Homework 5 (PSTAT 131/231)

Jonathan Palada Rosal

Contents

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. Darkrai, a Dark-type mythical Pokémon from Generation 4.

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(glmnet)
library(tidyverse)
library(tidymodels)
```

```
library(ISLR)
library(ggplot2)
library(dplyr)
```

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("C:\\Users\\Jonat\\OneDrive\\schoolwork\\PSTAT 131\\HW\\HW 5\\homework-5\\data\\Poke
Pokemon <- Pokemon %>%
  clean_names()
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
```

The clean_names() function just renamed or "cleaned" the predictor's names so it will be easier to code with.

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.

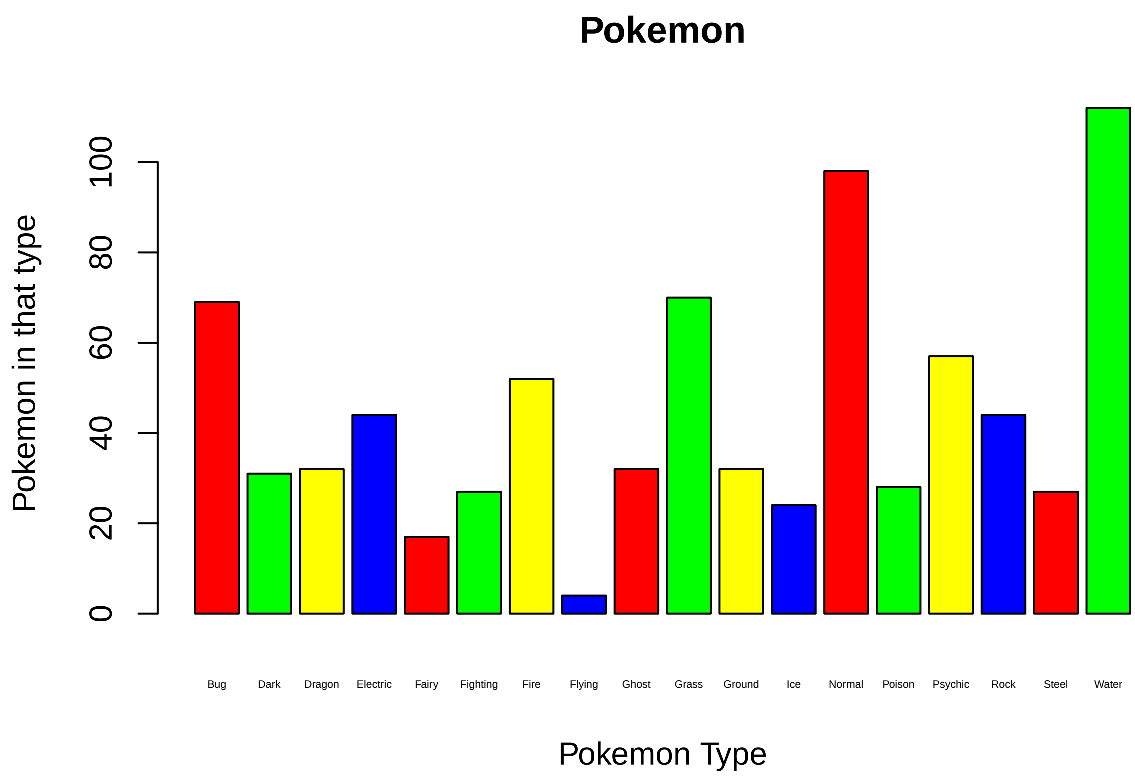
After filtering, convert `type_1` and `legendary` to factors.

```
type1 <- table(Pokemon$type_1)
type1
```

```
##
##      Bug     Dark   Dragon Electric    Fairy Fighting     Fire   Flying
```

```
##       69       31       32       44       17       27       52        4
##    Ghost    Grass   Ground      Ice   Normal   Poison  Psychic     Rock
##       32       70       32       24       98       28       57       44
##    Steel    Water
##       27      112
```

```
barplot(type1, xlab = "Pokemon Type", ylab = "Pokemon in that type", main = "Pokemon",width = 0.1, cex.r
```




```
Pokemon %>%
  group_by(type_1) %>%
  summarise(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 18 x 2
##    type_1   count
##    <chr>    <int>
##  1 Water      112
##  2 Normal      98
##  3 Grass       70
##  4 Bug         69
##  5 Psychic     57
##  6 Fire        52
##  7 Electric    44
##  8 Rock        44
##  9 Dragon      32
```

```
## 10 Ghost       32
## 11 Ground      32
## 12 Dark        31
## 13 Poison      28
## 14 Fighting    27
## 15 Steel       27
## 16 Ice         24
## 17 Fairy       17
## 18 Flying       4
```

```
Common_pokemon_types <- Pokemon %>%
   filter(type_1 == "Bug" | type_1 == "Fire" | type_1 == "Grass" | type_1 == "Normal" | type_1 == "Wate
Common_pokemon_types %>%
  group_by(type_1) %>%
  summarise(count = n()) %>%
  arrange(desc(count)) #Now there are only 6 types
```

