131HW6

Scott Shang (8458655)

May 26, 2022

Question1

```r
library("tidyverse")
```

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

## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## 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()
```

```r
library("tidymodels")
```

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

## v broom        0.8.0     v rsample      0.1.1
## v dials        0.1.1     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()
## * Dig deeper into tidy modeling with R at https://www.tmwr.org
```

```r
library("dplyr")
library("yardstick")
library(tidymodels)
library(readr)
library(pROC)
```

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

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(discrim)
```

```
##
## Attaching package: 'discrim'

## The following object is masked from 'package:dials':
##
##     smoothness
```

```
library(poissonreg)
library(corrr)
library(klaR)
```

```
## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(knitr)
library(MASS)
library(ggplot2)
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
```

```r
library(janitor)
```

```
##
## Attaching package: 'janitor'

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

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

```
## Loading required package: rpart

##
## Attaching package: 'rpart'

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

```r
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
```

```r
library(ranger)
```

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

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

```r
library(vip)
```

```
##
## Attaching package: 'vip'

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

```r
library(xgboost)
```

```
##
## Attaching package: 'xgboost'

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

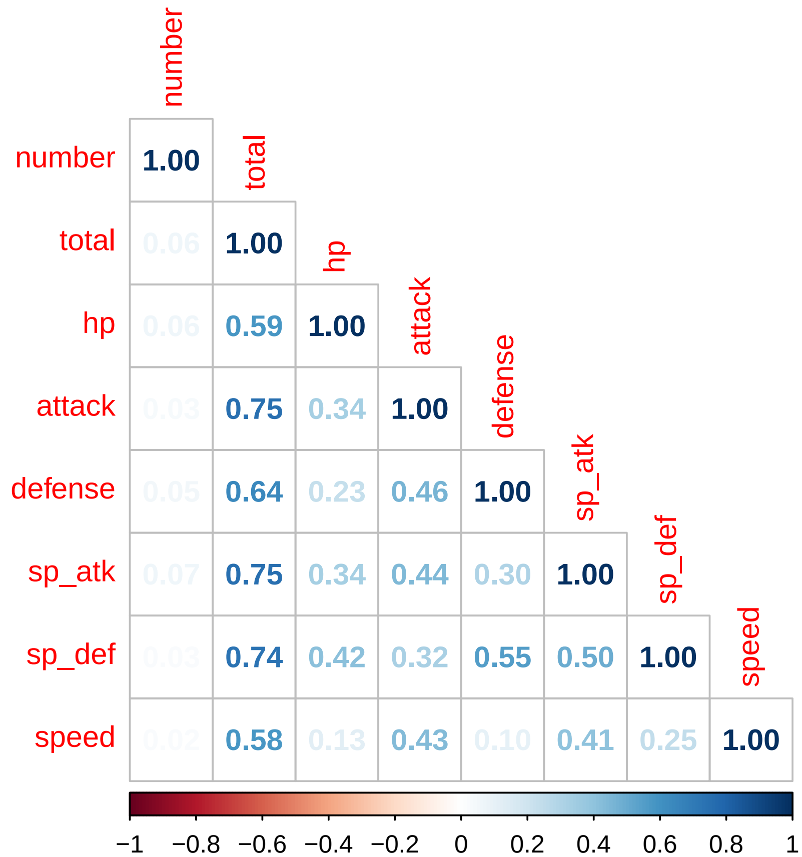
```r
pkm=read_csv('Pokemon.csv')
```

```
## Rows: 800 Columns: 13

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Name, Type 1, Type 2
## dbl (9): #, Total, HP, Attack, Defense, Sp. Atk, Sp. Def, Speed, Generation
## lgl (1): Legendary
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
pkm=clean_names(pkm)
pkm=filter(pkm,type_1 %in% c("Bug","Fire","Grass","Normal","Water","Psychic"))
pkm$type_1=as.factor(pkm$type_1)
pkm$legendary=as.factor(pkm$legendary)
pkm$generation=as.factor(pkm$generation)
set.seed(1234)
pkm_split=initial_split(pkm,prop=0.70,strata=type_1)
train=training(pkm_split)
test=testing(pkm_split)
folds=vfold_cv(train,v=5,strata=type_1)
rcp=recipe(type_1~legendary+generation+sp_atk+attack+speed+defense+hp+sp_def,data=train) %>%
  step_dummy(legendary) %>%
  step_dummy(generation) %>%
  step_normalize(all_predictors())
```

Question2

