# HW5 Xilong Li

Xilong Li (3467966)

2022-05-15

```
library(tidymodels)
```

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

## v broom        0.7.12     v recipes      0.2.0
## v dials        0.1.0      v rsample      0.1.1
## v dplyr        1.0.8      v tibble       3.1.6
## v ggplot2      3.3.5      v tidyr        1.2.0
## v infer        1.0.0      v tune         0.2.0
## v modeldata    0.1.1      v workflows    0.2.6
## v parsnip      0.2.1      v workflowsets 0.2.1
## v purrr        0.3.4      v yardstick    0.0.9

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

```
library(tidyverse)
```

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

## v readr   2.1.1     v forcats 0.5.1
## v stringr 1.4.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x readr::col_factor() masks scales::col_factor()
## x purrr::discard()    masks scales::discard()
## x dplyr::filter()     masks stats::filter()
## x stringr::fixed()    masks recipes::fixed()
## x dplyr::lag()        masks stats::lag()
## x readr::spec()       masks yardstick::spec()
```

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

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

```
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(janitor)
```

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

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

```
library(discrim)
```

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

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

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

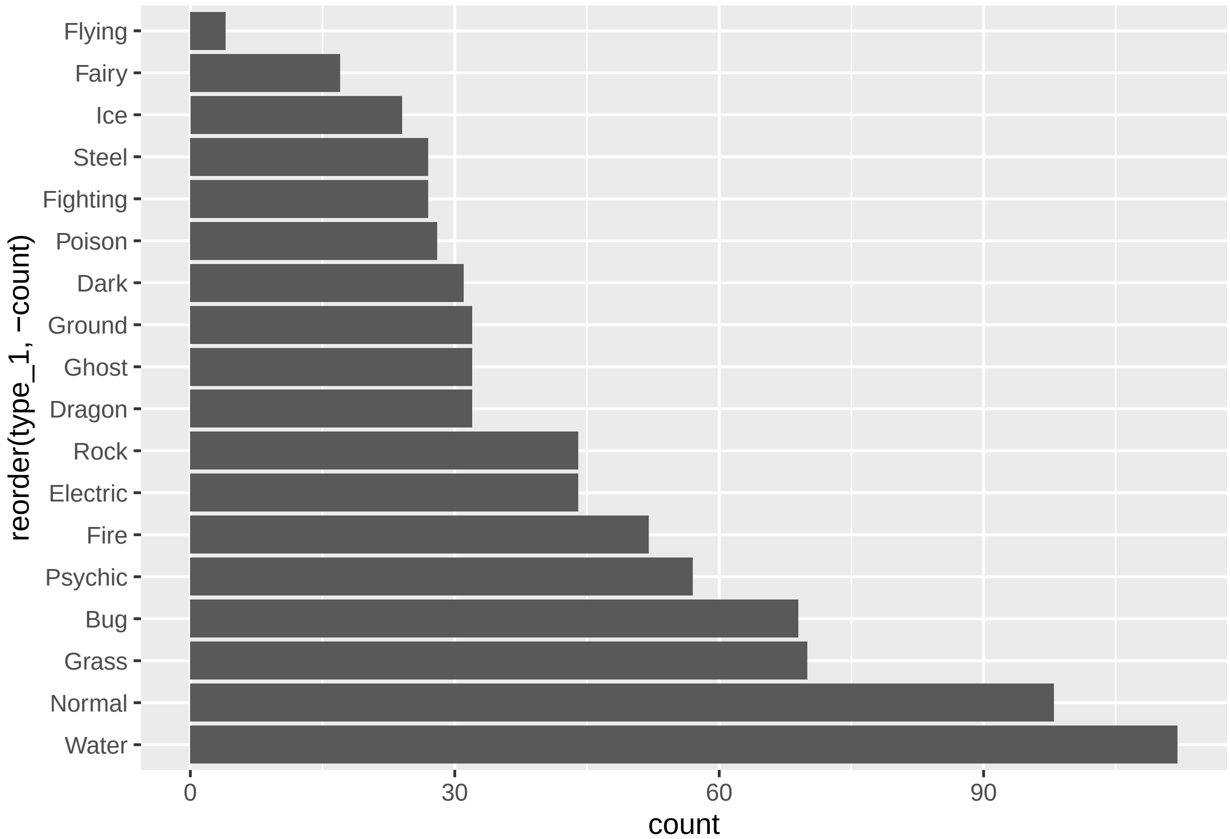
### Question 1:

```
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
```

By using the "clean_names" function, the resulting names are unique and consist only of the '_' character, numbers, and letters. Capitalization preferences can be specified using the case parameter. Accented characters are transliterated to ASCII. For example, an "o" with a German umlaut over it becomes "o", and the Spanish character "enye" becomes "n".(This explanation is cited from the website: https://rdrr.io/cran/janitor/man/clean_names.html)
## Question 2:

```
copy_pokemon <- pokemon
ordered_data <- copy_pokemon %>%
  group_by(type_1) %>%
  summarise(count = n()) %>%
  arrange(count)

ggplot(ordered_data, aes(x = count, y = reorder(type_1, -count))) + geom_bar(stat = "identity")
```




As it is shown above, there are 18 classes in total, and classes such as flying, fairy, and ice have fewer pokemons than others,

```
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
```

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

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

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

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

### Question 3:

```
set.seed(2200)

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"
```

By stratifying the folds, we can make sure that the folds are representative of the data, since the splited data is also stratified on type_1. So that the distribution of types in each folds are approximately the same.

## Question 4:

```
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())

class(poke_train$generation)
```

```
## [1] "integer"
```

## Question 5:

```
poke_spec <- multinom_reg (penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet")

poke_workflow <- workflow() %>%
  add_recipe(poke_recipe) %>%
  add_model(poke_spec)

poke_grid <- grid_regular(penalty(range = c(-5, 5)),
                             mixture(range = c(0,1)),
                             levels = c(10,10))
```

Thus, there will be 500 models in total, since there are ten levels each for penalty and mixture and 5 folds in the data.

### Question 6:

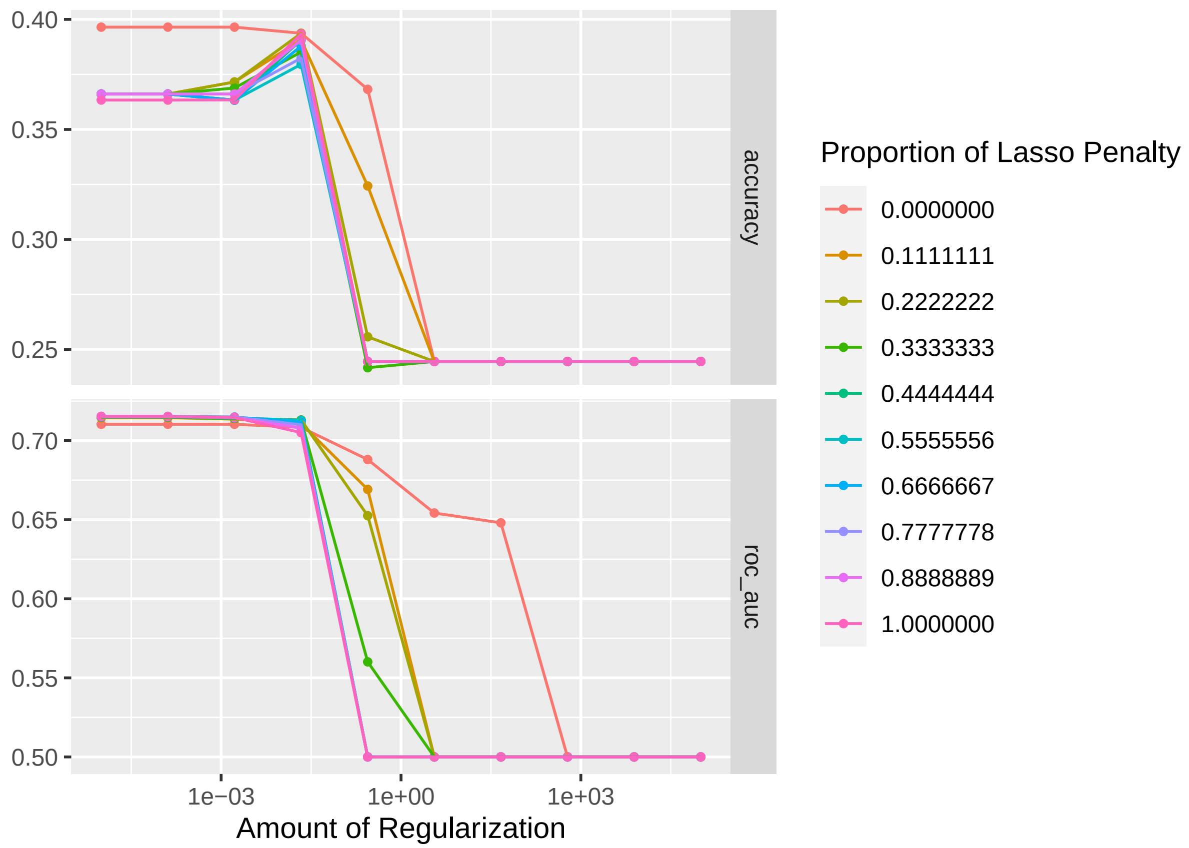
```
poke_workflow
```

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