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

```
Pokemon_factored <- Common_pokemon_types %>%
  mutate(type_1 = factor(type_1)) %>%
  mutate(legendary = factor(legendary)) %>%
  mutate(generation = factor(generation))
```

18 types of Pokemon. Flying has the least types in this dataset (4). Fairy (17) and Ice (24) are followed up with the least amount of Pokemon in that type.

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?

```
set.seed(3515)
Pokemon_split <- initial_split(Pokemon_factored, strata = type_1, prop = 0.7)
Pokemon_training <- training(Pokemon_split)
Pokemon_testing <- testing(Pokemon_split)
dim(Pokemon_training) #318 observations
```

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

```
dim(Pokemon_testing) #140 observations
```

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

```
Pokemon_fold <- vfold_cv(Pokemon_training, strata = type_1, v = 5)
Pokemon_fold
```

```
## #  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
```

Stratifying the folds will be useful because it will make sure there is a balance distribution of the types to make sure the folds are as fair as possible.

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_d
  step_dummy(legendary) %>%
  step_dummy(generation) %>%
  step_center(all_predictors()) %>%
  step_scale(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_net <- multinom_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet")
Pokemon_workflow <- workflow() %>%
  add_recipe(Pokemon_recipe) %>%
  add_model(Pokemon_net)
Pokemon_grid <- grid_regular(penalty(range = c(-5,5)), mixture(range = c(0,1)), levels = 10)
Pokemon_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
```

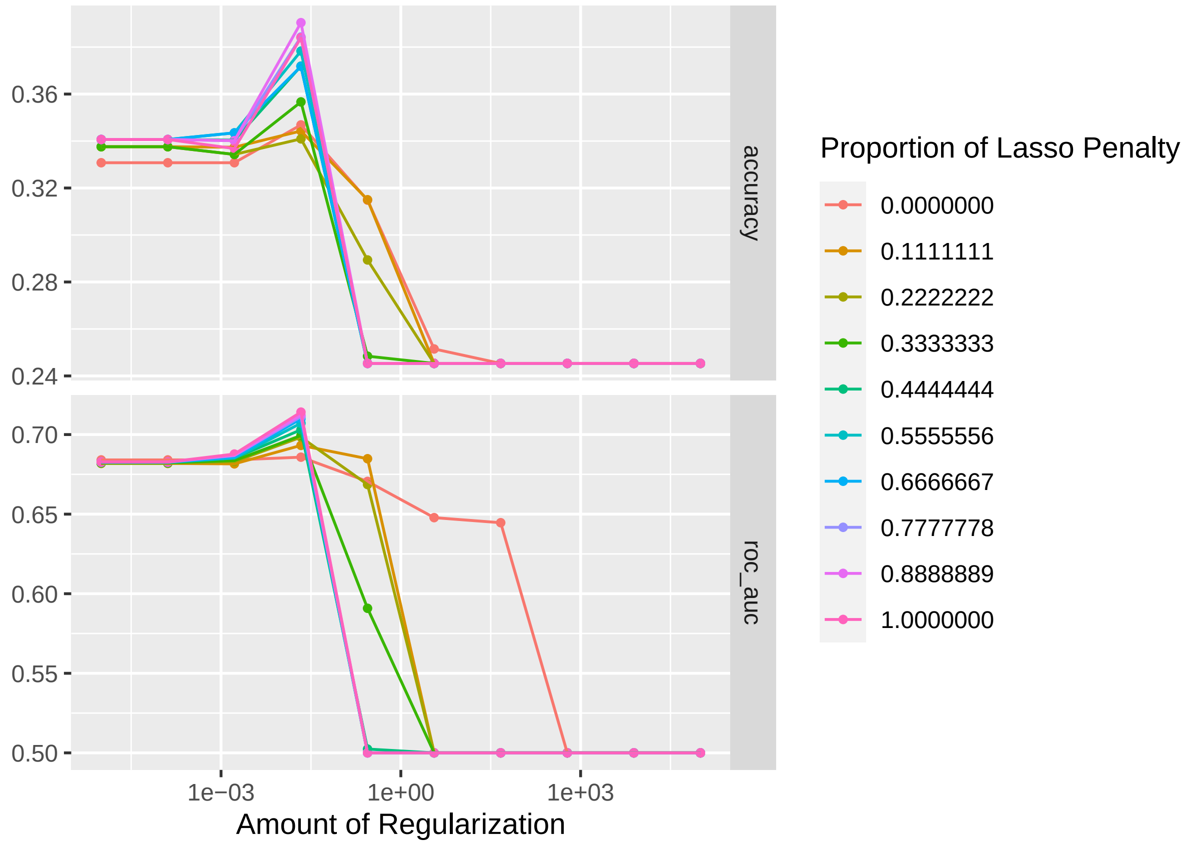
We have 100 rows and we will be doing it 5 times so we will have a total of 500 models.

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?

