homework5

Yingshan Li (7937790)

May 11, 2022

Load the data

```
pokemon <- read.csv(file = "Pokemon.csv" )
```

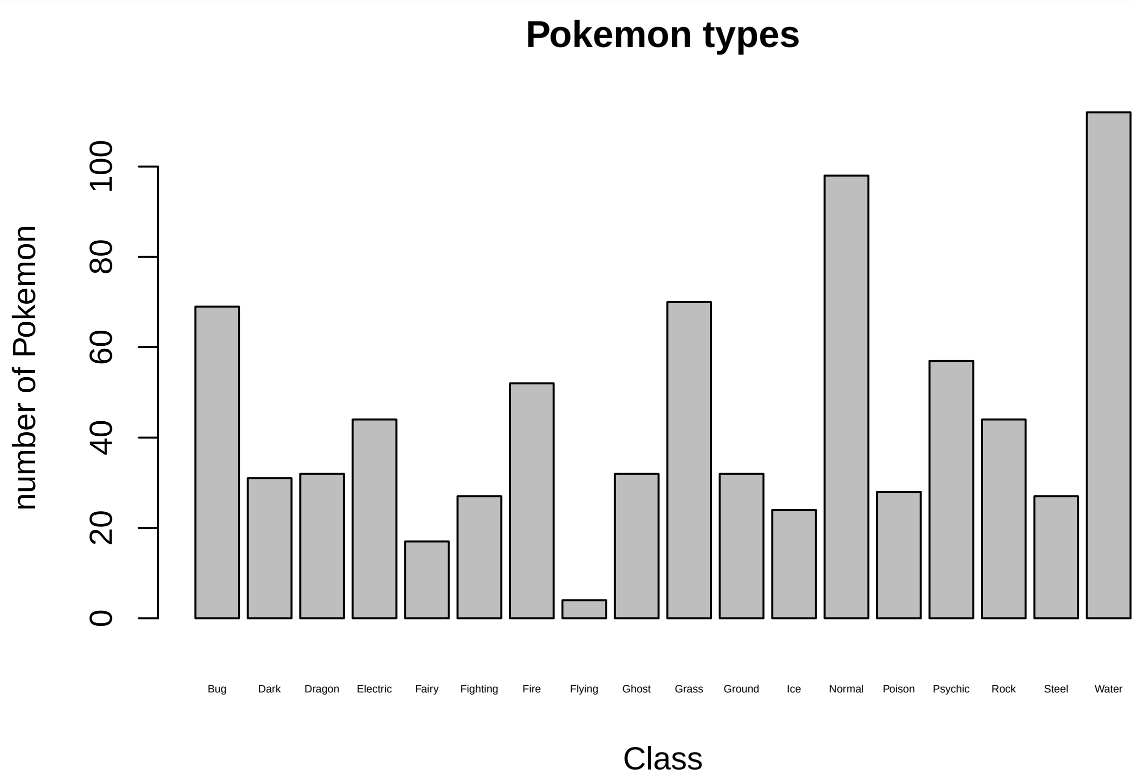
Exercise 1

```
pokemon1 <- pokemon %>%
  clean_names()
```

We can see all the variable name change to the lower-case letter and contain only "_" character within variable names to separate words. This function is useful because it can convert all the variable names to snake case, which is recommended style for tidyverse. Therefore, it is more convenient for us to use in the later process.

Exercise 2

```
counts <- table(pokemon1$type_1)
barplot(counts, xlab = "Class", ylab = "number of Pokemon", main = "Pokemon types", cex.names = .3)
```




```
pokemon1 %>%
  group_by(type_1) %>%
```

```
  summarise(count = n())
```

```
## # A tibble: 18 x 2
##    type_1   count
##    <chr>    <int>
##  1 Bug         69
##  2 Dark        31
##  3 Dragon      32
##  4 Electric    44
##  5 Fairy       17
##  6 Fighting    27
##  7 Fire        52
##  8 Flying       4
##  9 Ghost       32
## 10 Grass       70
## 11 Ground      32
## 12 Ice         24
## 13 Normal      98
## 14 Poison      28
## 15 Psychic     57
## 16 Rock        44
## 17 Steel       27
## 18 Water      112
```

