Homework 5

Tonia Wu

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.

Read in the file and familiarize yourself with the variables using pokemon_codebook.txt.

```
set.seed = 667
library(tidyverse)
library(tidymodels)
library(ggplot2)
library(yardstick)
library(dplyr)
library(corrr)
library(klaR)
library(discrim)
library(poissonreg)
library(pROC)
library(MASS)
library(glmnet)
rawdata <- read.csv('C:\\Users\\me\\Downloads\\homework-5\\homework-5\\data\\Pokemon.csv')
```

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 <- rawdata %>%
  clean_names()
head(rawdata)
```

```
##   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
```

```
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
```

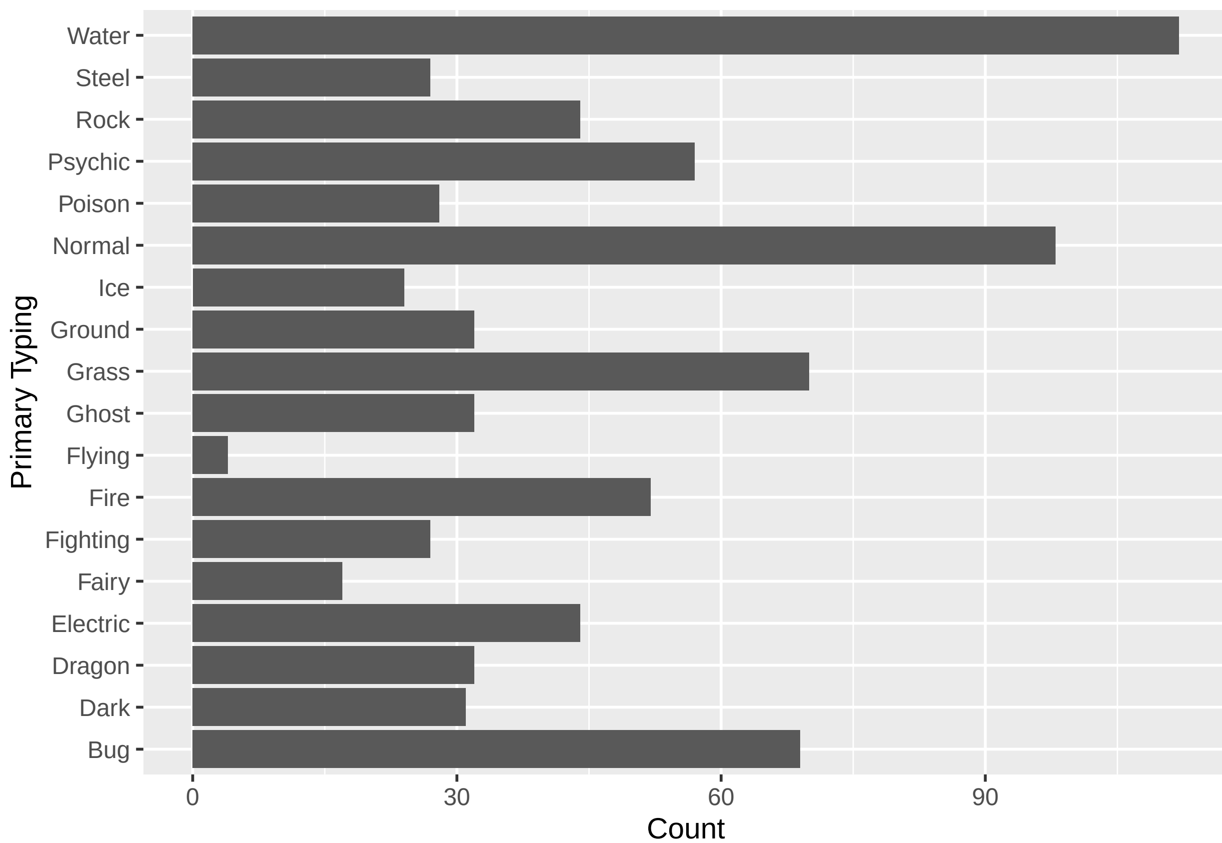
Column headers are now all lowercase, and the only delimiters are underscores. Standardizing the variables makes the data easier to work with, for example reducing the chance of forgetting a period in 'Sp..Atk' and giving the programmer unecessary headaches.

Exercise 2

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

```
bchart1 <- ggplot(data = pokemon, aes(x = type_1)) +
geom_bar() + coord_flip() + labs(y = 'Count', x = 'Primary Typing')

bchart1
```



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

> There are 18 primary typings, with Flying having the least. Fairy is also quite rare, but its count is closer to that of the next cluster of uncommon pokemon than it is to Flying.

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.

