PSTAT131 HW5

Liangchen Xia

2022-05-14

Elastic Net Tuning

For this assignment, we will be working with the file `"pokemon.csv"`, found in `/data`. The file is from Kaggle: https://www.kaggle.com/abcsds/pokemon.

The Pokémon franchise encompasses video games, TV shows, movies, books, and a card game. This data set was drawn from the video game series and contains statistics about 721 Pokémon, or "pocket monsters." In Pokémon games, the user plays as a trainer who collects, trades, and battles Pokémon to (a) collect all the Pokémon and (b) become the champion Pokémon trainer.

Each Pokémon has a primary type (some even have secondary types). 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.

Figure 1: Fig 1. Vulpix, a Fire-type fox Pokémon from Generation 1.

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.

Read in the file and familiarize yourself with the variables using `pokemon_codebook.txt`.

```
# set seed
set.seed(10086)

library(ISLR)
library(tidyverse)
library(tidymodels)
library(ggplot2)
library(corrr)
library(dplyr)
```

```
library(discrim)
library(glmnet)
library(MASS)

pokemon_data <- read.csv('data/pokemon.csv')
#load the data
head(pokemon_data)
```

```
##   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
```

Exercise 1

Install and load the `janitor` package. Use its `clean_names()` function on the Pokémon data, and save the results to work with for the rest of the assignment. What happened to the data? Why do you think `clean_names()` is useful?

```
library(janitor)

pokemon_clean <- pokemon_data %>% clean_names()
head(pokemon_clean)
```

```
##   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
```

I use clean_names() to handle problematic variable names with special characters, spaces. It fix the repeat naming issues. As the. it could see that the column names are now all lowercase and void of special characters (contain only "_" character within variable names) and replaced with a more standard naming convention. That's good for what we do next.

Exercise 2

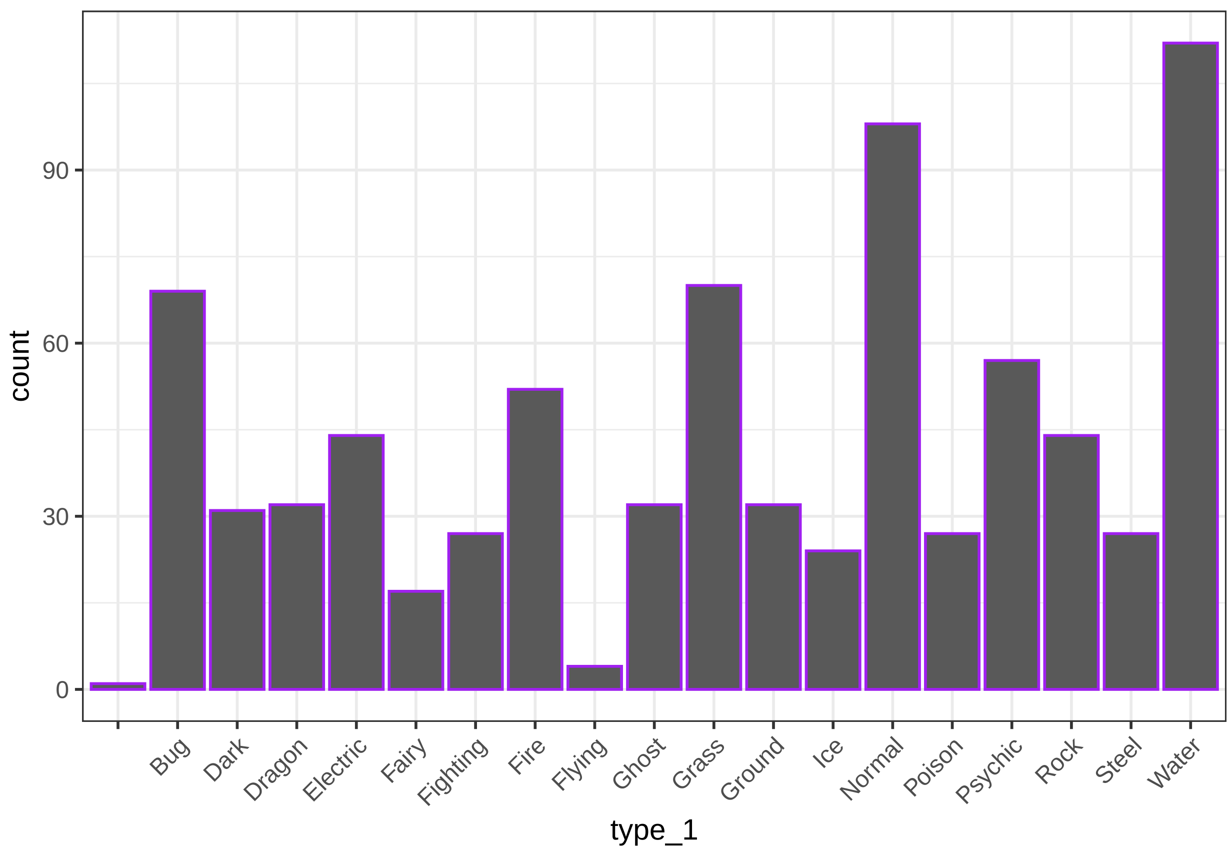
Using the entire data set, create a bar chart of the outcome variable, `type_1`.

