4     7     8     8 roc_auc hand_till  0.710     5 0.0268  Preprocessor1_Model…
##  5     7     8    12 roc_auc hand_till  0.709     5 0.0213  Preprocessor1_Model…
##  6     4     7    20 roc_auc hand_till  0.707     5 0.0185  Preprocessor1_Model…
##  7     8     6    16 roc_auc hand_till  0.707     5 0.0139  Preprocessor1_Model…
##  8     7     8     4 roc_auc hand_till  0.707     5 0.0134  Preprocessor1_Model…
##  9     8     7    16 roc_auc hand_till  0.707     5 0.0159  Preprocessor1_Model…
## 10     3     6    24 roc_auc hand_till  0.706     5 0.0183  Preprocessor1_Model…
## # … with 502 more rows
```

The roc_auc of the best-performing random forest model on the folds is 0.7166293.

Exercise 8

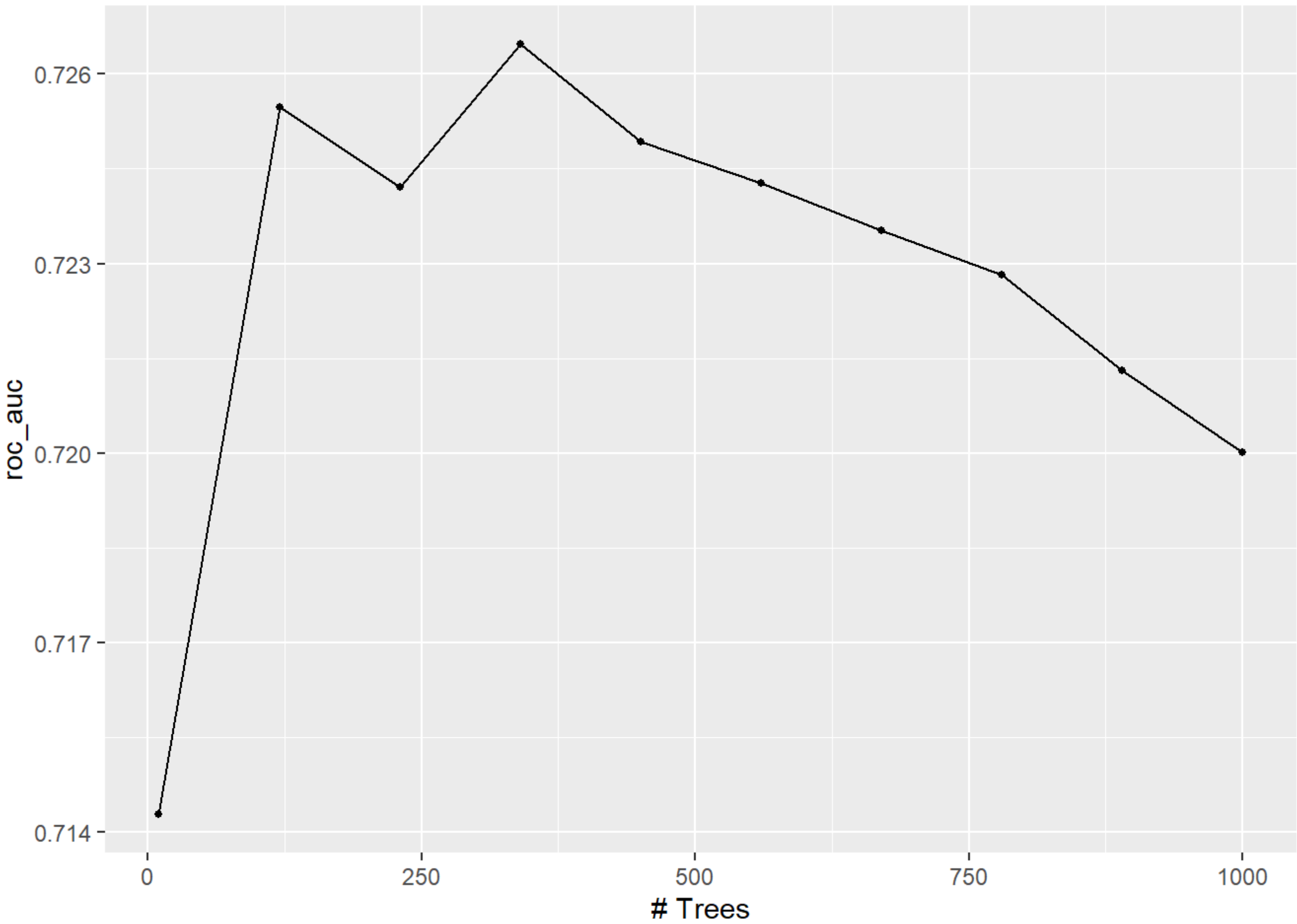
```
#fit
rf_best_para=select_best(rf_res, metric = "roc_auc")
rf_final=finalize_workflow(rf_wf, rf_best_para)
rf_final_fit=fit(rf_final, data = data_train)
augment(rf_final_fit, new_data = data_train) %>%
  accuracy(truth = type_1, estimate = .pred_class)
```

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.698
```

```
#vip(rf_final_fit)
```

