# Homework 5

Yixin Wang

## Contents



### Elastic Net Tuning

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`.

```
library(tidymodels)
library(tidyverse)
library(discrim)
library(poissonreg)
library(ISLR)
library(ISLR2)
library(corrr)
library(klaR)
library(glmnet)
tidymodels_prefer()
set.seed(3435)
```

#### 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 <- read_csv(file = "Pokemon.csv")
pokemon <- clean_names(pokemon)
pokemon <- as_tibble(pokemon)
pokemon
```

```
## # A tibble: 800 x 13
##    number name      type_1 type_2 total    hp attack defense sp_atk sp_def speed
##     <dbl> <chr>     <chr>  <chr>  <dbl> <dbl>  <dbl>   <dbl>  <dbl>  <dbl> <dbl>
##  1      1 Bulbasaur Grass  Poison   318    45     49      49     65     65    45
##  2      2 Ivysaur   Grass  Poison   405    60     62      63     80     80    60
```

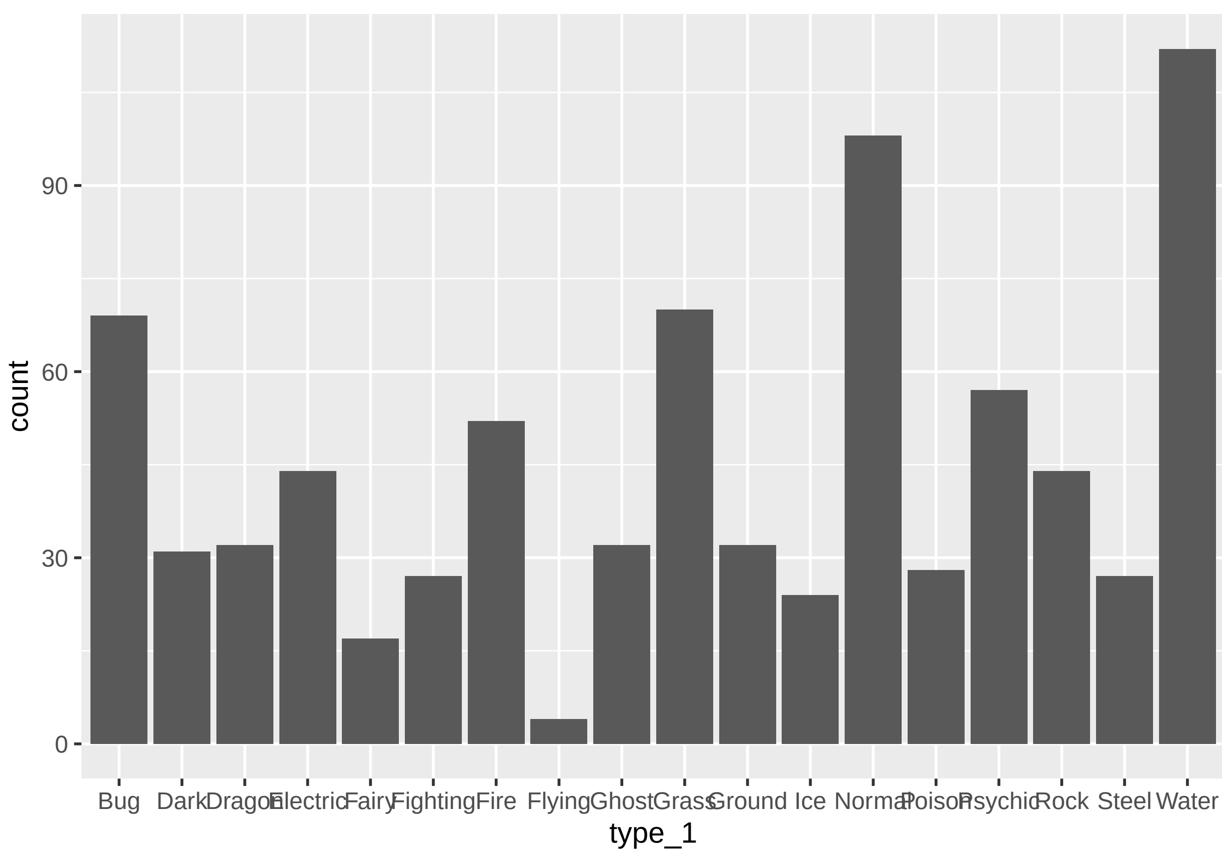
```
##  3      3 Venusaur  Grass  Poison   525    80     82      83     100     100    80
##  4      3 Venusaur~ Grass  Poison   625    80    100     123     122     120    80
##  5      4 Charmand~ Fire   <NA>     309    39     52      43      60      50    65
##  6      5 Charmele~ Fire   <NA>     405    58     64      58      80      65    80
##  7      6 Charizard Fire   Flying   534    78     84      78     109      85   100
##  8      6 Charizar~ Fire   Dragon   634    78    130     111     130      85   100
##  9      6 Charizar~ Fire   Flying   634    78    104      78     159     115   100
## 10      7 Squirtle  Water  <NA>     314    44     48      65      50      64    43
## # ... with 790 more rows, and 2 more variables: generation <dbl>,
## #   legendary <lgl>
```

The data handles problematic variable names. The clean_names() function cleans the variable names and return to data frame.

**Exercise 2**

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