```
tune_res <- tune_grid(
  poke_workflow,
  resamples = poke_folds,
  grid = poke_grid
)
```

```
## ! 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(tune_res)
```




I noticed that when the mixture and penalty gets too large, the accuracy and ROC_AUC actually get smaller and smaller, and thus smaller mixture and penalty is better.

### Question 7:

```
penalty_chosen <- select_best(tune_res, metric = "roc_auc")
penalty_chosen
```

```
## # A tibble: 1 x 3
##   penalty mixture .config
##     <dbl>   <dbl> <chr>
## 1 0.00001   0.667 Preprocessor1_Model061
```

```
poke_final <- finalize_workflow(poke_workflow, penalty_chosen)
poke_final_fit <- fit(poke_final, data = poke_train)
```

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

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

augment(poke_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     0.383
```

### Question 8:

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

head(predicted_result)
```

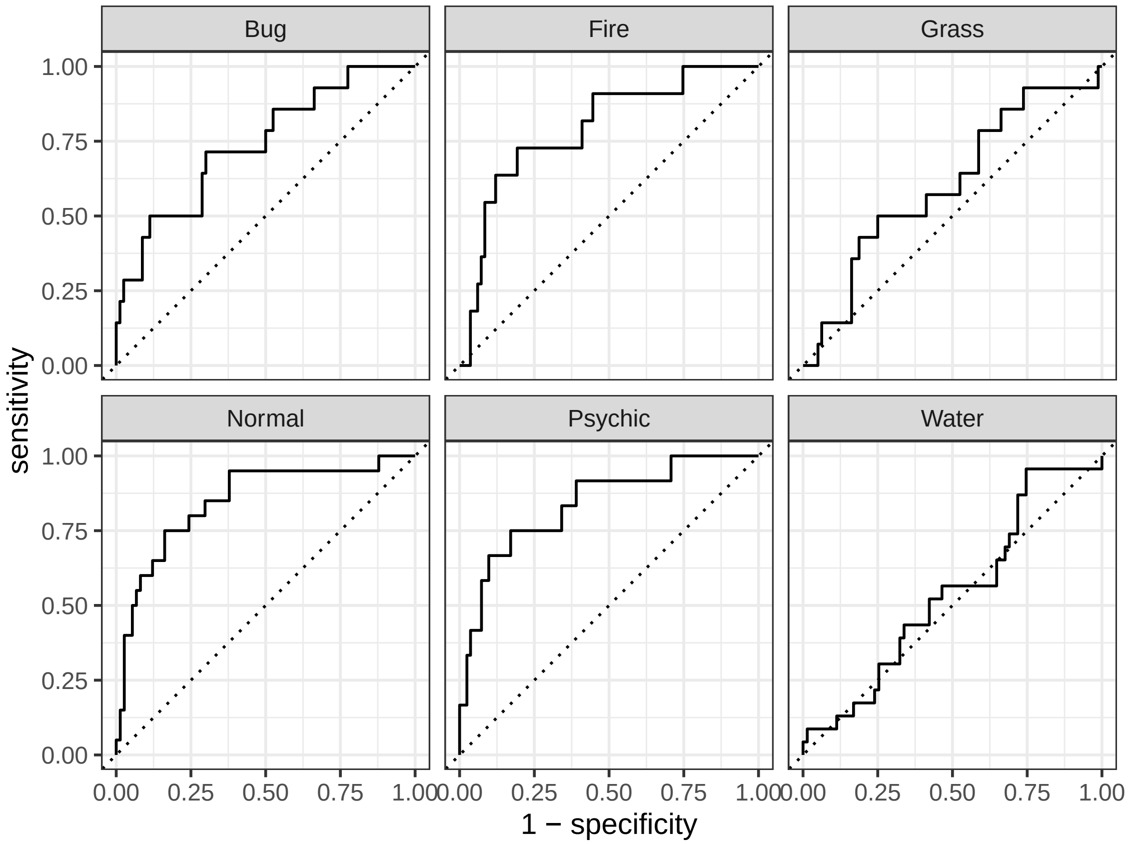
```
## # A tibble: 6 x 8
##   type_1 .pred_class .pred_Bug .pred_Fire .pred_Grass .pred_Normal .pred_Psychic
##   <fct>  <fct>           <dbl>      <dbl>       <dbl>        <dbl>         <dbl>
## 1 Grass  Water          0.0652     0.152       0.203        0.0189        0.0505
## 2 Water  Water          0.287      0.0688      0.167        0.125         0.0564
## 3 Water  Water          0.124      0.0912      0.124        0.107         0.0412
## 4 Fire   Bug            0.245      0.0699      0.115        0.217         0.126
## 5 Bug    Bug            0.358      0.0856      0.215        0.106         0.0255
## 6 Normal Normal         0.113      0.0716      0.0567       0.433         0.0475
## # ... with 1 more variable: .pred_Water <dbl>
```