There are total 18 classes of type_1. The Flying Pokemon type have very few Pokemon since there are only 4 pokemons are Flying type. The Fairy Pokemon type is also contain fewer pokemons than others. There are 17 Fairy Pokemon, but all the other types except Flying have more than 20 pokemons.

```
filtered_pokemon <- pokemon1 %>%
  filter(type_1 == "Bug" | type_1 == "Fire" | type_1 == "Grass" | type_1 == "Normal" | type_1 == "Water
filtered_pokemon %>%
  group_by(type_1) %>%
  summarise(count = n())
```

```
## # A tibble: 6 x 2
##   type_1  count
##   <chr>   <int>
## 1 Bug        69
## 2 Fire       52
## 3 Grass      70
## 4 Normal     98
## 5 Psychic    57
## 6 Water     112
```

```
pokemon2 <- filtered_pokemon %>%
  mutate(type_1 = factor(type_1)) %>%
  mutate(legendary = factor(legendary)) %>%
  mutate(generation = factor(generation))
```

Exercise 3 split the data

```
set.seed(3435)
pokemon_split <- initial_split(pokemon2, strata = type_1, prop = 0.7)

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

verify correct number of observations in each data set

```
dim(pokemon_train)
```

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

```
318/458
```

```
## [1] 0.6943231
```

```
dim(pokemon_test)
```

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

```
140/458
```

```
## [1] 0.3056769
```

```
pokemon_folds <- vfold_cv(pokemon_train, strata = type_1, v = 5)
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
```

The number of pokemons in each type are all different in our data set. Thus, stratifying the folds can make sure the distribution of types in each folds are approximately the same with the data set. Each fold would be a good representative of the data set, and avoid the problem of class imbalance between folds that might affect future results.

Exercise 4

Create Recipe

```
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense +
                           hp + sp_def, data = pokemon_train) %>%
  step_dummy(legendary) %>%
  step_dummy(generation) %>%
  step_normalize(all_predictors())
```

Exercise 5

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

```
pokemon_wkflow <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(pokemon_sepc)
```

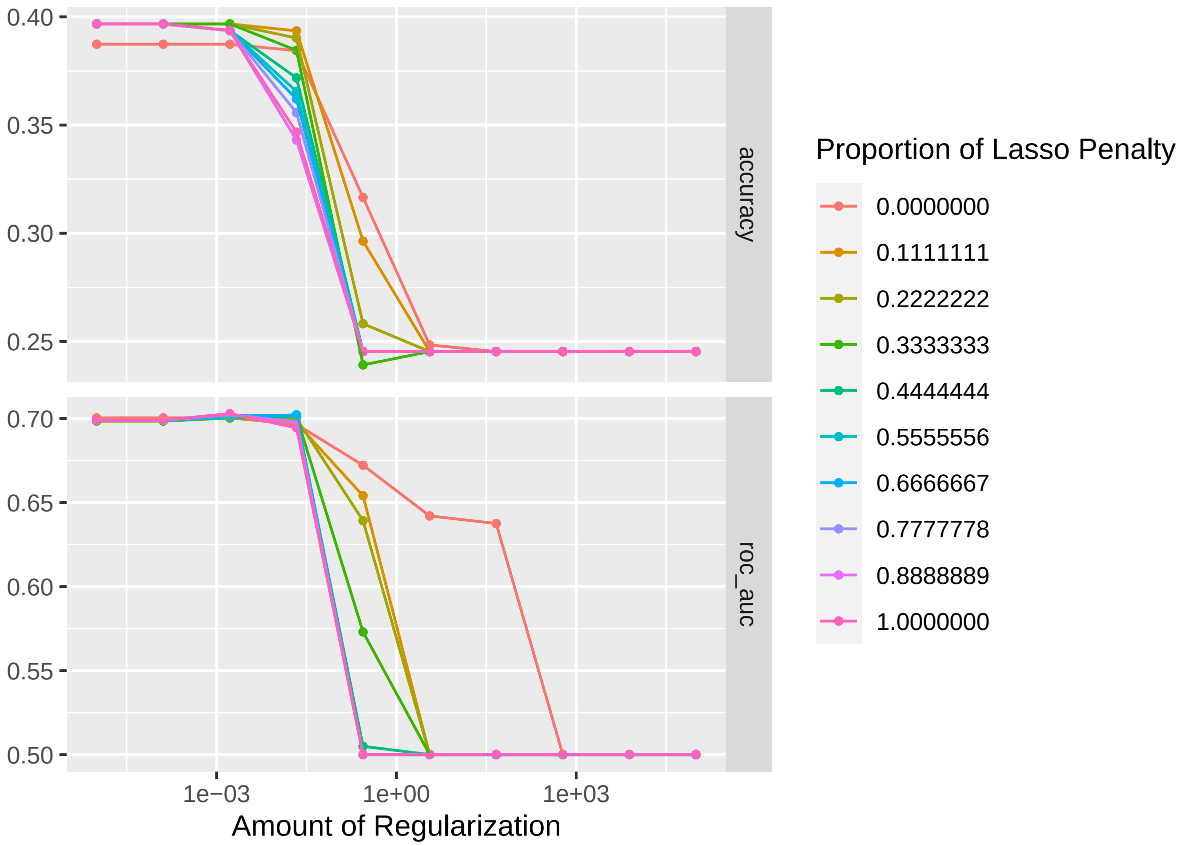
```
pm_grid <- grid_regular(penalty(range = c(-5, 5)), mixture(range = c(0 , 1)), levels = c(10, 10))
```

I will fit a total of $10 \times 10 \times 5 = 500$ models.

Exercise 6