```
library(ggplot2)
ggplot(pokemon, aes(x = type_1)) + geom_bar()
```




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

There are 18 classes of the outcome. The flying and fairy types are have few Pokemons.

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.

```
pokemon <- pokemon[pokemon$type_1 %in% c("Bug","Fire","Grass","Normal","Water","Psychic"), ]
```

```
pokemon$type_1 <- as.factor(pokemon$type_1)
pokemon$legendary <- as.factor(pokemon$legendary)
```

### 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?

```
pokemon_split <- initial_split(pokemon, prop = 0.7, strata = type_1)
pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)
```

```
dim(pokemon_train)
```

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

```
dim(pokemon_test)
```

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

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

Since the outcome variable is imbalanced, there are less observations for flying and fairy than other types.

### 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.

```
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_
  step_dummy(all_nominal_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_center(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).

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

pokemon_grid <- grid_regular(penalty(c(-5, 5)), mixture(c(0,1)), levels = 10)

pokemon_workflow <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(pokemon_spec)
```

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

There are 10 models will be fitted to 5 folded data.

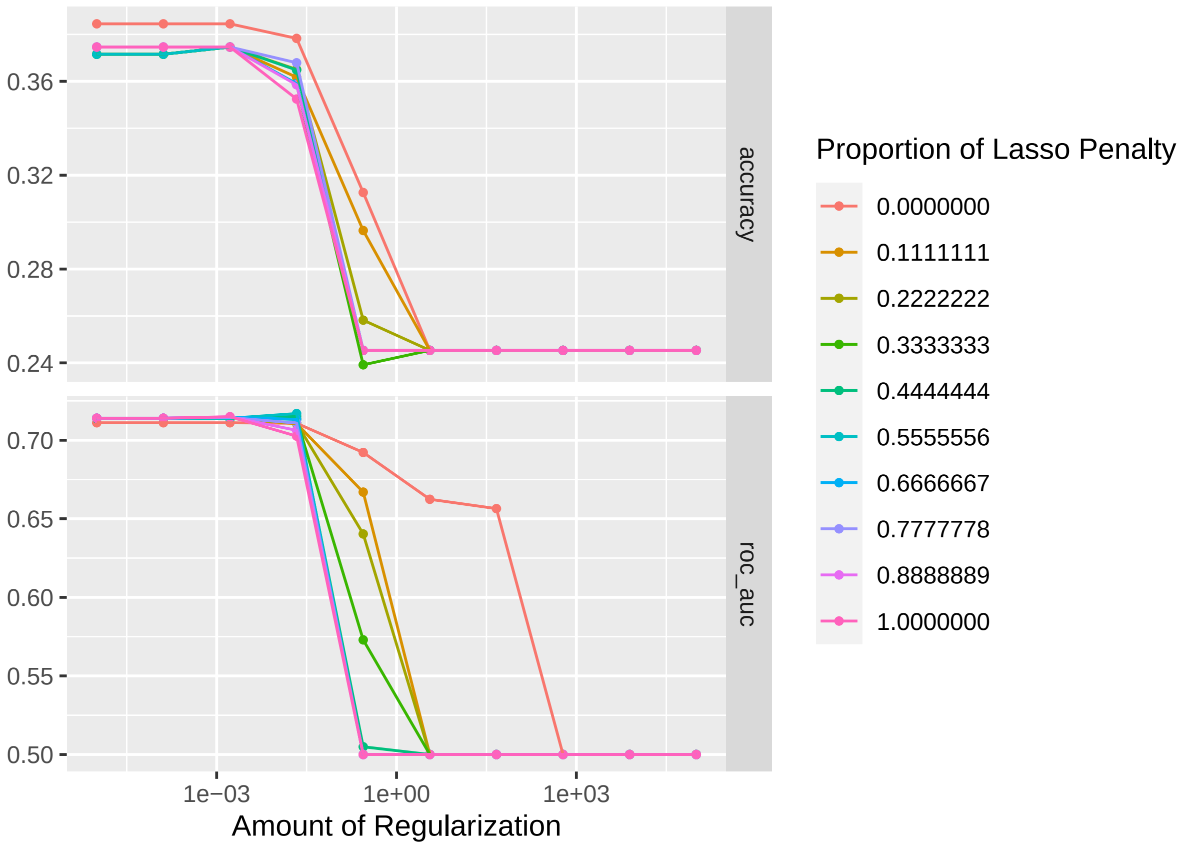
**Exercise 6**

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