```
filter_array <- c('Bug', 'Fire', 'Grass', 'Normal', 'Water', 'Psychic')
pokemon1 <- filter(pokemon, type_1 %in% filter_array)
```

Converting `type_1`, `legendary`, and generation to factors.

```
pokemon1$type_1 <- as.factor(pokemon1$type_1)
pokemon1$legendary <- as.factor(pokemon1$legendary)
pokemon1$generation <- as.factor(pokemon1$generation)
```

Exercise 3

Perform an initial split of the data. Stratify by the outcome variable.

```r
p_split <- pokemon1 %>%
  initial_split(prop = 0.8, strata = 'type_1')

p_train <- training(p_split)
p_test <- testing(p_split)
```

Verifying number of observations:

```r
dim(pokemon1)
```

```
## [1] 458  13
```

```r
dim(p_train)
```

```
## [1] 364  13
```

```r
dim(p_test)
```

```
## [1] 94 13
```

> The training and testing sets have 364 and 94 observations, respectively. As the full dataset has 458 observations, this means we have a 79.5% to 20.5% split, which is close to 80/20.

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
p_folded <- vfold_cv(p_train, v = 5, strata = 'type_1')
```

> We want to stratify since we do not have an equal number of Pokemon per primary typing. This makes the folds more balanced and improves our model.

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.

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

p_recipe
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          8
##
## Operations:
##
## Dummy variables from legendary
## Dummy variables from generation
## Centering and 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.

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

p_wkflow <- workflow() %>%
  add_model(p_multireg) %>%
  add_recipe(p_recipe)
```

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).

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

p_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
```

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

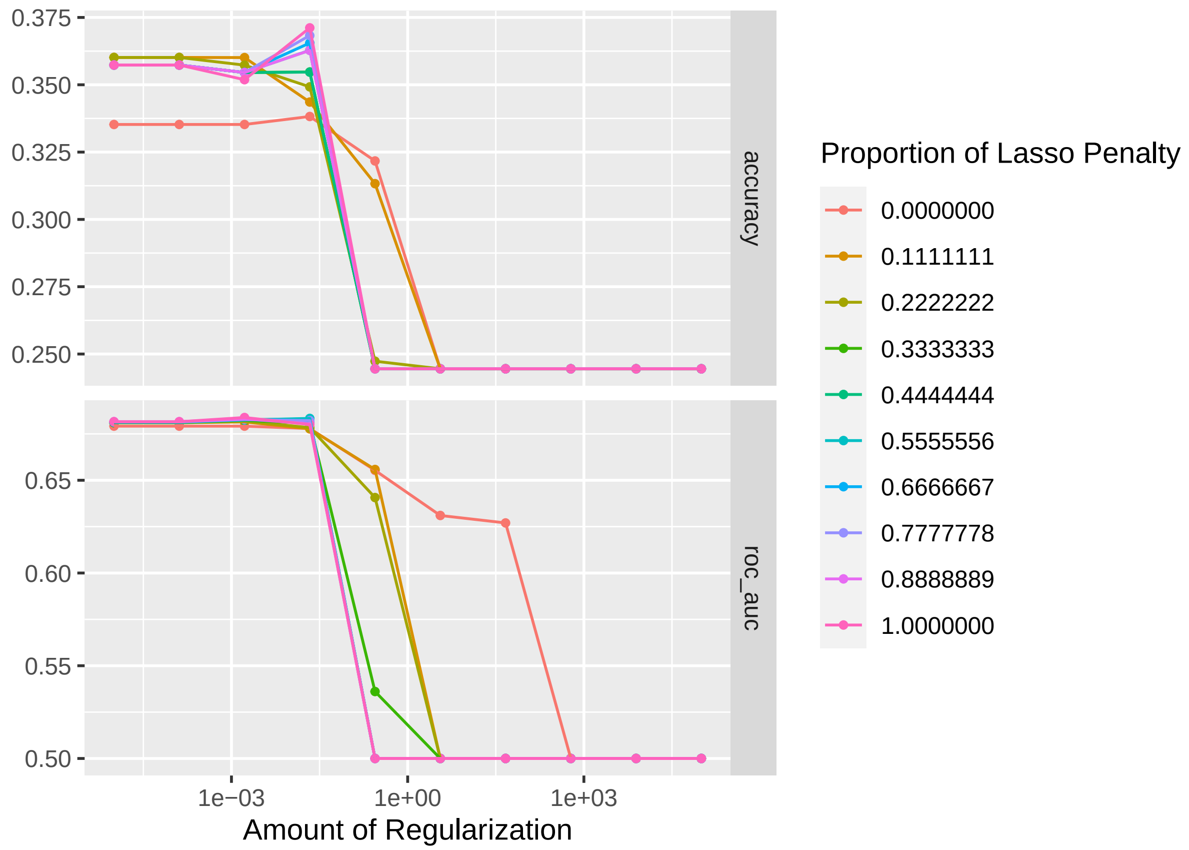
> There are 5 folds and 100 rows in the grid (10 penalty * 10 mixture), so we will be fitting 500 models.