```
pkm %>%
  dplyr::select(where(is.numeric)) %>%
  cor() %>%
  corrplot(method='number',type='lower')
```

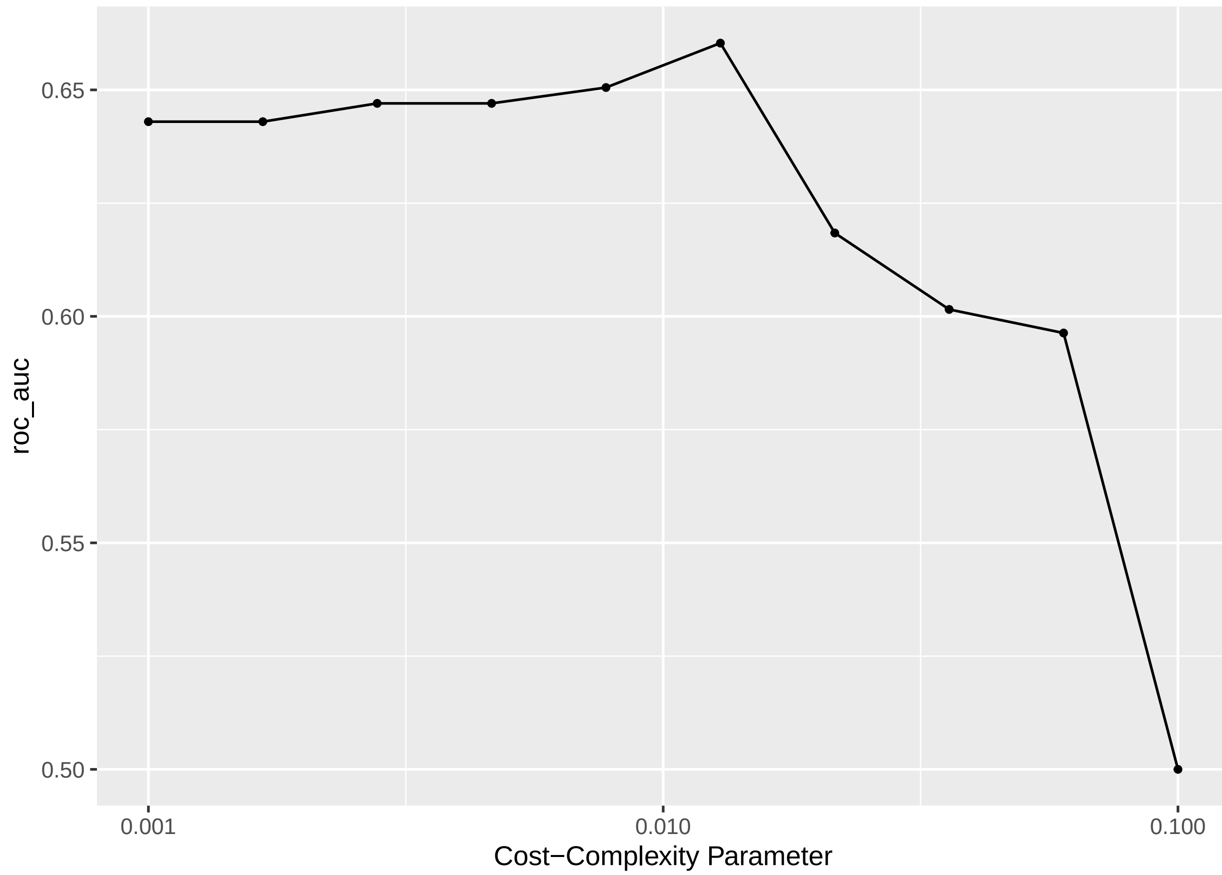


I choose to include all the numeric/continuous variables in this plot. We observe a strong correlation between total with other variables, which make sense because total is the sum of all stats. Also, we don't see any negative correlation, maybe because all the stats of a Pokemon tend to grow together.

Question3

```
tree_spec=decision_tree() %>%
  set_engine("rpart")
class_tree_spec=tree_spec %>%
  set_mode("classification")
class_tree_wf=workflow() %>%
  add_model(class_tree_spec %>%
      set_args(cost_complexity = tune())) %>%
  add_recipe(rcp)

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

class_tree_tune_res=tune_grid(
  class_tree_wf,
  resamples=folds,
  grid=class_tree_grid,
```