```r
tune_res <- tune_grid(
  object = pokemon_wkflow,
  resamples = pokemon_folds,
  grid = pm_grid
)
```

```r
autoplot(tune_res)
```



As the penalty and mixture get larger, the accuracy and roc_auc decreases, which means smaller values of penalty and mixture produce better accuracy and ROC AUC.

Exercise 7

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

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

```r
pokemon_final <- finalize_workflow(pokemon_wkflow, best_pm)

pokemon_final_fit <- fit(pokemon_final, data = pokemon_train)
```

```r
prediction_result <- augment(pokemon_final_fit, new_data = pokemon_test) %>%
    select(type_1, .pred_class, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic, .pred_

accuracy(prediction_result, type_1, .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.343
```

The accuracy of the model on the testing data is approximately 34.3%.

ROC AUC on the testing set

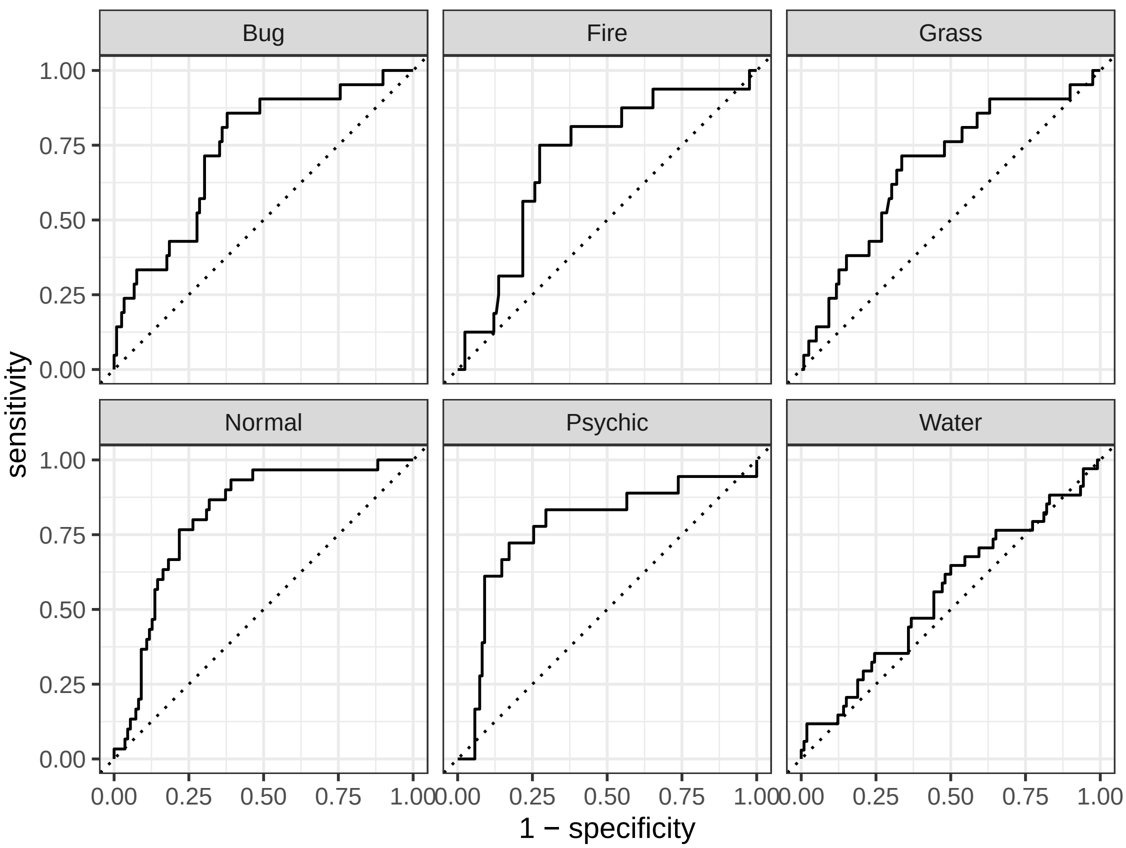
```
roc_aunp(prediction_result, type_1, .pred_Bug:.pred_Water)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 roc_aunp macro          0.701