Exercise 9

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

boost_wf=workflow() %>%
  add_model(boost_spec) %>%
  add_recipe(Pokemon_recipe)

para_grid=grid_regular(trees(range = c(10,1000)), levels = c(trees = 10))

boost_res=tune_grid(
  boost_wf,
  resamples = data_folds,
  grid = para_grid,
  metrics = metric_set(roc_auc)
)
autoplot(boost_res)
```




```
collect_metrics(boost_res)%>% arrange(-mean)
```

```
## # A tibble: 10 × 7
##    trees .metric .estimator  mean     n std_err .config
##    <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1   340 roc_auc hand_till  0.726     5 0.0109  Preprocessor1_Model04
##  2   120 roc_auc hand_till  0.725     5 0.00920 Preprocessor1_Model02
##  3   450 roc_auc hand_till  0.725     5 0.0113  Preprocessor1_Model05
##  4   560 roc_auc hand_till  0.724     5 0.0117  Preprocessor1_Model06
##  5   230 roc_auc hand_till  0.724     5 0.0107  Preprocessor1_Model03
##  6   670 roc_auc hand_till  0.724     5 0.0118  Preprocessor1_Model07
##  7   780 roc_auc hand_till  0.723     5 0.0114  Preprocessor1_Model08
##  8   890 roc_auc hand_till  0.721     5 0.0113  Preprocessor1_Model09
##  9  1000 roc_auc hand_till  0.720     5 0.0117  Preprocessor1_Model10
## 10    10 roc_auc hand_till  0.714     5 0.0128  Preprocessor1_Model01
```

The roc_auc of the best-performing boosted tree model on the folds is 0.7264726.

Exercise 10