```
  metrics=metric_set(roc_auc)
)

autoplot(class_tree_tune_res)
```



We observe that a single decision tree perform better with a smaller complexity penalty overall, but it perform best on the middle range.

Question4

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

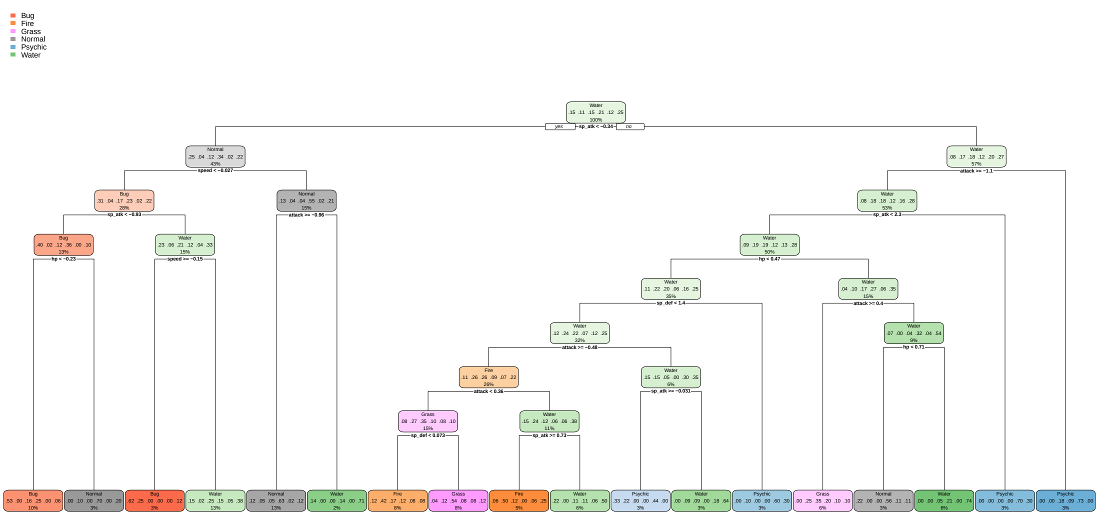
```
## # A tibble: 10 x 7
##    cost_complexity .metric .estimator  mean     n std_err .config
##              <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         0.0129  roc_auc hand_till  0.660     5 0.0141  Preprocessor1_Model06
##  2         0.00774 roc_auc hand_till  0.651     5 0.0163  Preprocessor1_Model05
##  3         0.00278 roc_auc hand_till  0.647     5 0.0186  Preprocessor1_Model03
##  4         0.00464 roc_auc hand_till  0.647     5 0.0186  Preprocessor1_Model04
##  5         0.001   roc_auc hand_till  0.643     5 0.0183  Preprocessor1_Model01
##  6         0.00167 roc_auc hand_till  0.643     5 0.0183  Preprocessor1_Model02
##  7         0.0215  roc_auc hand_till  0.618     5 0.00683 Preprocessor1_Model07
##  8         0.0359  roc_auc hand_till  0.602     5 0.0132  Preprocessor1_Model08
##  9         0.0599  roc_auc hand_till  0.596     5 0.0146  Preprocessor1_Model09
## 10         0.1     roc_auc hand_till  0.5       5 0       Preprocessor1_Model10
```

The roc_auc of our best-performing pruned decision tree on the folds is 0.66 in model 06.

Question5

```
class_tree_best=select_best(class_tree_tune_res)

class_tree_final=finalize_workflow(class_tree_wf,class_tree_best)

class_tree_final_fit=fit(class_tree_final,data=train)

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.binar
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.
```



Another Question5

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

```
rf_wf=workflow() %>%
  add_model(rf_spec) %>%
  add_recipe(rcp)

rf_grid=grid_regular(mtry(range=c(1,8)),trees(range=c(1,10)),min_n(range=c(1,10)),levels=8)