How many classes of the outcome are there? Are there any Pokémon types with very few Pokémon? If so, which ones?

For this assignment, we'll handle the rarer classes by simply filtering them out. Filter the entire data set to contain only Pokémon whose `type_1` is Bug, Fire, Grass, Normal, Water, or Psychic.

After filtering, convert `type_1` and `legendary` to factors.

```
bar_1 <- ggplot(pokemon_clean, aes(x = type_1)) +
  geom_bar(color = "purple") +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1))
bar_1
```



As the graph, we could see there are 18 types, and the flying have very few Pokemon, and like normal or water have huge proportion of the data.

```
# we choose the data only pokemon whose type is Bug,
# so we have Fire, Grass, Normal, Water, Psychic
filtered_pokemon <- pokemon_clean %>% filter((type_1 == "Bug" | type_1 == "Fire" |
                             type_1 == "Grass" | type_1 == "Normal" |
                             type_1 == "Water" | type_1 == "Psychic"))
```

```r
# Converting type_1 and legendary to factors
filtered_pokemon$type_1 <- as.factor(filtered_pokemon$type_1)
filtered_pokemon$generation <- as.factor(filtered_pokemon$generation)
filtered_pokemon$legendary <- as.factor(filtered_pokemon$legendary)
head(filtered_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
```

Exercise 3

Perform an initial split of the data. Stratify by the outcome variable. You can choose a proportion to use. Verify that your training and test sets have the desired number of observations.

Next, use v-fold cross-validation on the training set. Use 5 folds. Stratify the folds by `type_1` as well. *Hint: Look for a* ***`strata`*** *argument.* Why might stratifying the folds be useful?

```r
# Stratified initial split
pokemon_split <- initial_split(filtered_pokemon,
                               prop = 0.7,
                               strata = type_1)

pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)
```

```r
# check number of observations
dim(pokemon_train)
```

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

```r
dim(pokemon_test)
```

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

```r
# v-fold
pokemon_folds <- vfold_cv(data = pokemon_train, v = 5, strata = "type_1")
pokemon_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
```

In the data, number of pokemons in each type are all different. So, stratifying the folds can make sure the distribution of types in each folds are approximately the same with the data set. I think Stratifying the folds might be useful in making sure that each fold has similar/equivalent proportions as the original data.