```
Pokemon_tune_grid <- tune_grid(object = Pokemon_workflow,
                               resamples = Pokemon_fold,
                               grid = Pokemon_grid)
autoplot(Pokemon_tune_grid)
```



The smaller values of penalty and mixture have a better accuracy and ROC AUC than the larger values of penalty and mixture. This means as the penalty and mixture increase, the accuracy and ROC AUC decreases.

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_fit <- select_best(Pokemon_tune_grid, metric = "roc_auc")
Pokemon_finalized <- finalize_workflow(Pokemon_workflow, best_fit)
Pokemon_final_fit <- fit(Pokemon_finalized, data = Pokemon_training)

Prediction <- augment(Pokemon_final_fit, new_data = Pokemon_testing) %>%
  select(type_1, .pred_class, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic, .pred_Wa
accuracy(Prediction,type_1,.pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass      0.35
```

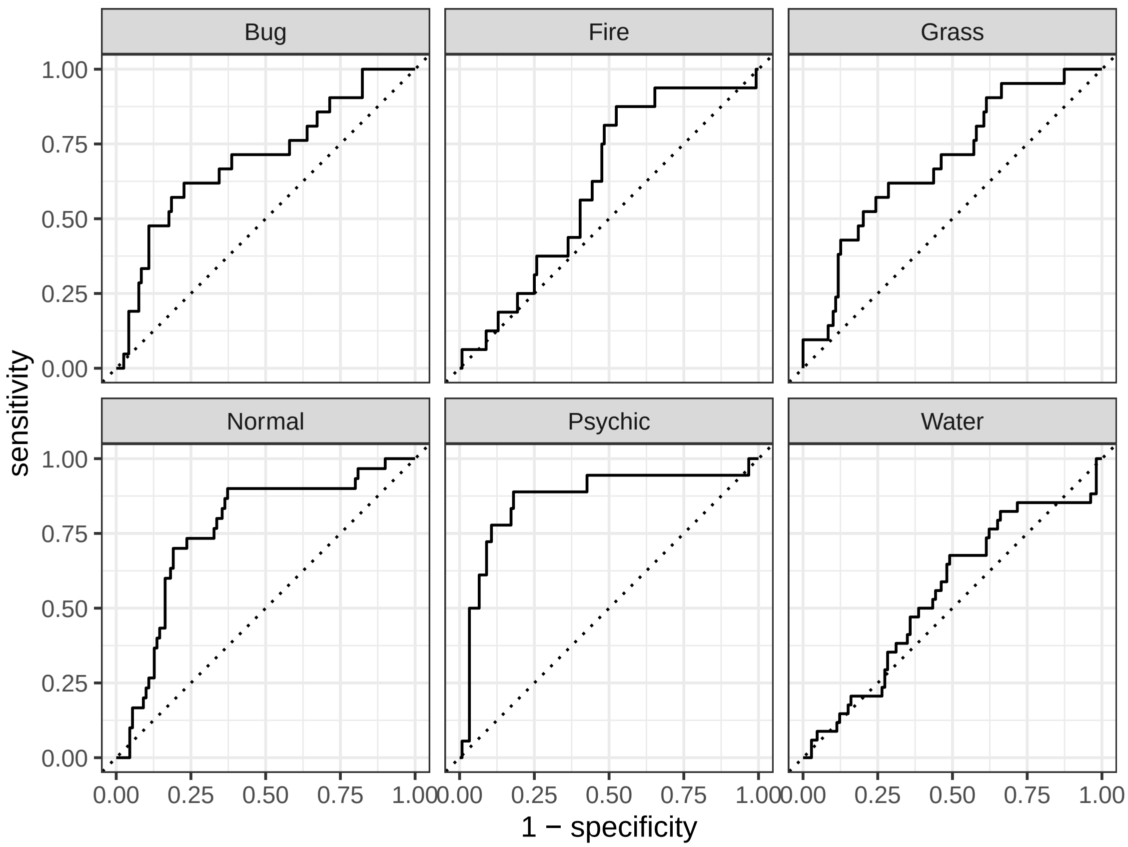
Exercise 8

Calculate the overall ROC AUC on the testing set.

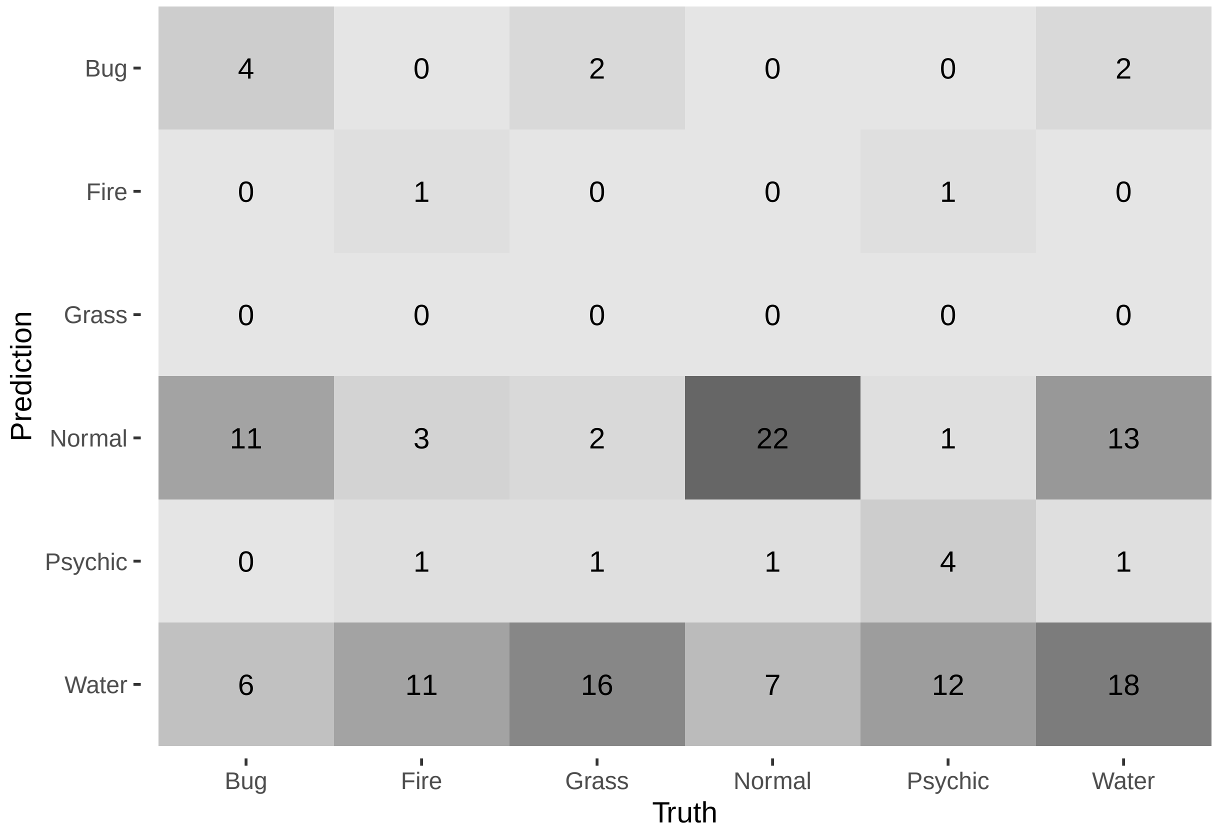
Then create plots of the different ROC curves, one per level of the outcome. Also make a heat map of the confusion matrix.

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?

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




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



Normal was predicted the most accurately while Grass was predicted the least accurately. Normal had 22 correct predictions while Grass had 0 correct predictions. I would say that Normal was the most accurate because it was second to the highest with observations (98). I think my model was able to find a pattern to accurately predict the Normal type. I think Grass contained 0 correct observations because the model was not able to find a consistent pattern that most/all Grass types contain.