Homework 5

Aidan Baker

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(tidyverse)
library(tidymodels)
library(ggplot2)
library(janitor)
library(yardstick)
```

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?

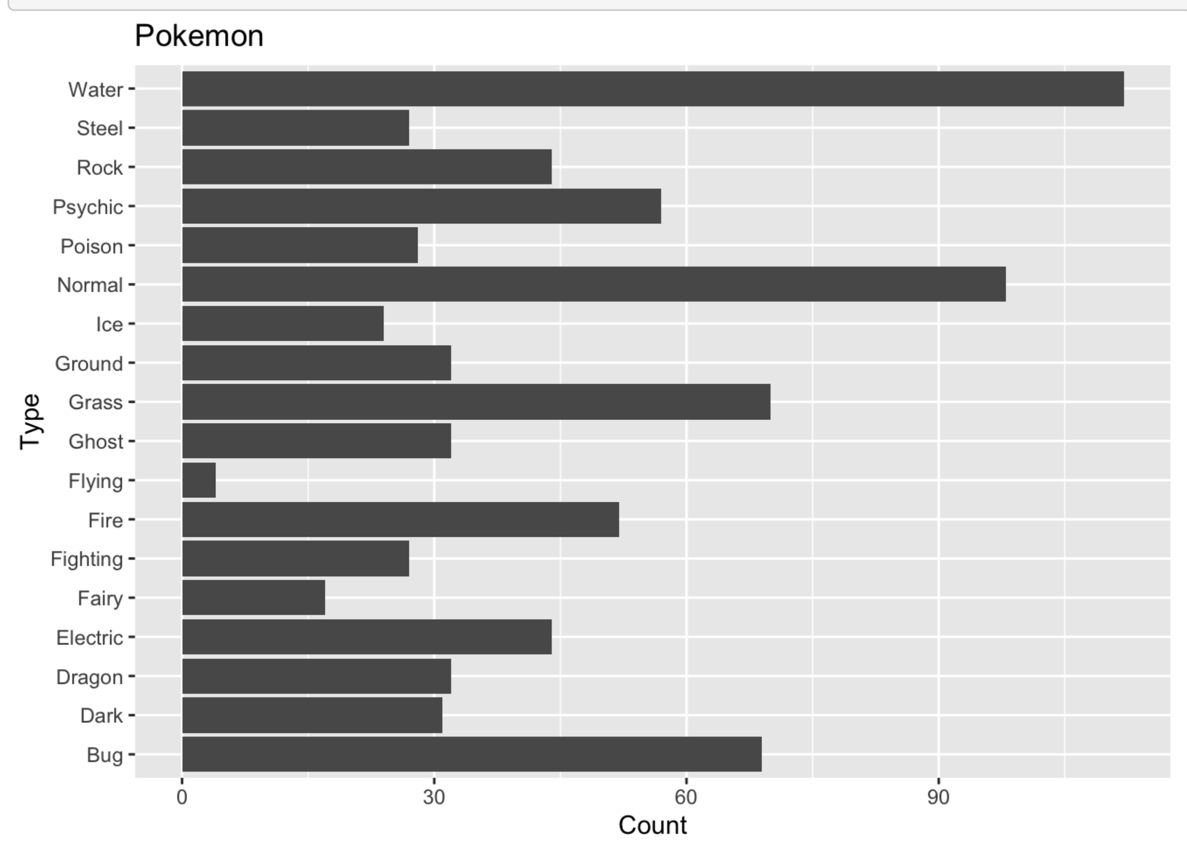
```
pokemon <- read.csv('/Users/aidanbaker/Downloads/homework-5/data/pokemon.csv') %>%
  clean_names()
```

We can see that our variable names are cleaned up without the weird periods. This will make our life easier in analyzing our predictor variables.

Exercise 2

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

```
ggplot(pokemon, aes(type_1)) +
  geom_bar() +
  labs(
    title = "Pokemon",
    x = "Type",
    y = "Count"
  ) +
  coord_flip()
```



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

```
length(unique(pokemon$type_1))
```

```
## [1] 18
```

We can see that there are 18 unique classes of the outcome

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.