Exercise 4

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.

```
# get the recipe
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_
                         data= pokemon_train) %>%
                    step_dummy(legendary) %>%
                    step_dummy(generation) %>%
                    step_center(all_predictors()) %>%
                    step_scale(all_predictors())

pokemon_recipe
```

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

Exercise 5

We'll be fitting and tuning an elastic net, tuning `penalty` and `mixture` (use `multinom_reg` with the `glmnet` engine).

Set up this model and workflow. Create a regular grid for `penalty` and `mixture` with 10 levels each; `mixture` should range from 0 to 1. For this assignment, we'll let `penalty` range from -5 to 5 (it's log-scaled).

How many total models will you be fitting when you fit these models to your folded data?

```
# set up model w/ parameters to tune
pokemon_spec <- multinom_reg(penalty = tune(), mixture = tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")
pokemon_spec
```

```
## Multinomial Regression Model Specification (classification)
##
## Main Arguments:
##   penalty = tune()
##   mixture = tune()
##
## Computational engine: glmnet
```

```
# set up workflow with recipe and model
pokemon_workflow <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(pokemon_spec)
pokemon_workflow
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: multinom_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 4 Recipe Steps
##
## * step_dummy()
## * step_dummy()
## * step_center()
## * step_scale()
##
## -- Model -----------------------------------------------------------------------
## Multinomial Regression Model Specification (classification)
##
## Main Arguments:
##   penalty = tune()
##   mixture = tune()
##
## Computational engine: glmnet
```

```
# regular tuning grid
regular_grid <- grid_regular(penalty(range = c(-5, 5)),
                             mixture(range = c(0,1)),
                             levels = 10)
regular_grid
```

```
## # A tibble: 100 x 2
```

```
##          penalty mixture
##            <dbl>   <dbl>
##  1      0.00001        0
##  2      0.000129       0
##  3      0.00167        0
##  4      0.0215         0
##  5      0.278          0
##  6      3.59           0
##  7     46.4            0
##  8    599.             0
##  9   7743.             0
## 10 100000              0
## # ... with 90 more rows
```

Because we will tuning penalty and mixture with 10 levels each, and fit 100 models per fold. there are 5 folds, we will be fitting 500 models total.

Exercise 6

Fit the models to your folded data using `tune_grid()`.