Ploting the ROC_AUC curve:

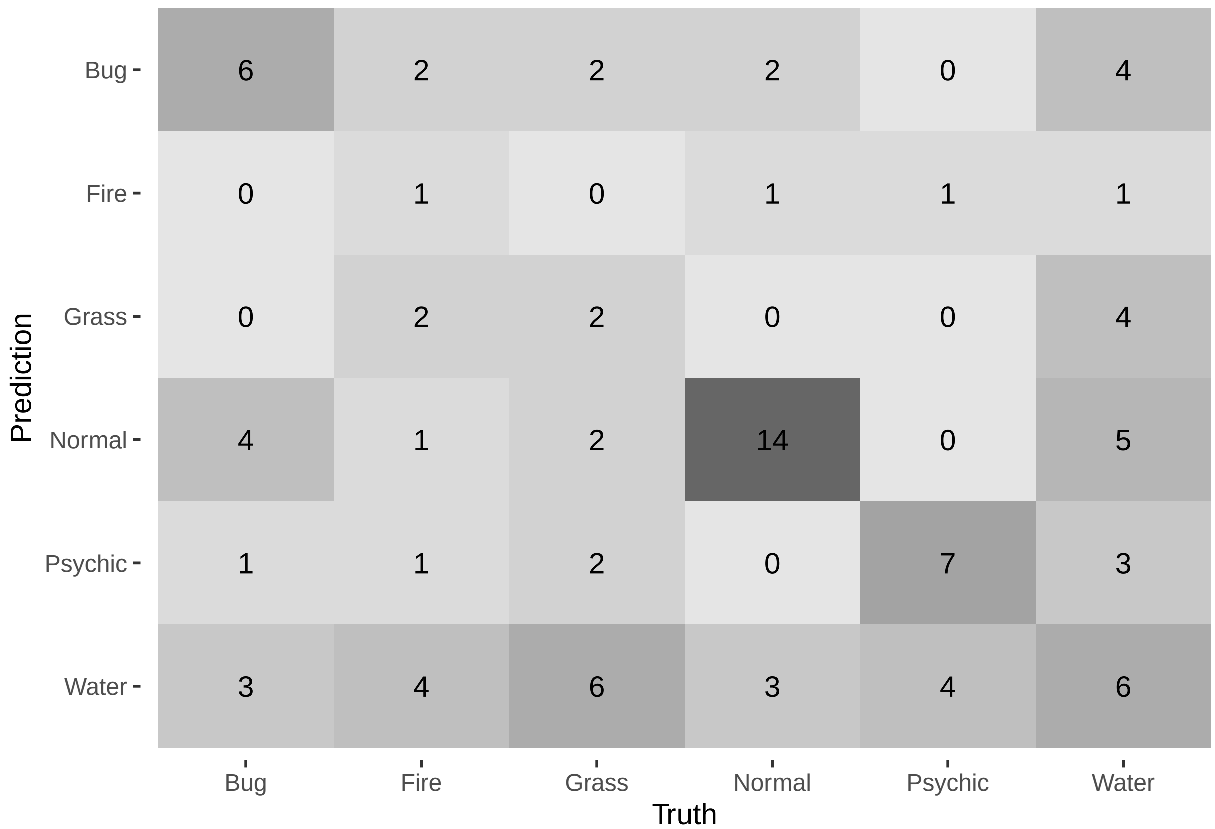
```r
# ?roc_auc
roc_auc(predicted_result, type_1, c(.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic, .p
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.727
```

```r
# ?roc_curve
roc_curve(predicted_result, type_1, c(.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic,.p
```




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




As it is shown in the graph above, the number on the diagnal means the score that the model predict the type_1 correctly.
However, the class of Fire, water, and Grass are not predicted well by this model, and perhaps water performs the worst since the model mistakenly predicted the class for many times. While the class of Normal is best predicted, as most of its predictions are correct. It shows that this model is not performing well enough to predict pokemons' types, but it is understandable since there are too many types to be predicted in this model while there are limited data to train it. It is also interesting, perhaps irrelevant, to notice that Grass, Fire, and Water are three most basic types of Pokemons. So the reason why they are poorly predicted might be these three types share many basic and common features LOL.

The model dose not have a great performance on Pokemon type prediction with 0.66 roc_auc. Also, the overall accuracy is 0.24468. The psychic is the model best at predicting. The water is the model worst at predicting on. The reason might be size of each type is vary, for example water has 112 observations and fire has 52 observations.