```
tune_res <- tune_grid(pokemon_workflow, resamples = pokemon_fold, grid = pokemon_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?

```
autoplot(tune_res)
```




The smaller the penalty lead to the better accuracy and ROC AUC. Similarly, the smaller mixture leads to the 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.

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

```
final = pokemon_workflow %>%
  finalize_workflow(pokemon_penalty) %>%
  fit(pokemon_train) %>%
  augment(pokemon_test)

accuracy(final, truth = type_1, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.364
```

**Exercise 8**

Calculate the overall ROC AUC on the testing set.

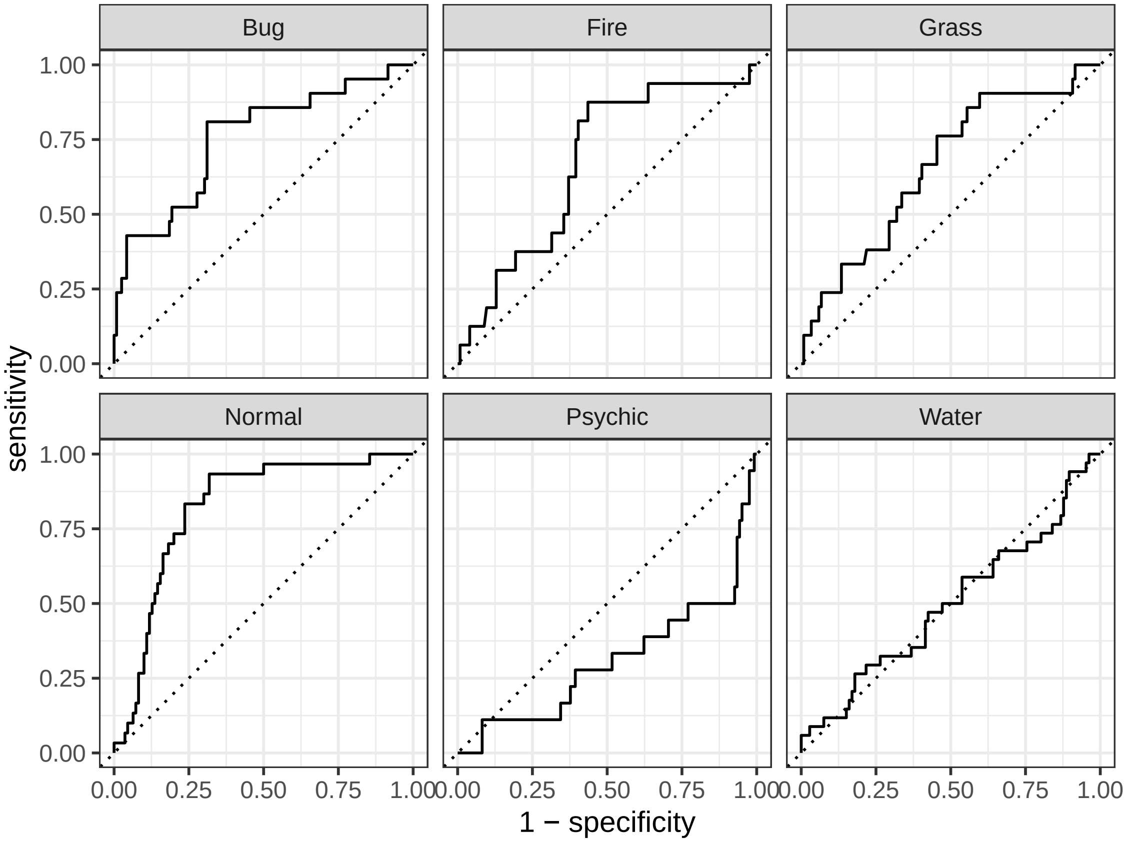
```r
roc_auc(final, truth = type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Water, .pred_Psy
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.615
```