```
pokemon <- pokemon %>% filter(grepl("Bug|Fire|Grass|Normal|Water|Psychic", type_1))
```

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

```
pokemon <- pokemon %>%
 mutate(type_1 = factor(type_1),
       legendary = factor(legendary),
       generation = factor (generation))
```

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(2001)

pokemon_split <- initial_split(pokemon, prop = 0.75,
                               strata = type_1)
pokemon_training <- training(pokemon_split)
pokemon_testing <- testing(pokemon_split)

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

Stratifying by type_1 on our folds will allow us to more accurately test our data against our model by isolating the variable. ### 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_def, data
= pokemon_training) %>%
  step_novel(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors())%>%
  step_zv(all_predictors()) %>%
  step_normalize(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).

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

multinom_wflow <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(multinom_model)

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

```
## # A tibble: 100 × 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
```

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

We will have a total of 500 models due to having 100 different validations of penalty and mixture multiplied by our 5 folds.

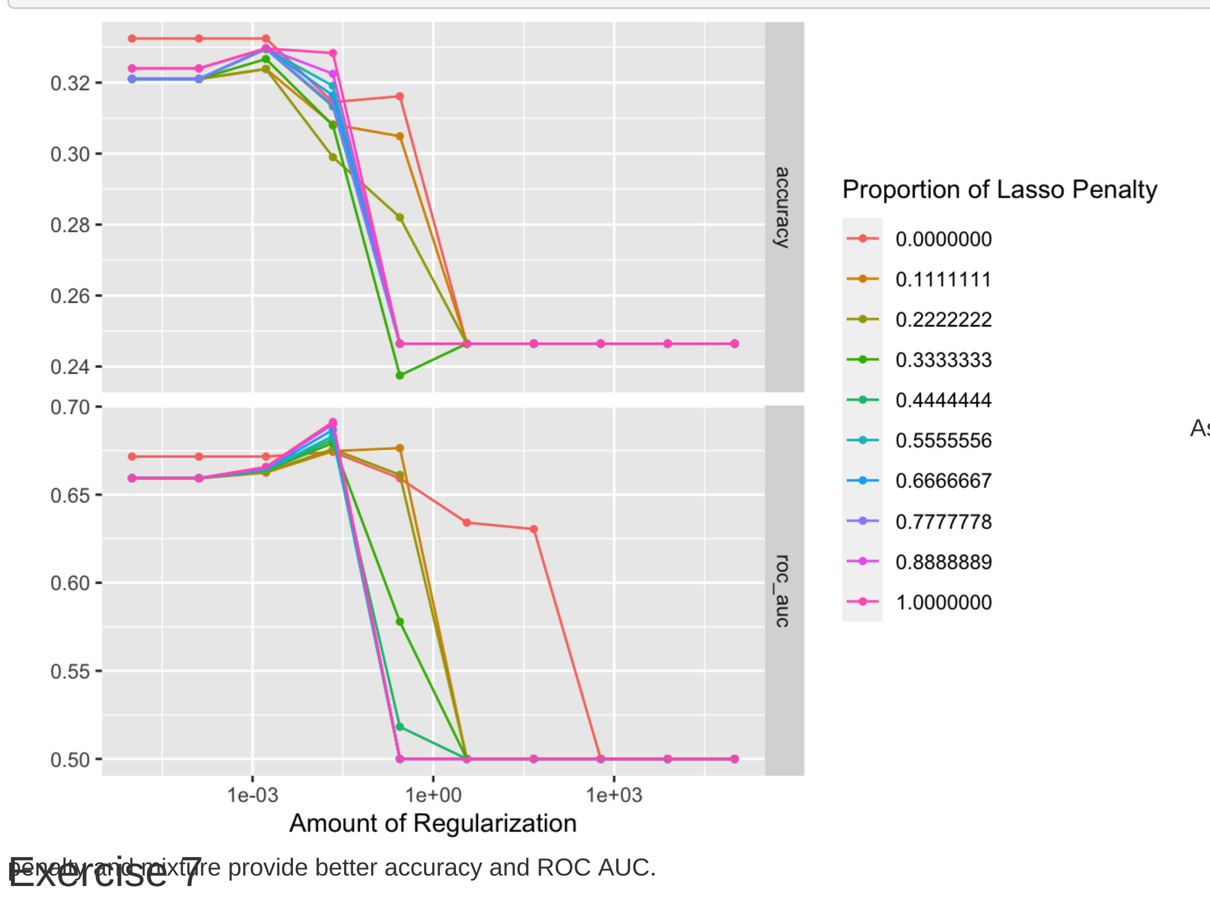
Exercise 6

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(
  multinom_wflow,
  resamples = pokemon_fold,
  grid = pokemon_grid
)
```

```
autoplot(tune_res)
```



As we can see, smaller values of penalty and mixture provide 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.

```
best <- select_best(tune_res)
multinom_final <- finalize_workflow(multinom_wflow, best)
multinom_finalfit <- fit(multinom_final, data = pokemon_training)
augment(multinom_finalfit, new_data = pokemon_testing) %>%
  accuracy(truth = type_1, estimate = .pred_class)
```