rf_grid
```

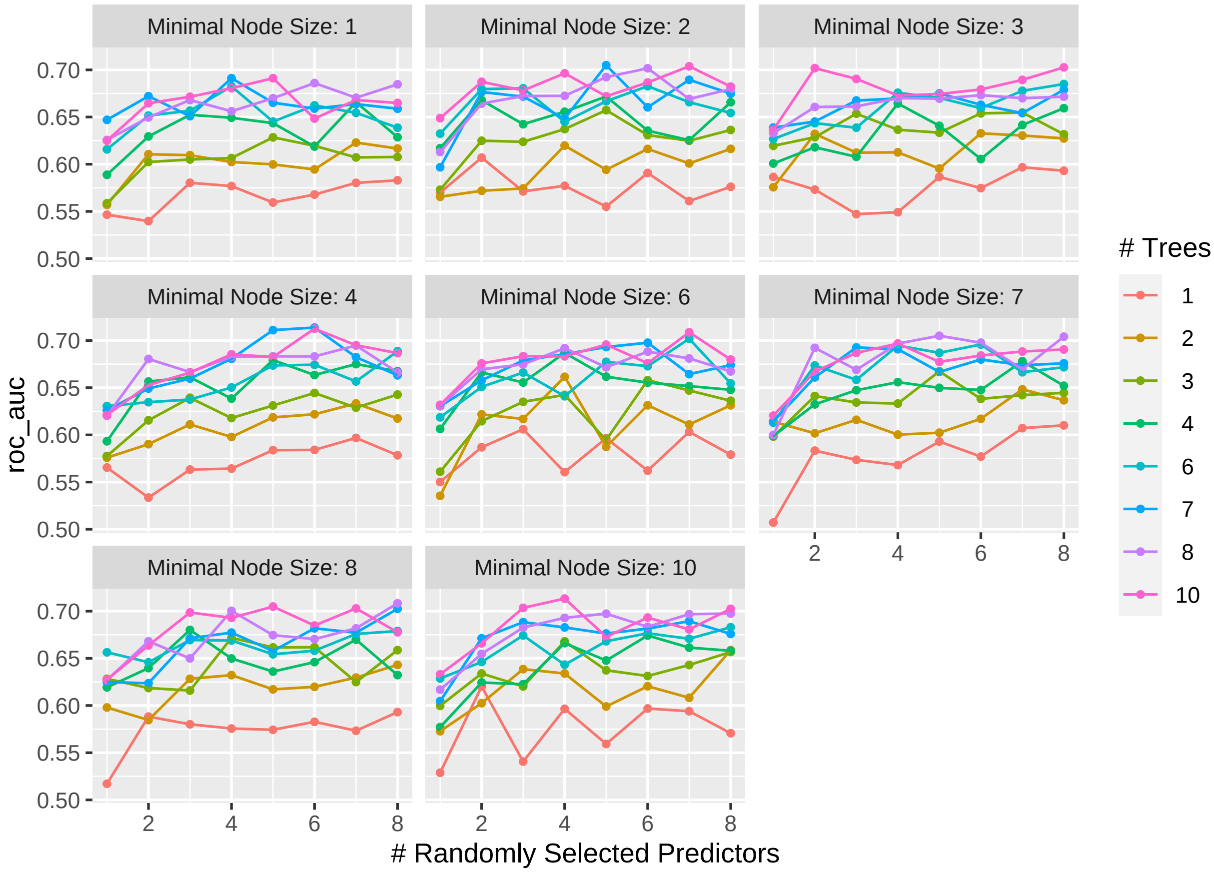
```
## # A tibble: 512 x 3
##     mtry trees min_n
##    <int> <int> <int>
##  1     1     1     1
##  2     2     1     1
##  3     3     1     1
##  4     4     1     1
##  5     5     1     1
##  6     6     1     1
##  7     7     1     1
##  8     8     1     1
##  9     1     2     1
## 10     2     2     1
## # ... with 502 more rows
```

mtry is the number of our selected predictors that we assign to each tree to make its decisions. trees is the number of trees we create in our forest. min_n is the minimum number of data values needed to create further split.

mtry should not be smaller than 8 because it can't exceed the number of predictors in our grid, if so, there is no subset of the predictors that can be chosen. And mtry=0 means we don't have predictors at all, which doesn't make sense. mtry=8 represents all predictors we have will be randomly sampled.

Question6

```
rf_tune_res=tune_grid(
  rf_wf,
  resamples=folds,
  grid=rf_grid,
  metrics=metric_set(roc_auc)
)