```

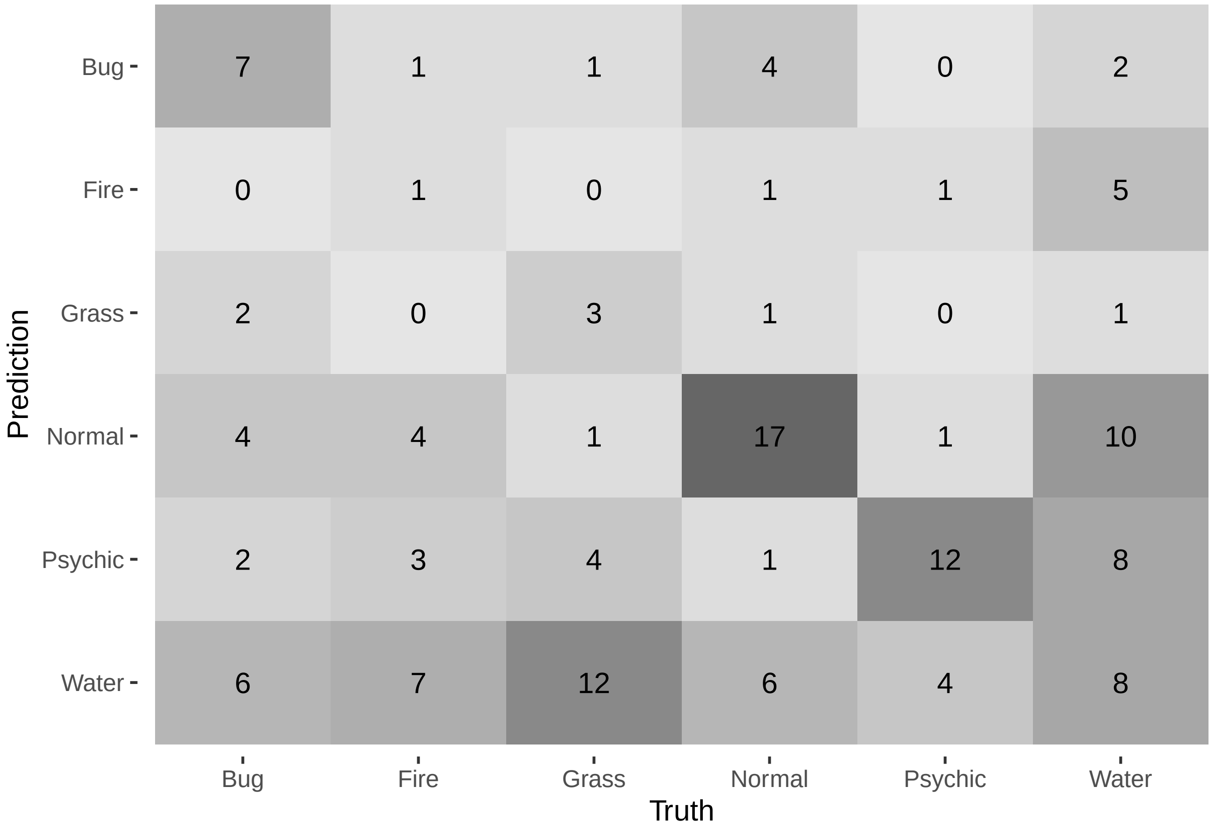
```
roc_auc(prediction_result, type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic, .pr
```

```
## # A tibble: 1 x 3
##   .metric .estimator     .estimate
##   <chr>   <chr>              <dbl>
## 1 roc_auc macro_weighted     0.701
```

```
prediction_result %>%
  roc_curve(type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic, .pred_Water) %>%
  autoplot()
```



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



The model generally not doing very well since the accuracy and roc_auc are not high. However, considering that this is a multiclass case, the model's performances is actually reasonable and acceptable. I have noticed that the model's prediction accuracy are different among all six types. The Psychic Pokemon type is the model best at predicting, and the model also perform fairly well for Normal type. On the other hand, the Fire Pokemon type is the model worst at predicting. I think this might be due to the imbalanced dataset. Some types such as Normal and Water have more observations than other types.