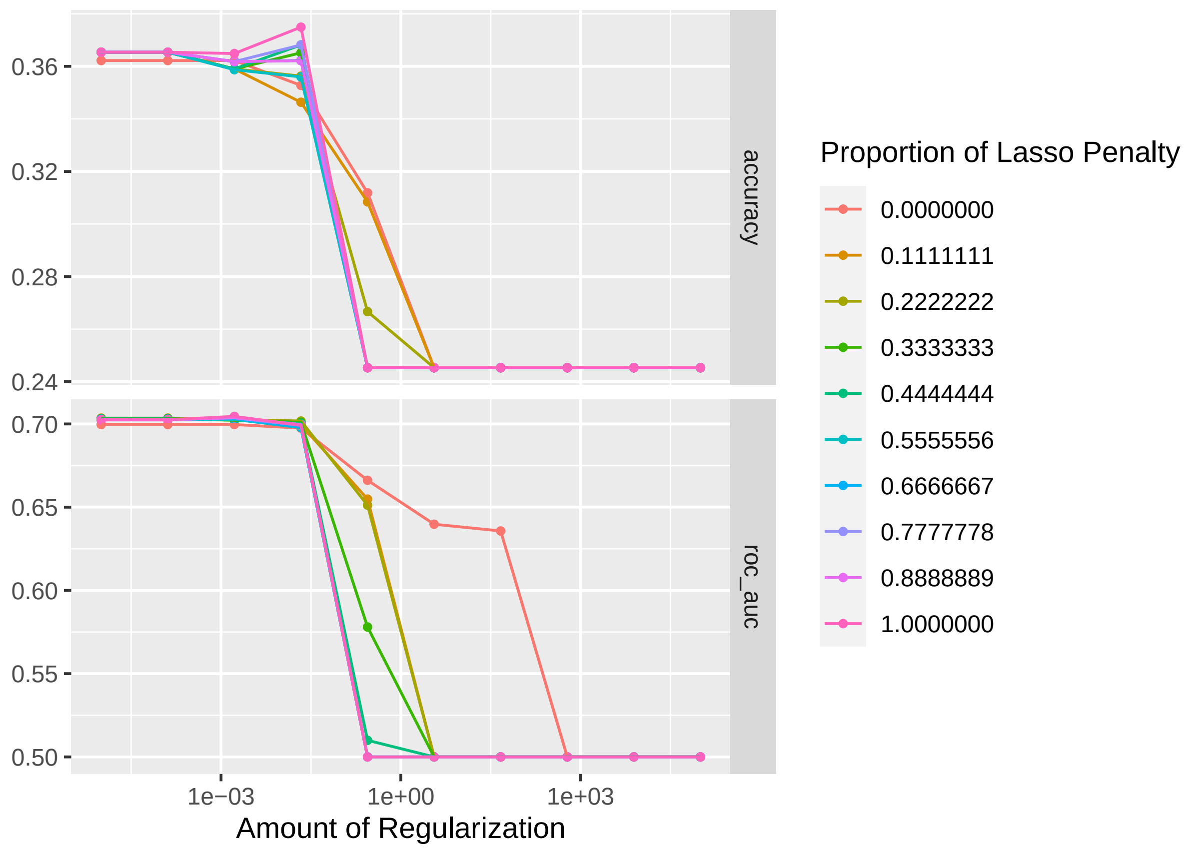
Use `autoplot()` on the results. What do you notice? Do larger or smaller values of `penalty` and `mixture` produce better accuracy and ROC AUC?

```
tune_res <- tune_grid(pokemon_workflow, resamples = pokemon_folds, grid = regular_grid)

tune_res
```

```
## # Tuning results
## # 5-fold cross-validation using stratification
## # A tibble: 5 x 4
##   splits           id    .metrics           .notes
##   <list>           <chr> <list>             <list>
## 1 <split [252/66]> Fold1 <tibble [200 x 6]> <tibble [0 x 3]>
## 2 <split [253/65]> Fold2 <tibble [200 x 6]> <tibble [0 x 3]>
## 3 <split [253/65]> Fold3 <tibble [200 x 6]> <tibble [0 x 3]>
## 4 <split [256/62]> Fold4 <tibble [200 x 6]> <tibble [0 x 3]>
## 5 <split [258/60]> Fold5 <tibble [200 x 6]> <tibble [0 x 3]>
```

```
autoplot(tune_res)
```



Look at the graph, we could see as the values of 'penalty' smaller, we got better accuracy and roc_auc generally. as the values of 'penalty' increases, the plots down sharply. That's mean smaller values of penalty and mixture produce better accuracy and ROC AUC.

Exercise 7

Use `select_best()` to choose the model that has the optimal `roc_auc`. Then use `finalize_workflow()`, `fit()`, and `augment()` to fit the model to the training set and evaluate its performance on the testing set.

```
#model selection
best_penalty <- select_best(tune_res, metric = "roc_auc")
best_penalty
```

```
## # A tibble: 1 x 3
##   penalty mixture .config
##     <dbl>   <dbl> <chr>
## 1 0.00167       1 Preprocessor1_Model093
```

```
# finalizing workflow
pokemon_final <- finalize_workflow(pokemon_workflow, best_penalty)
# fitting best model on the training set
pokemon_final_fit <- fit(pokemon_final, data = pokemon_train)


# evaluating best model on the test set
```

```r
final_model_acc <- augment(pokemon_final_fit, new_data = pokemon_test) %>%
  accuracy(truth = type_1, estimate = .pred_class)
final_model_acc
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.314
```

The accuracy of predicting the type standing at only around 0.314.