Exercise 6

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

```
p_tune <- tune_grid(p_wkflow,
                    resamples = p_folded,
                    grid = p_grid)

autoplot(p_tune)
```



Use `autoplot()` on the results. What do you notice? Do larger or smaller values of `penalty` and `mixture` produce better accuracy and ROC AUC?

> Lower values of penalty and mixture had higher accuracy and ROc AUC than did the larger ones.

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.

```r
p_best <- p_tune %>%
  select_best('roc_auc')

p_final_wkflow <- p_wkflow %>%
  finalize_workflow(p_best)

p_fit <- p_final_wkflow %>%
  fit(p_train)

p_prediction <- augment(p_fit, p_test)
```

Exercise 8

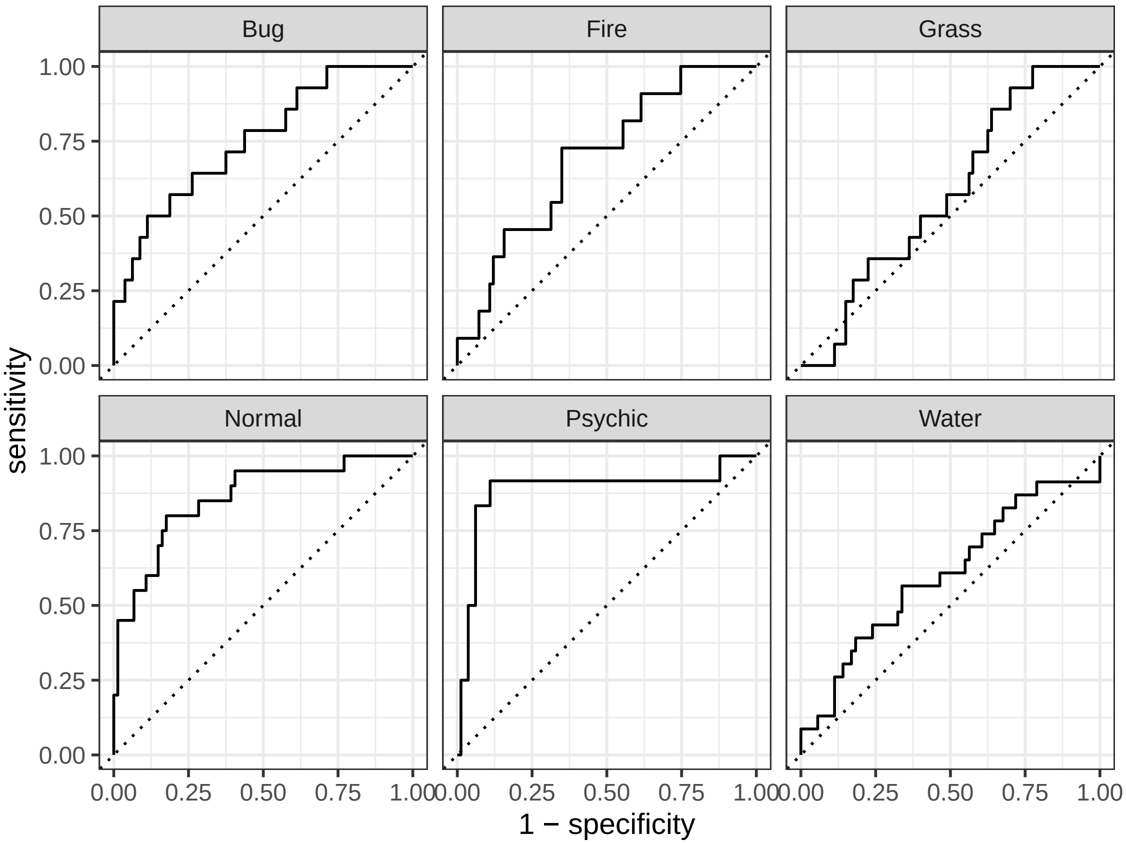
Calculate the overall ROC AUC on the testing set.

```r
accuracy(p_prediction, truth = type_1, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.394
```

Then create plots of the different ROC curves, one per level of the outcome.