```
ROC_AUC=c(0.6688242, 0.7166293, 0.7264726)
models=c("pruned tree", "random forest", "boosted tree")
results=tibble(ROC_AUC = ROC_AUC, models = models)
results %>%
  arrange(-ROC_AUC)
```

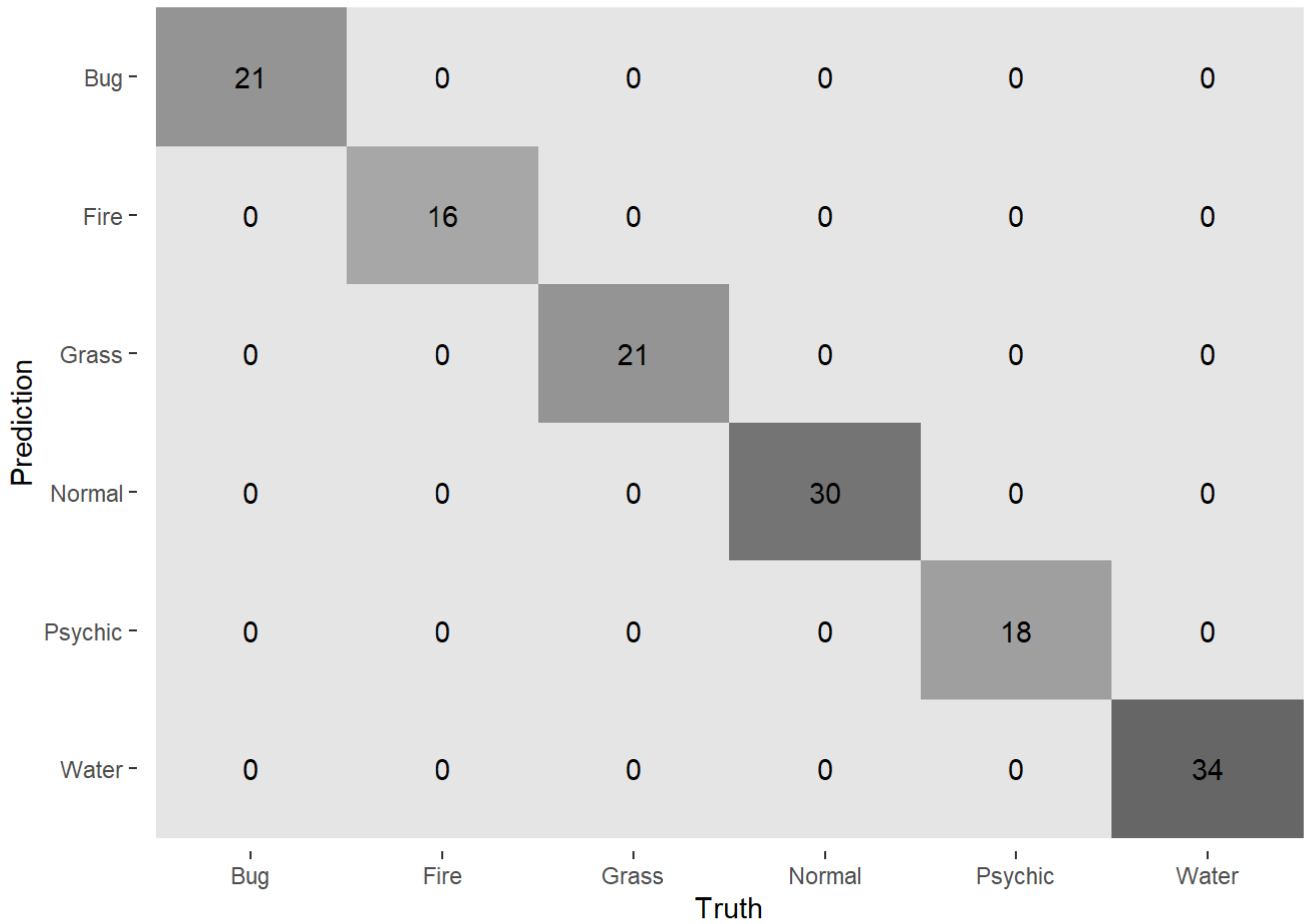
```
## # A tibble: 3 × 2
##   ROC_AUC models
##     <dbl> <chr>
## 1   0.726 boosted tree
## 2   0.717 random forest
## 3   0.669 pruned tree
```

Boosted tree models performed best on the folds.

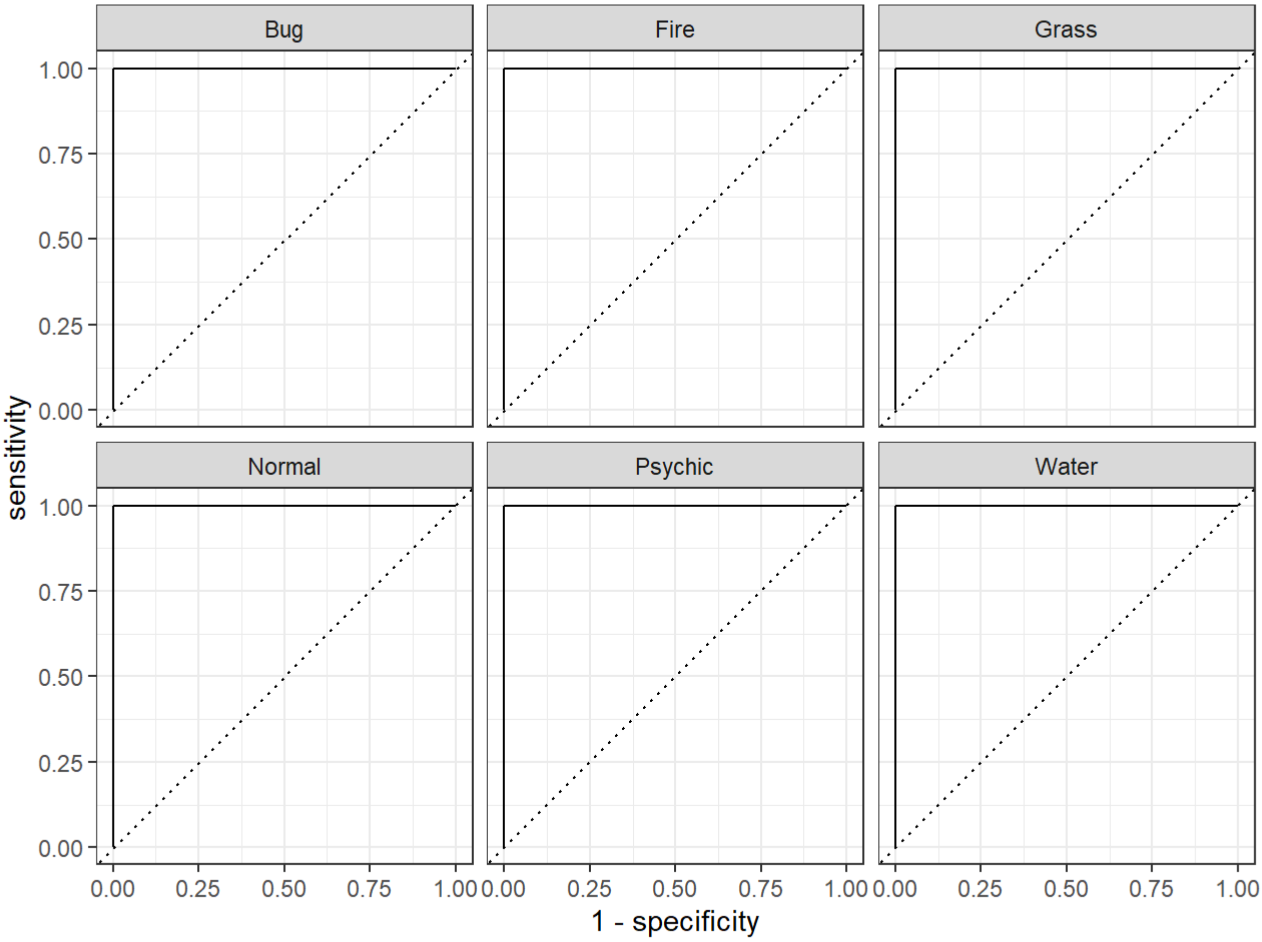
```
best_para=select_best(boost_res, metric = "roc_auc")
final=finalize_workflow(boost_wf, best_para)
final_fit=fit(final, data = data_test)
k1=predict(final_fit, data_test, type="prob")
k2=predict(final_fit, data_test, type="class")
k3=cbind(k1,k2)
Pre=cbind(data_test[,3],k3)
Pre %>%roc_auc(data_test[,3],  .pred_Bug: .pred_Water)
```

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