Exercise 8

Calculate the overall ROC AUC on the testing set.

Then create plots of the different ROC curves, one per level of the outcome. Also make a heat map of the confusion matrix.

What do you notice? How did your model do? Which Pokemon types is the model best at predicting, and which is it worst at? Do you have any ideas why this might be?

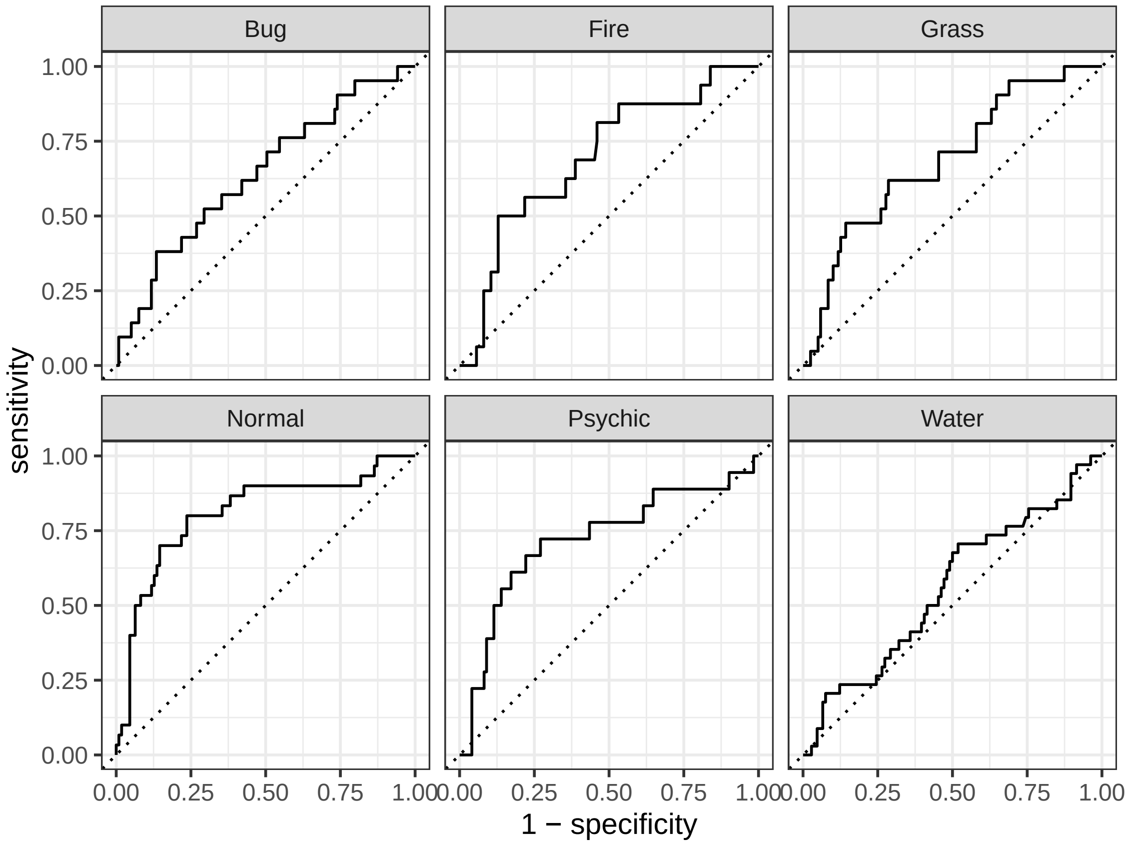
```r
#overall ROC AUC
total_roc_auc <- augment(pokemon_final_fit, new_data = pokemon_test) %>%
  roc_auc(truth = type_1, estimate =
            c(.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic, .pred_Water))

total_roc_auc
```

