PSTAT 131 Homework 4

Matthew Zhang

Spring 2022

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`.

```
library(tidyverse)
library(tidymodels)
library(corrr)
library(discrim)
library(ggplot2)

library(glmnet)
library(janitor)

pokemon <- read.csv('data/pokemon.csv')
head(pokemon)
```

```
##   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?

```
cn_pokemon <- pokemon %>% clean_names()
head(cn_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
```

Observing the data, it is evident that the variable names are now all in the same format which produces consistency within the data set. More specifically, the variable names have been converted to lower case and variables with multiple words are now seperated by an underscore "_" rather than a space. This is useful because it makes dealing with the data a lot more fluid and concise in which variables are all recognizable.

Exercise 2

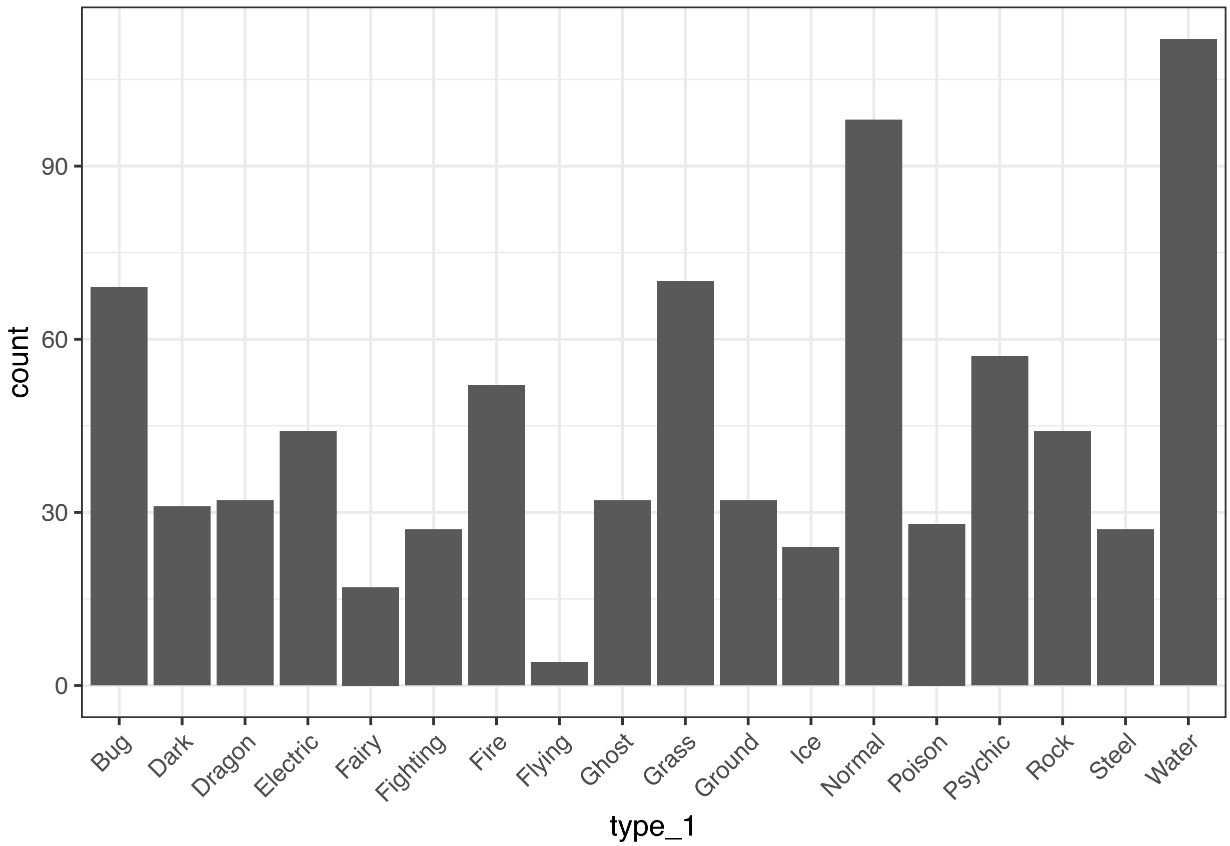
Using the entire data set, create a bar chart of the outcome variable, `type_1`.

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.

```
cn_pokemon %>% ggplot(aes(x = type_1)) +
  geom_bar() +
  theme_bw() + theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1))
```



There are 18 classes of the outcome; the 'Flying' type seems to have much less Pokemon compared to the others.