```
augment(final_fit, new_data = data_test) %>%
  conf_mat(truth = type_1, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```



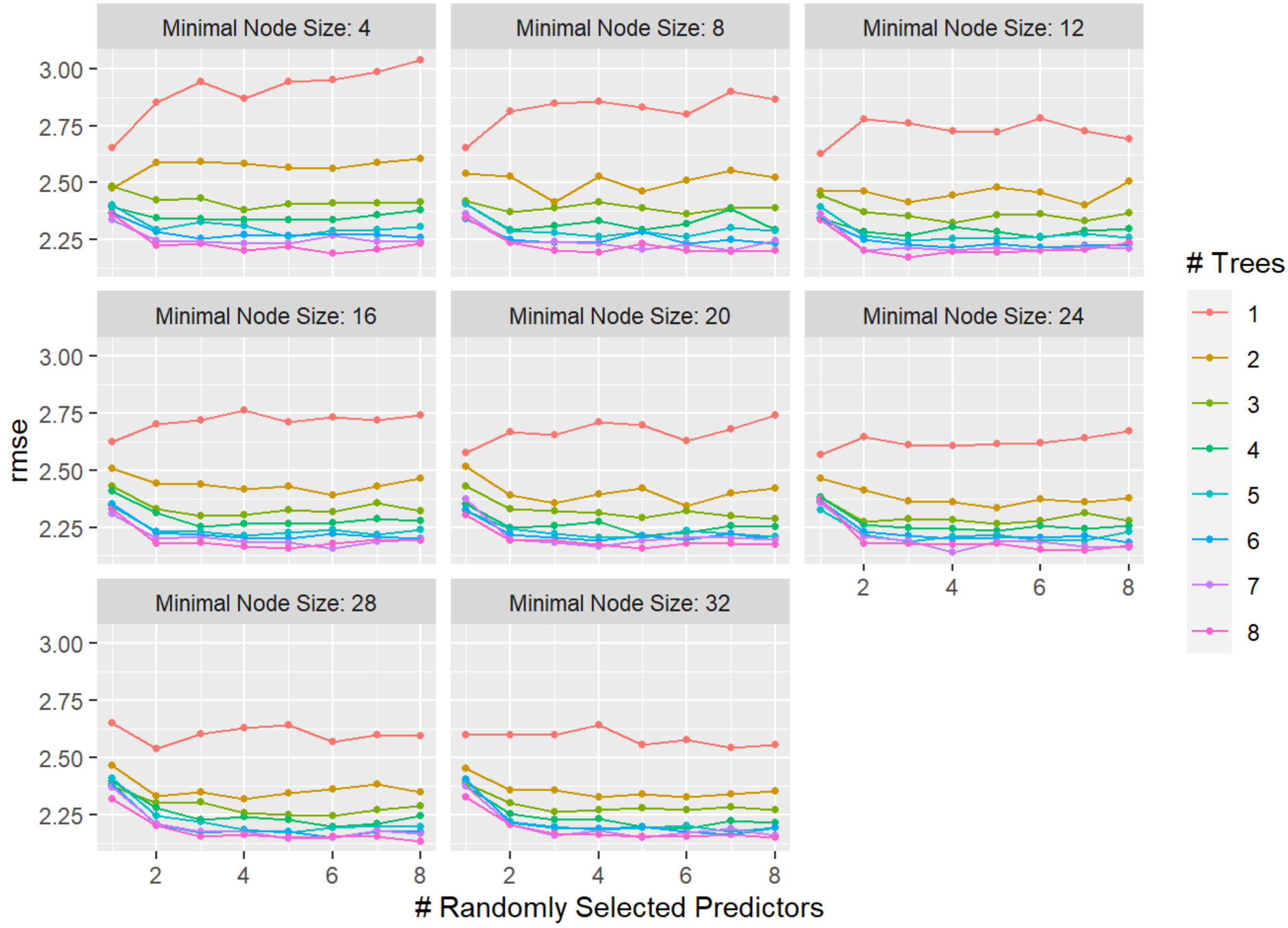

```
Pre %>%
  roc_curve(data_test[,3],  .pred_Bug: .pred_Water) %>%
  autoplot()
```