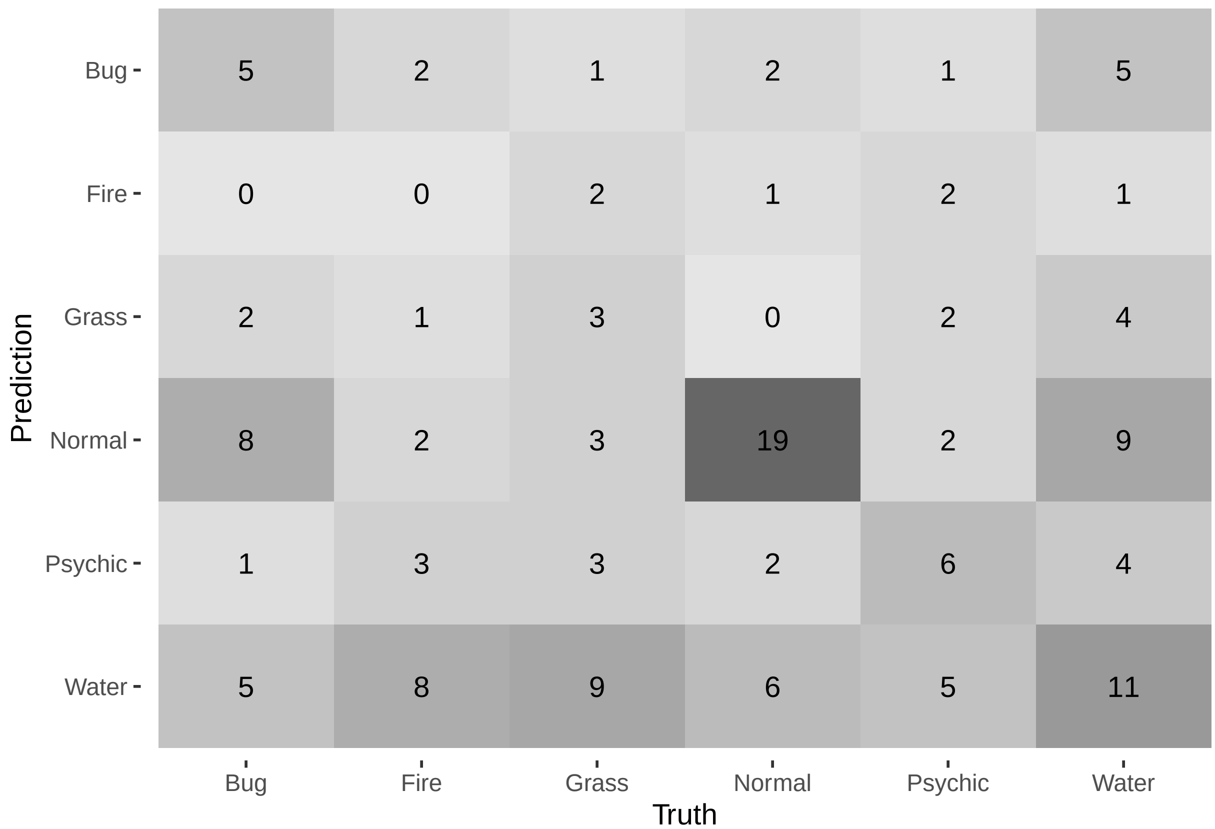
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.682
```

```r
roc_curves <- augment(pokemon_final_fit, new_data = pokemon_test) %>%
  roc_curve(truth = type_1, estimate =
              c(.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic, .pred_Water)) %>%
  autoplot()

roc_curves
```




```
final_model_conf <- augment(pokemon_final_fit, new_data = pokemon_test) %>%
  conf_mat(truth = type_1, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
final_model_conf
```



For all, I think my model not doing very well. Because the low accuracy. The model's prediction accuracy are different among all six types. The Psychic and Normal type is the model best at predicting. But the fire type is not good at predicting. I think that's because there are less fire types in general, that's the reason the accuracy to suffer.