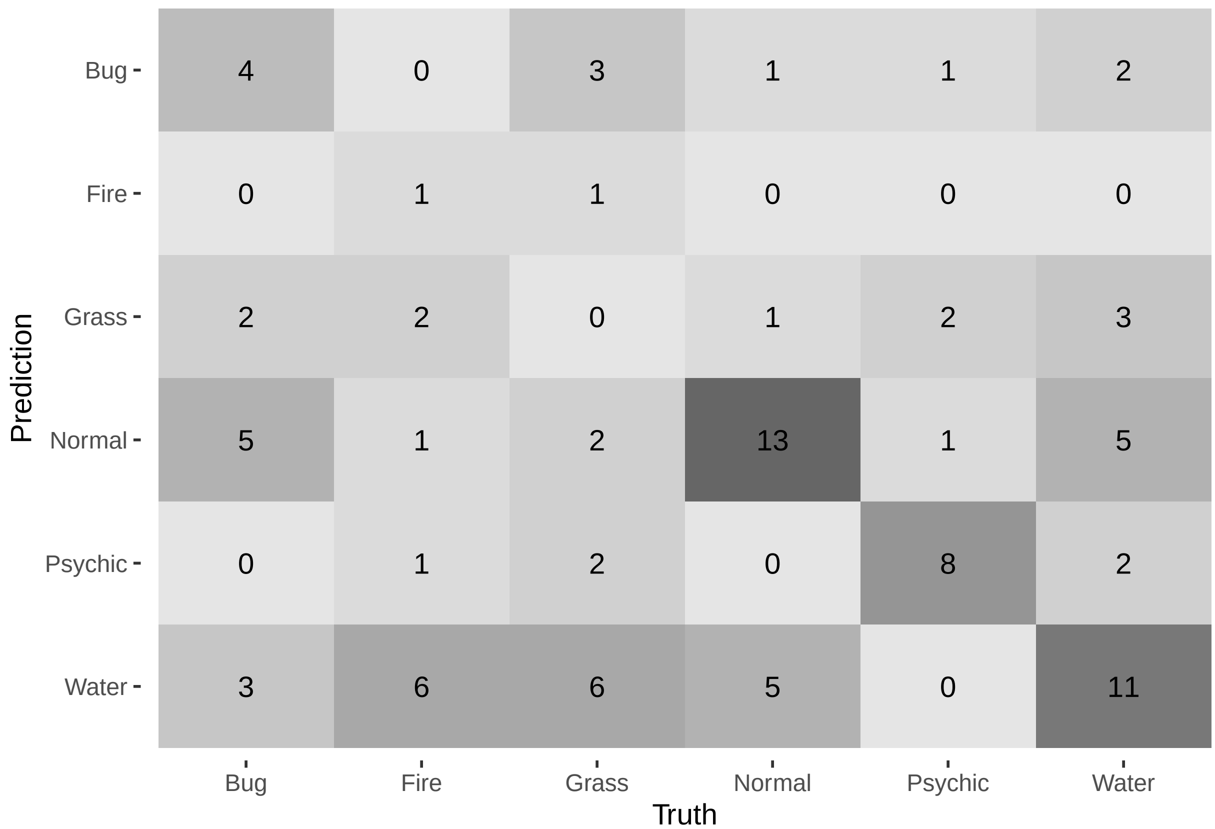
```r
roc_test <- roc_curve(data = p_prediction,
                    truth = type_1,
                    estimate = .pred_Bug:.pred_Water)
autoplot(roc_test)
```



Also make a heat map of the confusion matrix.

```
confusion_matrix <- conf_mat(data = p_prediction,
                             truth = type_1,
                             estimate = .pred_class) %>%
  autoplot(type = 'heatmap')

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?

For 231 Students

Exercise 9

In the 2020-2021 season, Stephen Curry, an NBA basketball player, made 337 out of 801 three point shot attempts (42.1%). Use bootstrap resampling on a sequence of 337 1's (makes) and 464 0's (misses). For each bootstrap sample, compute and save the sample mean (e.g. bootstrap FG% for the player). Use 1000 bootstrap samples to plot a histogram of those values. Compute the 99% bootstrap confidence interval for Stephen Curry's "true" end-of-season FG% using the quantile function in R. Print the endpoints of this interval.