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.410
```

Exercise 8

Calculate the overall ROC AUC on the testing set.

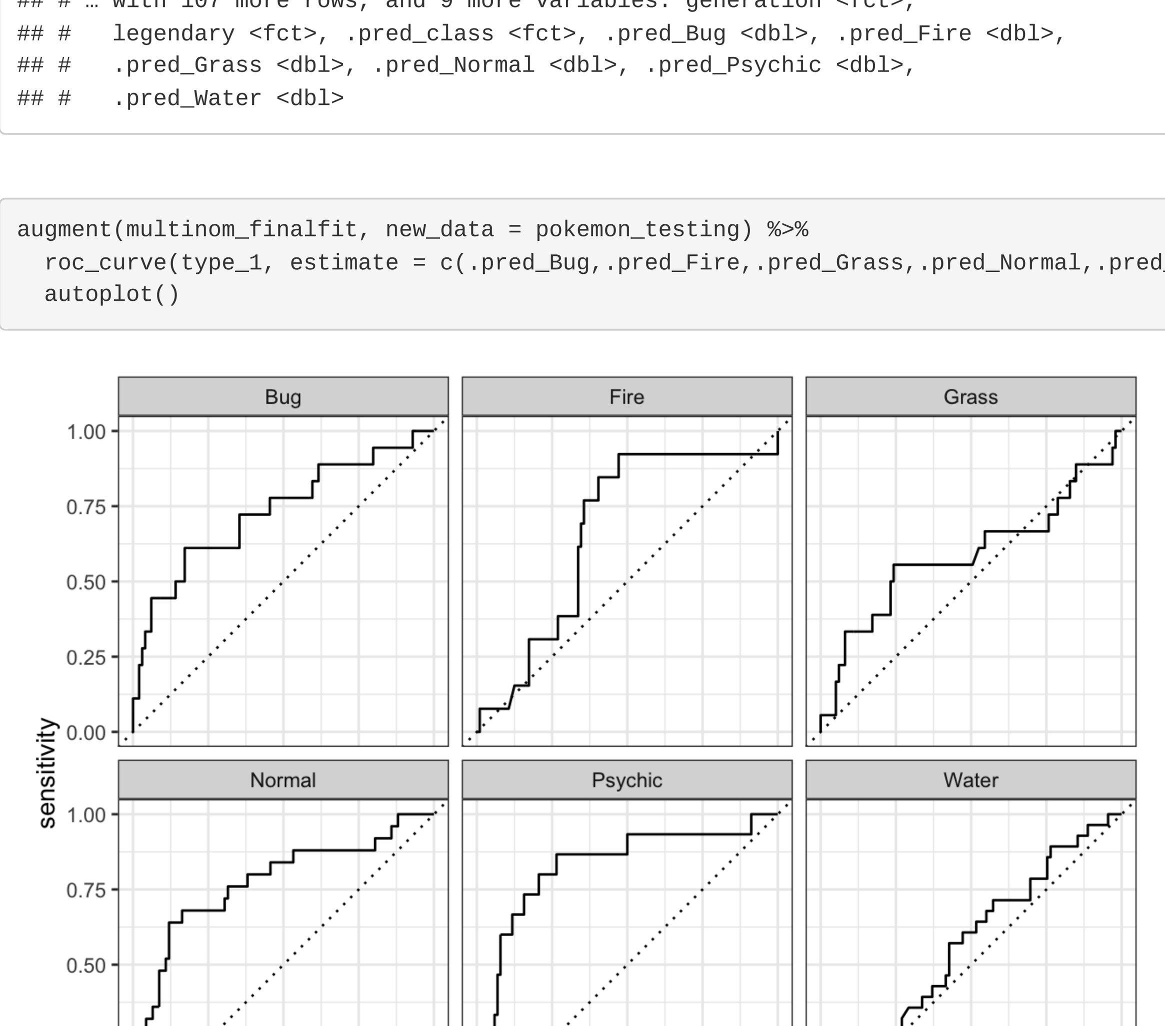
Then create plots of the different ROC curves, one per level of the outcome. Also make a heat map of the confusion matrix.

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?

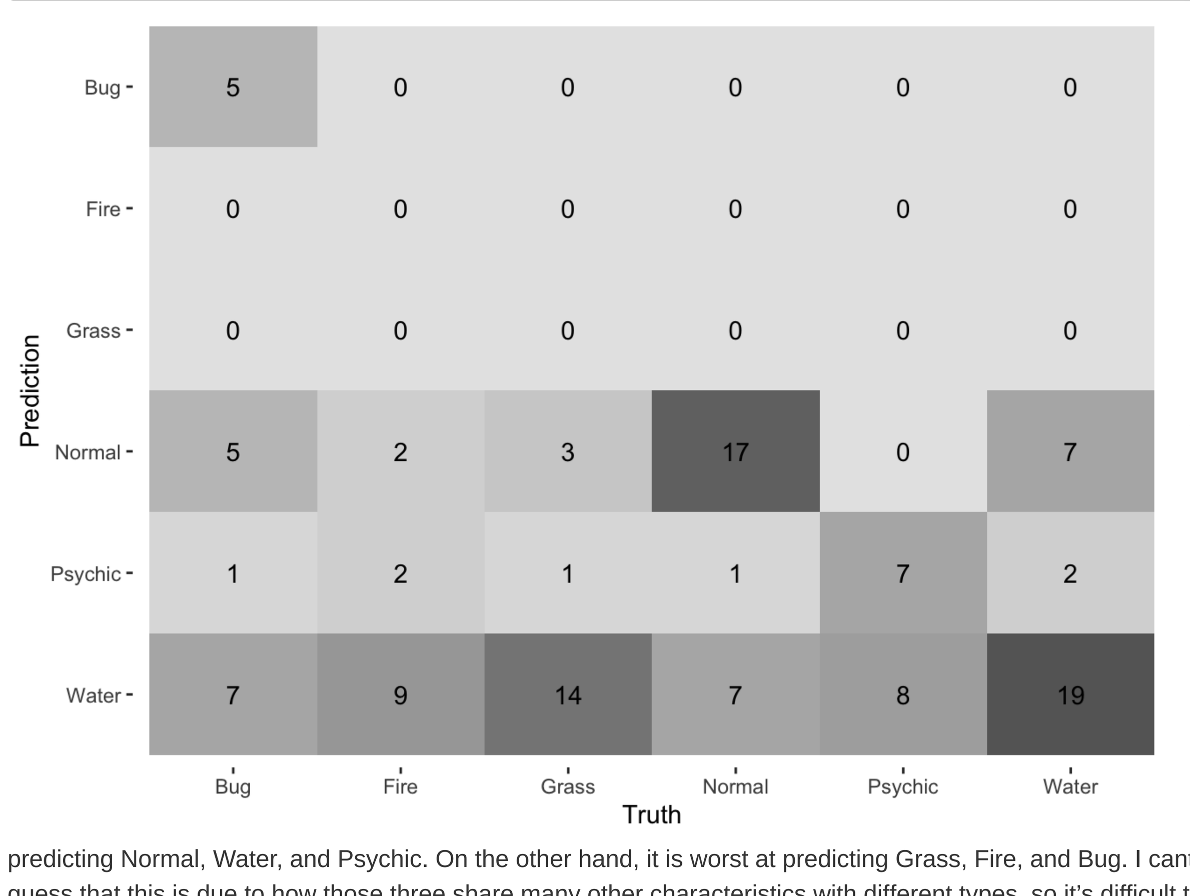
```
augment(multinom_finalfit, new_data = pokemon_testing, type = 'prob')
```