```r
cn_pokemon %>%
  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
```

```r
pokemon_types <- cn_pokemon %>%
   filter(type_1 == "Bug" | type_1 == "Fire" | type_1 == "Grass" | type_1 == "Normal" | type_1 == "Wate

pokemon_types %>%
  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
```

```r
pokemon_factor <- pokemon_types %>%
  mutate(type_1 = factor(type_1)) %>%
  mutate(legendary = factor(legendary)) %>%
  mutate(generation = factor(generation))
```

Diving into the distribution of Pokemon types, 'Flying' only has 4 Pokemon while 'Water' has 112 Pokemon.

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
set.seed(123)

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

dim(pokemon_train)
```

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

```r
dim(pokemon_test)
```

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

```r
pokemon_folds <- vfold_cv(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
```

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(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?

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

pokemon_wf <- workflow() %>%
                add_recipe(pokemon_recipe) %>%
                add_model(pokemon_elastic)

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

I will be fitting a total of 500 models as a result of 5 folds in k-folds cross validation and 100 models per fold such that 5*100=500.

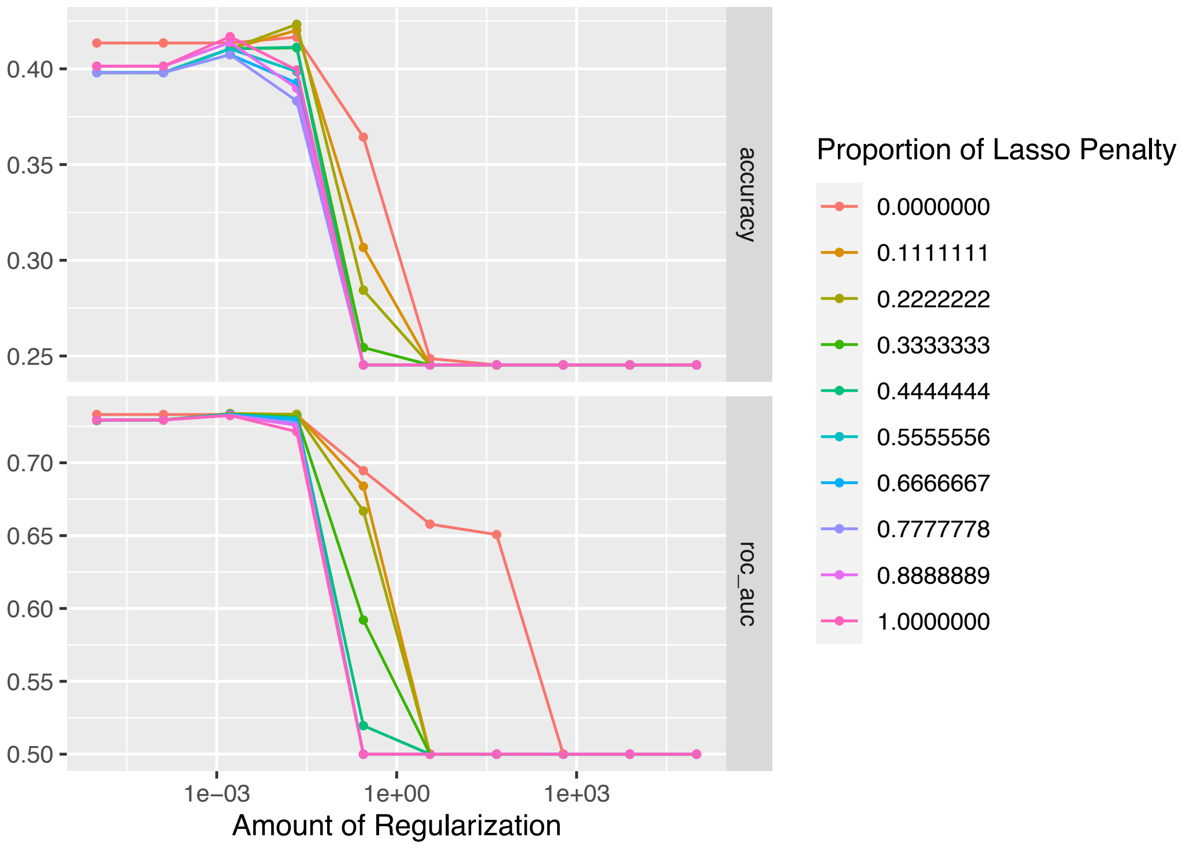
Exercise 6

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(
  object= pokemon_wf,
  resamples = pokemon_folds,
  grid = regular_grid
)