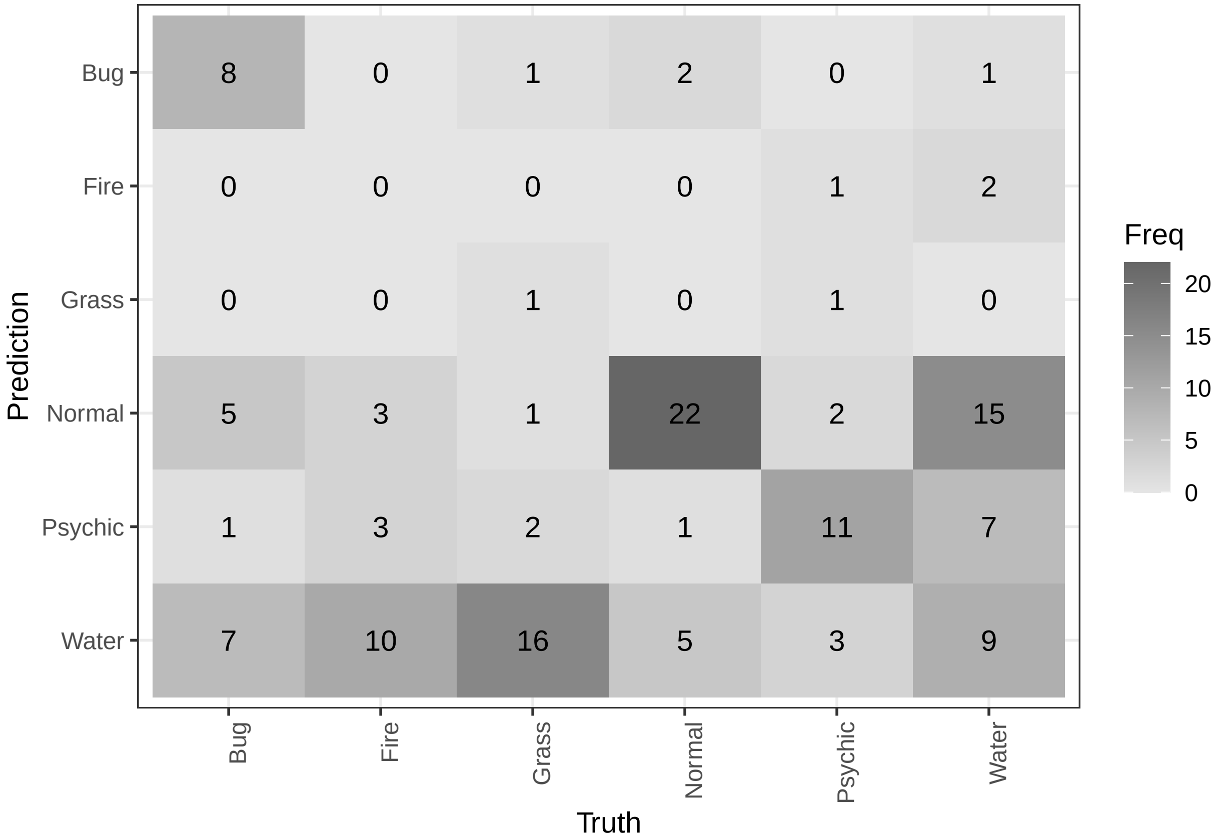
Then create plots of the different ROC curves, one per level of the outcome. Also make a heat map of the confusion matrix.

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
autoplot(roc_curve(final, truth = type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Water
```




```r
conf_mat(final, truth = type_1, estimate = .pred_class) %>%
  autoplot(type = "heatmap") + theme_bw() + theme(axis.text.x = element_text(angle = 90, hjust = 1))
```




The roc_auc is approximately 0.6072. The accuracy is approximately 0.45. I notice that the model is not good at predicting the type of Pokemon. However, the normal type is the model best at predicting. The psychic type is the worst type to predict in the graph. I think the reason is probably that the psychic is quite different than the other Pokemons.