autoplot(rf_tune_res)
```



WE observe that the best performing models features 7, 8, or 10 trees. Minimal node size of 4 seemed to perform pretty good. Increasing the number of selected variables improves the performance. The number of selected should be at least 5 for the sake of performance.

Question7

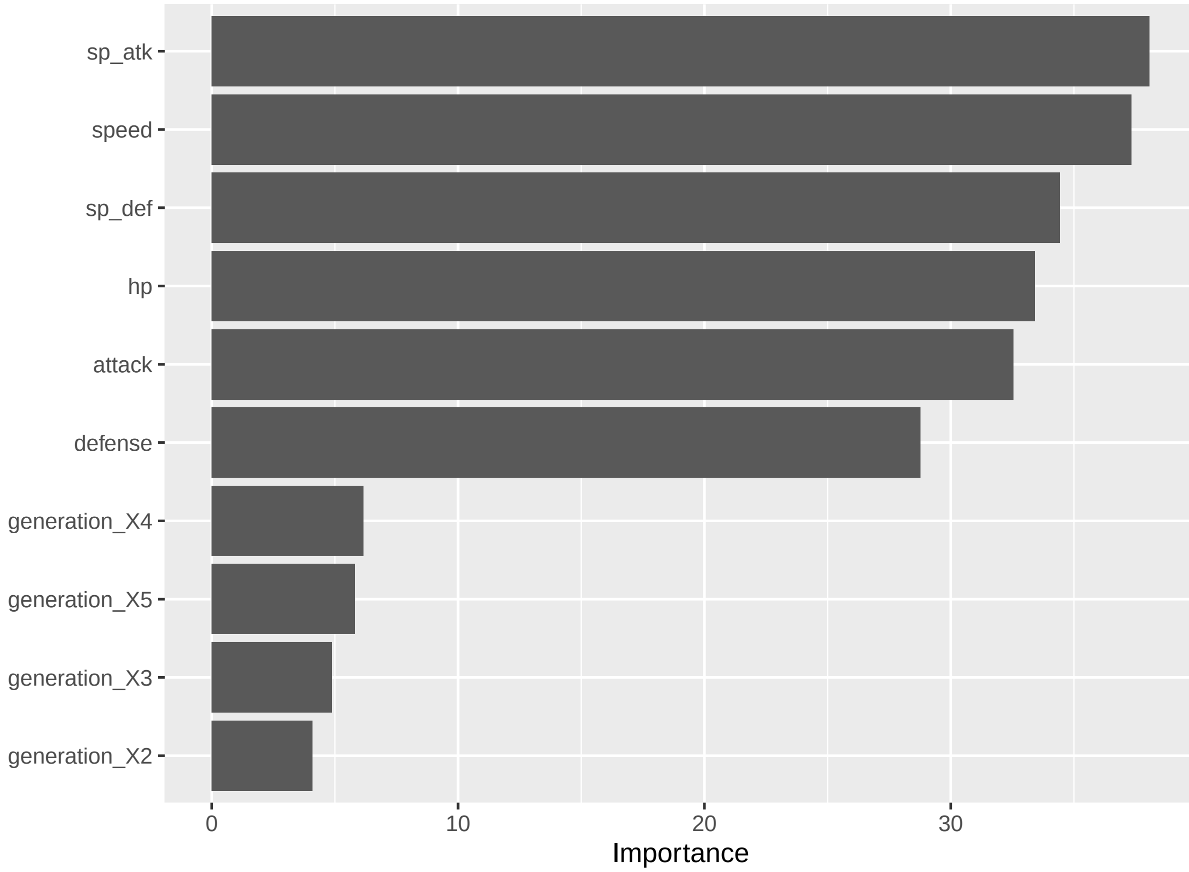
```
collect_metrics(rf_tune_res) %>%
  arrange(-mean)
```

```
## # 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     6     7     4 roc_auc hand_till  0.714     5 0.0215  Preprocessor1_Model~
##  2     4    10    10 roc_auc hand_till  0.713     5 0.0148  Preprocessor1_Model~
##  3     6    10     4 roc_auc hand_till  0.713     5 0.0127  Preprocessor1_Model~
##  4     5     7     4 roc_auc hand_till  0.711     5 0.00701 Preprocessor1_Model~
##  5     7    10     6 roc_auc hand_till  0.709     5 0.00458 Preprocessor1_Model~
##  6     8     8     8 roc_auc hand_till  0.708     5 0.0170  Preprocessor1_Model~
##  7     5     8     7 roc_auc hand_till  0.705     5 0.00908 Preprocessor1_Model~
##  8     5     7     2 roc_auc hand_till  0.705     5 0.0116  Preprocessor1_Model~
##  9     5    10     8 roc_auc hand_till  0.705     5 0.0177  Preprocessor1_Model~
## 10     8     8     7 roc_auc hand_till  0.704     5 0.0122  Preprocessor1_Model~
## # ... with 502 more rows
```

The roc_auc of our best-performing pruned decision tree on the folds is 0.7137 in model 238.

Question8

```r
rf_best=select_best(rf_tune_res)

rf_final=finalize_workflow(rf_wf,rf_best)

rf_final_fit=fit(rf_final,data=train)