autoplot(tune_res)
```



Observing the plots, it seems that smaller values of 'penalty' and 'mixture' end up producing higher accuracy and roc_auc as the plots tend to decrease dramatically. For accuracy, the lines seem much more closer together as opposed to roc_auc where the lines are hardly more dispersed.

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.

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

pokemon_final <- finalize_workflow(pokemon_wf, best_roc)
pokemon_fit <- fit(pokemon_final, data = pokemon_train)

pokemon_result <- metric_set(accuracy, mcc, f_meas)
pokemon_result(augment(pokemon_fit, new_data = pokemon_test), truth = type_1, estimate = .pred_class)
```

```
## # A tibble: 3 x 3
```

```
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.307
## 2 mcc      multiclass     0.148
## 3 f_meas   macro          0.274
```

According to the results, the model did not perform to a great extent, with a=n accuracy of 30.7%.

Exercise 8

Calculate the overall ROC AUC on the testing set.

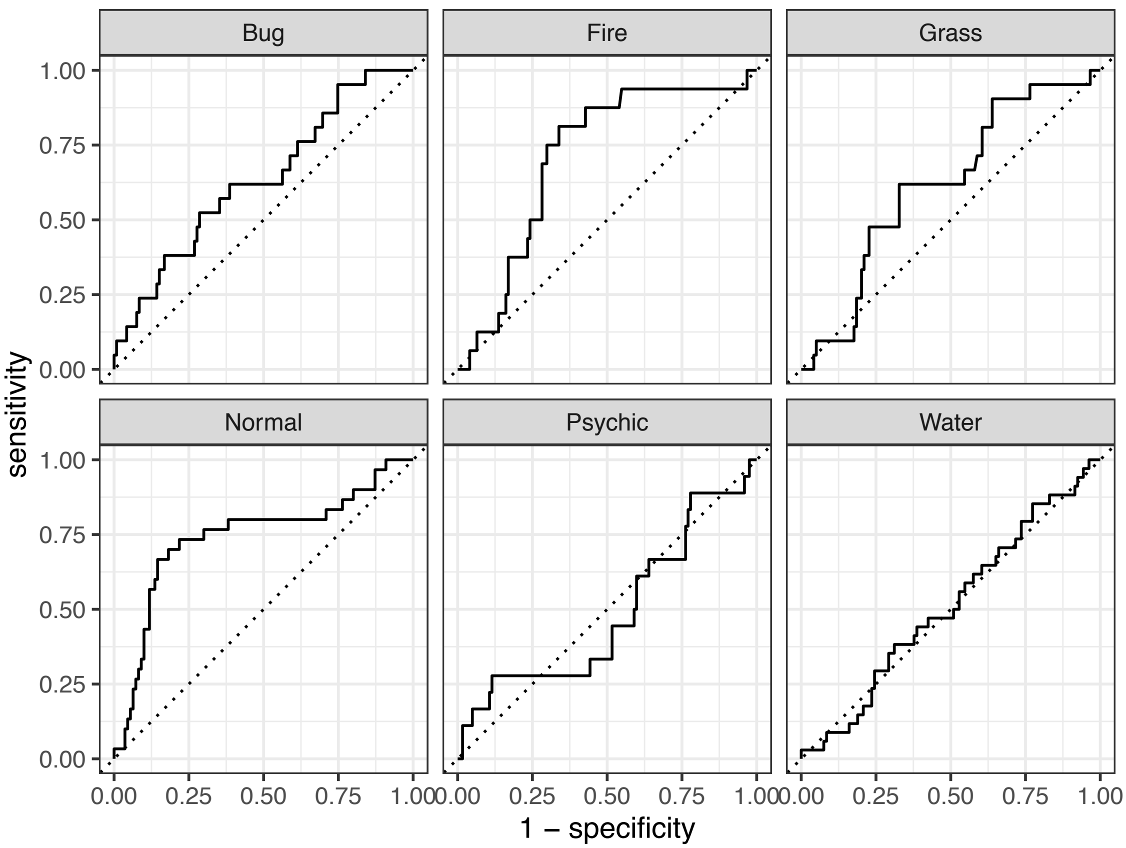
Then create plots of the different ROC curves, one per level of the outcome. Also make a heat map of the confusion matrix.