All classes was predicted prefectly by the model due to overfitting.

Exercise 11

```
abadata=read.csv("abalone.csv")
abadata=mutate(abadata,age=rings+1.5)
abadata_split = initial_split(abadata, prop = 0.80)
abadata_train = training(abadata_split)
abadata_test = testing(abadata_split)
train=select(abadata_train,-c(rings))
test=select(abadata_test,-c(rings))

abadata_recipe = recipe(age~ ., data = train)
recipe=abadata_recipe%>%
  step_dummy(all_nominal_predictors())%>%
  step_interact(terms = ~ starts_with("type"):shucked_weight)%>%
  step_interact(terms = ~ longest_shell:diameter)%>%
  step_interact(terms = ~ shucked_weight:shell_weight)%>%
  step_center(all_nominal_predictors())%>%
  step_scale(all_nominal_predictors())

abarf_spec=rand_forest(mtry = tune(),trees=tune(),min_n=tune()) %>%
  set_engine("ranger") %>%
  set_mode("regression")

# workflow
abarf_wf=workflow() %>%
  add_model(abarf_spec) %>%
  add_recipe(recipe)

para_grid=grid_regular(mtry(range = c(1, 8)),trees(range = c(1, 8)),min_n(range = c(4, 32)),
 levels = c(mtry = 8, trees = 8,min_n=8))

abadata_folds=vfold_cv(train, v = 5,strata = age)

aba_res=tune_grid(
  abarf_wf,
  resamples = abadata_folds,
  grid = para_grid,
  metrics = metric_set(rmse)
)
autoplot(aba_res)
```




```
collect_metrics(aba_res)%>% arrange(-mean)
```

```
## # A tibble: 512 × 9
##     mtry trees min_n .metric .estimator  mean     n std_err .config
##    <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     8     1     4 rmse    standard    3.04     5  0.0690 Preprocessor1_Model…
##  2     7     1     4 rmse    standard    2.99     5  0.0321 Preprocessor1_Model…
##  3     6     1     4 rmse    standard    2.95     5  0.0710 Preprocessor1_Model…
##  4     5     1     4 rmse    standard    2.94     5  0.0636 Preprocessor1_Model…
##  5     3     1     4 rmse    standard    2.94     5  0.0631 Preprocessor1_Model…
##  6     7     1     8 rmse    standard    2.90     5  0.0597 Preprocessor1_Model…
##  7     4     1     4 rmse    standard    2.87     5  0.0364 Preprocessor1_Model…
##  8     8     1     8 rmse    standard    2.87     5  0.0600 Preprocessor1_Model…
##  9     4     1     8 rmse    standard    2.86     5  0.0738 Preprocessor1_Model…
## 10     2     1     4 rmse    standard    2.85     5  0.0942 Preprocessor1_Model…
## # … with 502 more rows
```

```
best_para=select_best(aba_res, metric = "rmse")
abarf=finalize_workflow(abarf_wf, best_para)
abarf_fit=fit(abarf, data = train)
augment(abarf_fit, new_data = test) %>%
  rmse(truth = age, estimate = .pred)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        2.39
```