rf_final_fit %>%
  extract_fit_engine() %>%
  vip()
```

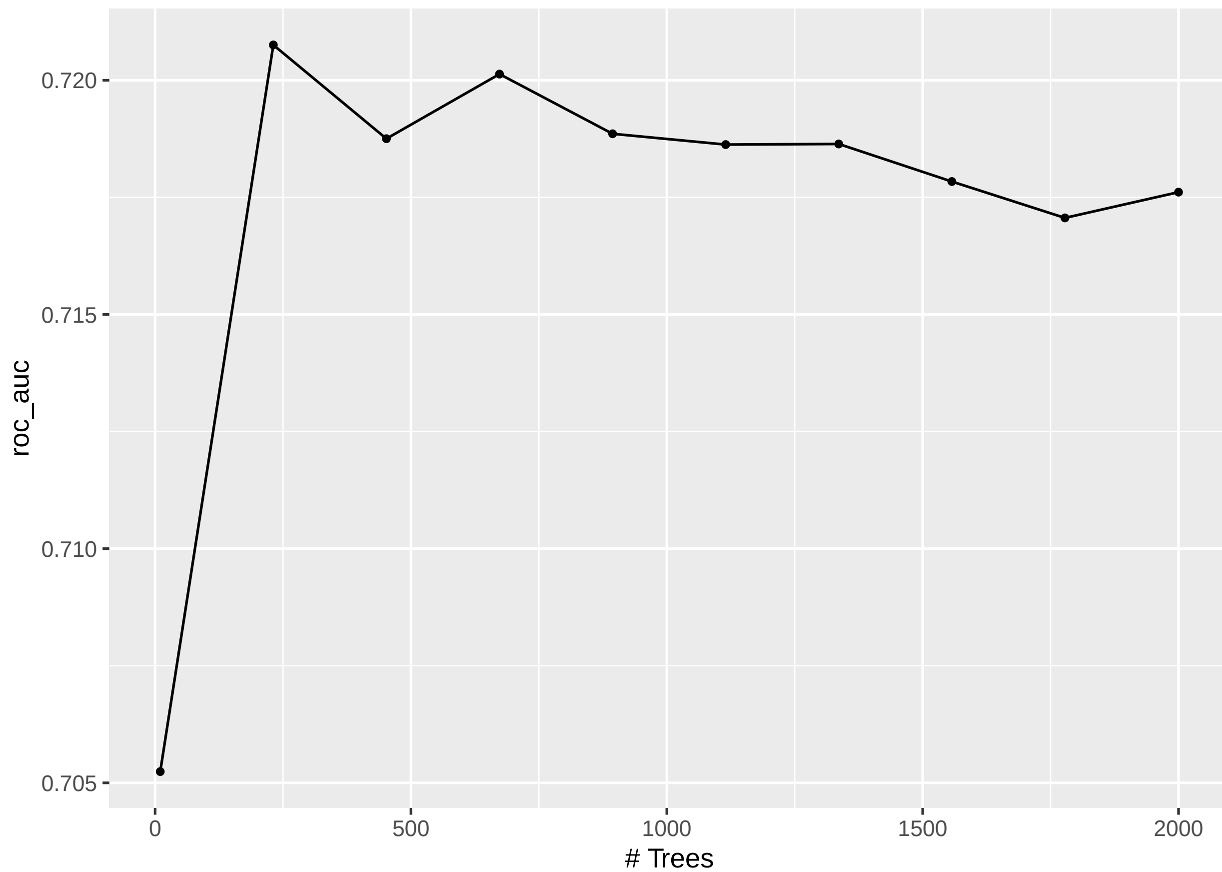


The most useful variables are attack, hp, then sp_atk. The least useful variables are generation_X5, generation_X4, generation_X3, and generation_X2. Although I know nothing about Pokemon, this makes sense to me.

Question 9

```r
boosted_spec=boost_tree(trees=tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

boosted_wf=workflow() %>%
  add_recipe(rcp) %>%
  add_model(boosted_spec)

grid_boosted=grid_regular(trees(range=c(10,2000)),levels = 10)
```

```r
tune_res_boosted=tune_grid(
  boosted_wf,
  resamples=folds,
  grid=grid_boosted,
  metrics=metric_set(roc_auc))

autoplot(tune_res_boosted)
```




```r
boost_best=select_best(tune_res_boosted)
```

We observe that there is a jump of roc_auc from the 0-250 tree range, after which we get slowly decreasing roc_auc.

```r
collect_metrics(tune_res_boosted) %>%
  arrange(-mean)
```

```
## # A tibble: 10 x 7
##    trees .metric .estimator  mean     n std_err .config
##    <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1   231 roc_auc hand_till  0.721     5 0.0105  Preprocessor1_Model02
##  2   673 roc_auc hand_till  0.720     5 0.00939 Preprocessor1_Model04
##  3   894 roc_auc hand_till  0.719     5 0.00971 Preprocessor1_Model05
##  4   452 roc_auc hand_till  0.719     5 0.00940 Preprocessor1_Model03
```