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?

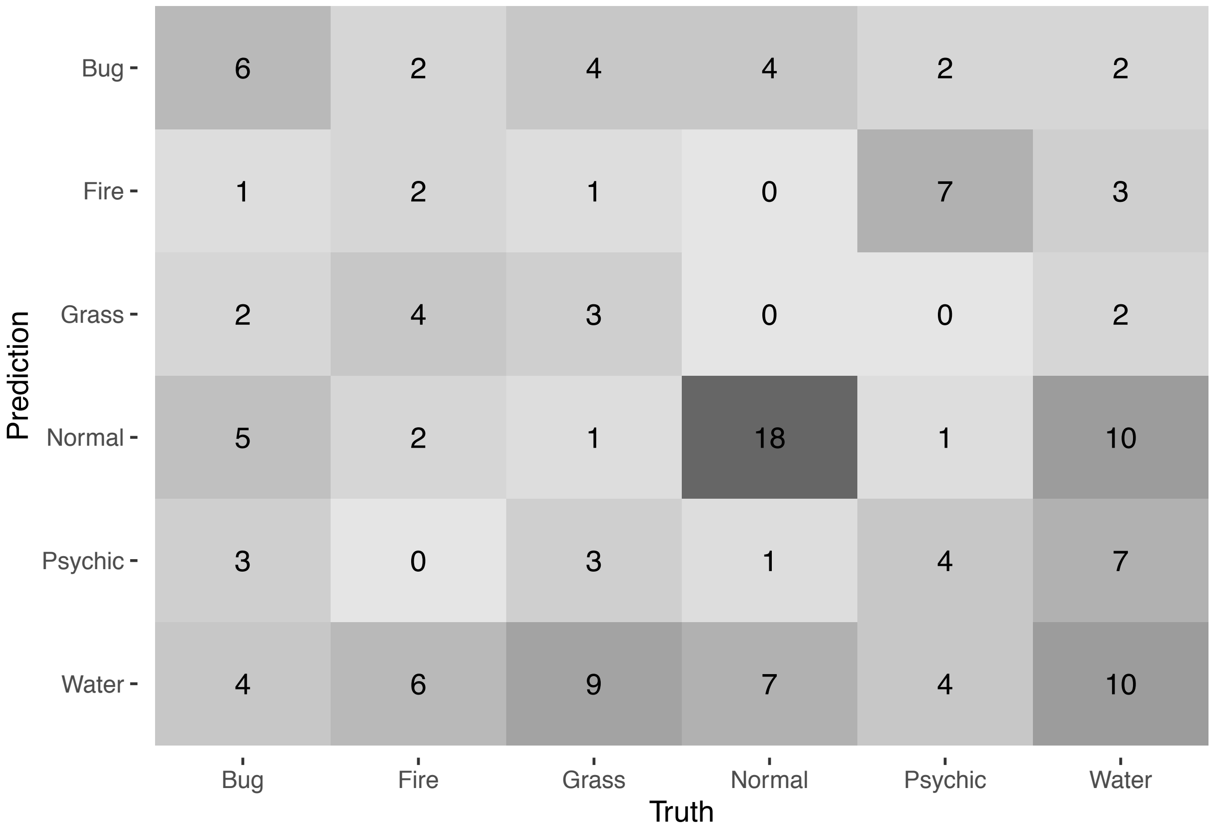
```
roc_auc(augment(pokemon_fit, new_data = pokemon_test), type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.612
```

```
augment(pokemon_fit, new_data = pokemon_test) %>%
  roc_curve(type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic, .pred_Water) %>%
  autoplot()
```



```r
augment(pokemon_fit, new_data = pokemon_test) %>%
  conf_mat(truth = type_1, estimate = .pred_class)  %>% autoplot(type = "heatmap")
```



Interpreting the results, it seems that the roc_auc metric did a lot better than the accuracy in terms of the model with an estimate of 61.2%; meaning that the model performed quite well. The model is best at predicting 'Normal' and 'Water' type Pokemon and worst at predicting the 'Fire' type Pokemon. This may be attributed to the large amount of 'Normal' and 'Water' type Pokemon as mentioned in Exercise Two with 98 and 112 Pokemon, respectively. More specifically, the class imbalance may produce inaccuracies within the model because it has a different amount of data to work with for each class.