```
## # A tibble: 117 × 20
##        x name       type_1 type_2 total    hp attack defense sp_atk sp_def speed
##    <int> <chr>      <fct>  <chr>  <int> <int>  <int>   <int>  <int>  <int> <int>
##  1     1 Bulbasaur  Grass  "Pois…   318    45     49      49     65     65    45
##  2     2 Ivysaur    Grass  "Pois…   405    60     62      63     80     80    60
##  3     5 Charmeleon Fire   ""       405    58     64      58     80     65    80
##  4     6 Charizard… Fire   "Flyi…   634    78    104      78    159    115   100
##  5     7 Squirtle   Water  ""       314    44     48      65     50     64    43
##  6     8 Wartortle  Water  ""       405    59     63      80     65     80    58
##  7    11 Metapod    Bug    ""       205    50     20      55     25     25    30
##  8    12 Butterfree Bug    "Flyi…   395    60     45      50     90     80    70
##  9    16 Pidgey     Normal "Flyi…   251    40     45      40     35     35    56
## 10    18 PidgeotMe… Normal "Flyi…   579    83     80      80    135     80   121
## # … with 107 more rows, and 9 more variables: generation <fct>,
## #   legendary <fct>, .pred_class <fct>, .pred_Bug <dbl>, .pred_Fire <dbl>,
## #   .pred_Grass <dbl>, .pred_Normal <dbl>, .pred_Psychic <dbl>,
## #   .pred_Water <dbl>
```

```
augment(multinom_finalfit, new_data = pokemon_testing) %>%
  roc_curve(type_1, estimate = c(.pred_Bug,.pred_Fire,.pred_Grass,.pred_Normal,.pred_Psychic,.pred_Water)) %>%
  autoplot()
```




```
augment(multinom_finalfit, new_data = pokemon_testing) %>%
  conf_mat(truth = type_1, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```



As we can see, the model is best at predicting Normal, Water, and Psychic. On the other hand, it is worst at predicting Grass, Fire, and Bug. I cant say im positive why this is, but I'd guess that this is due to how those three share many other characteristics with different types, so it's difficult to state which one is which.