```
## 5  1336 roc_auc hand_till  0.719     5 0.00961 Preprocessor1_Model07
## 6  1115 roc_auc hand_till  0.719     5 0.00991 Preprocessor1_Model06
## 7  1557 roc_auc hand_till  0.718     5 0.00942 Preprocessor1_Model08
## 8  2000 roc_auc hand_till  0.718     5 0.00929 Preprocessor1_Model10
## 9  1778 roc_auc hand_till  0.717     5 0.00934 Preprocessor1_Model09
## 10   10 roc_auc hand_till  0.705     5 0.0137  Preprocessor1_Model01
```

The roc_auc of our best-performing pruned decision tree on the folds is 0.721 in model 02.

Question 10

```
class_tree_metrics=collect_metrics(class_tree_tune_res) %>%
  arrange(-mean)
rf_metrics=collect_metrics(rf_tune_res) %>%
  arrange(-mean)
boosted_metrics=collect_metrics(tune_res_boosted) %>%
  arrange(-mean)

best_metrics=bind_rows(class_tree_best,rf_best,boost_best)
best_metrics=best_metrics %>% add_column('model' = c("Pruned Decision Tree","Random Forest","Boosted Tre
best_metrics[,c("model",".config","cost_complexity","mtry","trees","min_n","roc_auc")]
```

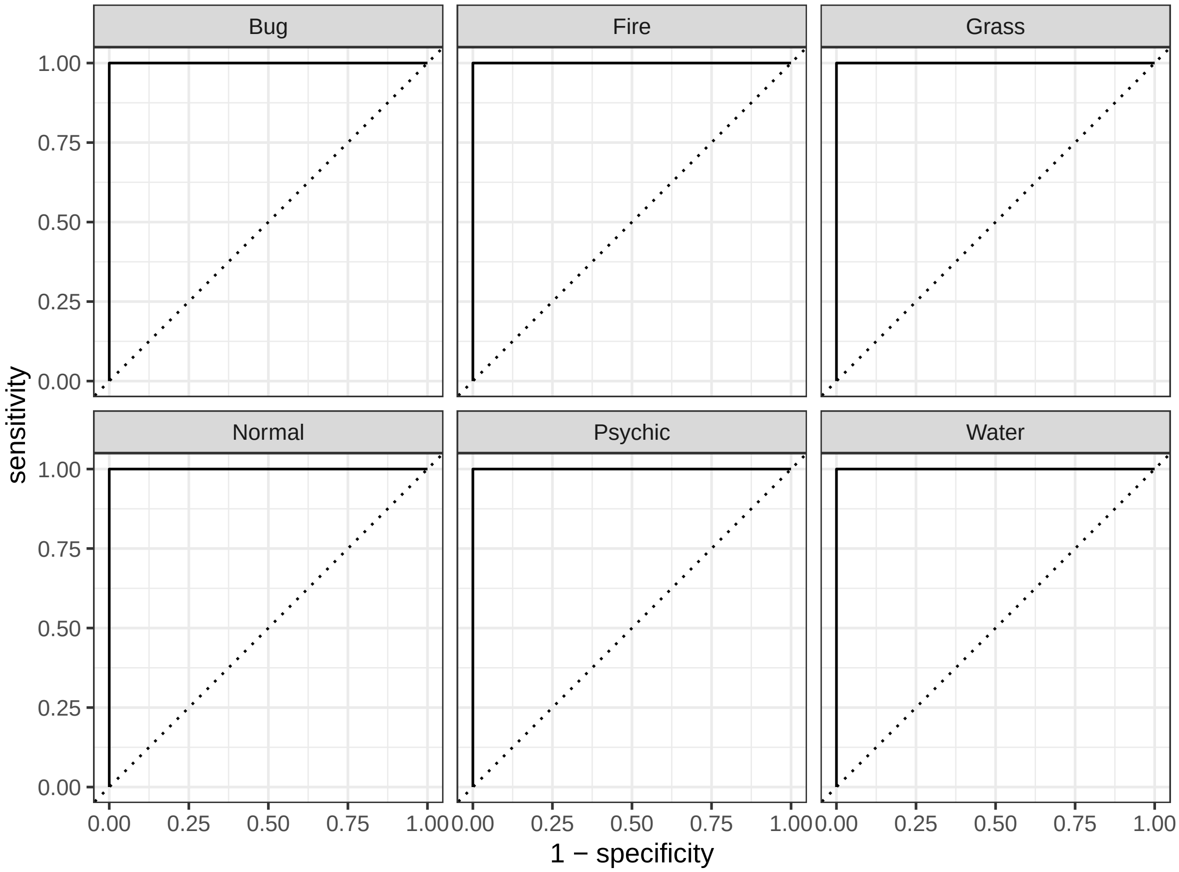
```
## # A tibble: 3 x 7
##   model                .config          cost_complexity  mtry trees min_n roc_auc
##   <chr>                <chr>                      <dbl> <int> <int> <int>   <dbl>
## 1 Pruned Decision Tree Preprocessor1_~           0.0129    NA    NA    NA   0.660
## 2 Random Forest        Preprocessor1_~          NA          6     7     4   0.714
## 3 Boosted Tree         Preprocessor1_~          NA         NA   231    NA   0.721
```

As we can see, the Boost Tree model with 231 trees performs best on the folds.

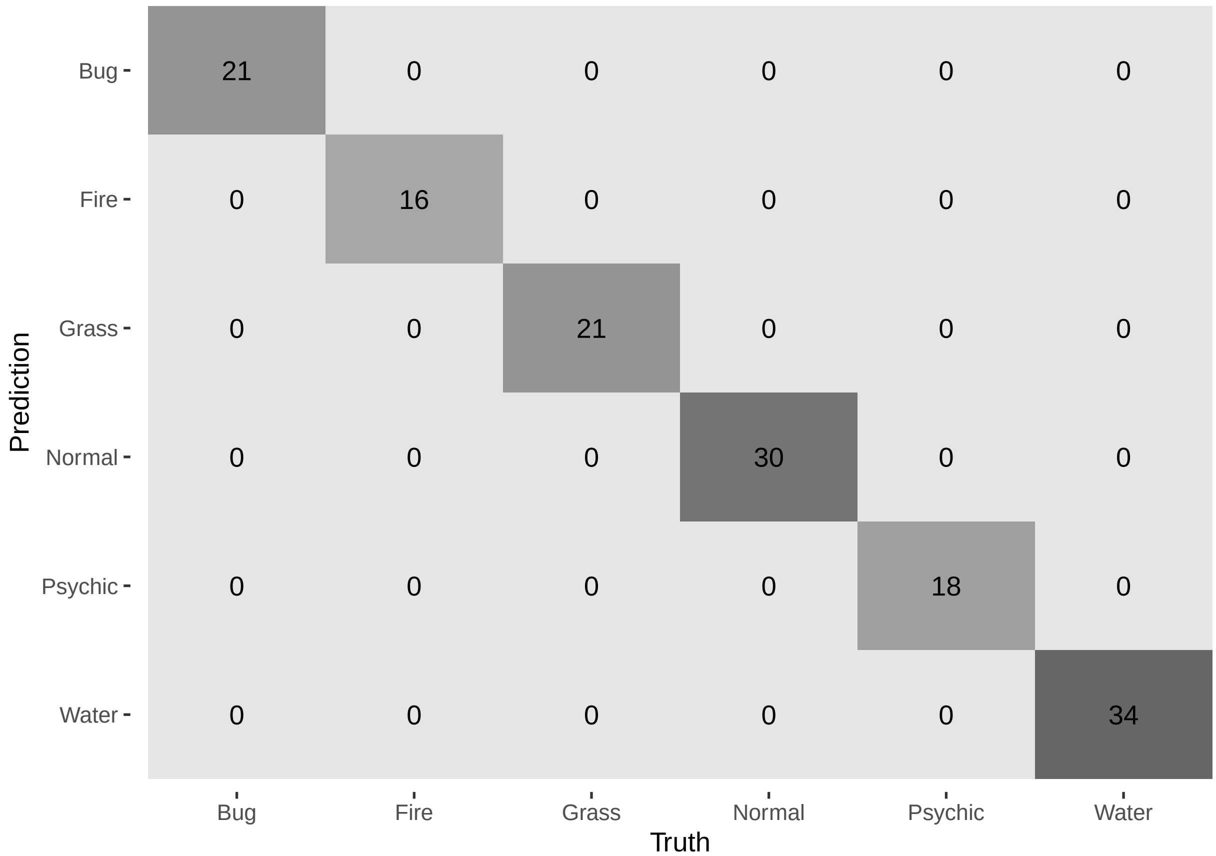
```
final=finalize_workflow(boosted_wf,boost_best)
final_fit=fit(final,data=test)

augment(final_fit,new_data=test) %>%
  roc_auc(truth=type_1,estimate=c('.pred_Bug','.pred_Fire','.pred_Grass','.pred_Normal','.pred_Psychic'
```

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

```
roc_curves=augment(final_fit,new_data=test) %>%
  roc_curve(truth=type_1,estimate=c('.pred_Bug','.pred_Fire', '.pred_Grass','.pred_Normal','.pred_Psych
autoplot(roc_curves)
```




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



Our best-performing model is extremely accurate. I tried other models, and they show my code has no problem. I think the only explanation might be the best model performs so well that it kind of overfit the data set, but from the test data set, our model predicts every classes accurately, and the auc roc is 1.