Code

Homework 5

PSTAT 131/231

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.

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`.

Hide

```
library(ggplot2)
library(tidyverse)
library(tidymodels)
library(corrplot)
library(ggthemes)
library(corrr)
library(discrim)
#install.packages("glmnet")
library(glmnet)
#install.packages("pROC")
library(pROC)
library(klaR)
tidymodels_prefer()
 setwd("/Users/abhayzope/Desktop/Pstat 131")
Pokemon_data=read.csv("Pokemon.csv")
Pokemon_data %>%
 head()
```

```
##   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?

Hide

```
#install.packages("janitor")
 library(janitor)
new_Pokemon <-Pokemon_data %>%
   clean_names()
```

In this case, every dataframe's column was changed from uppercase to lowercase. Clean_names() is useful as it will make all of the names in a dataframe easier to work with.

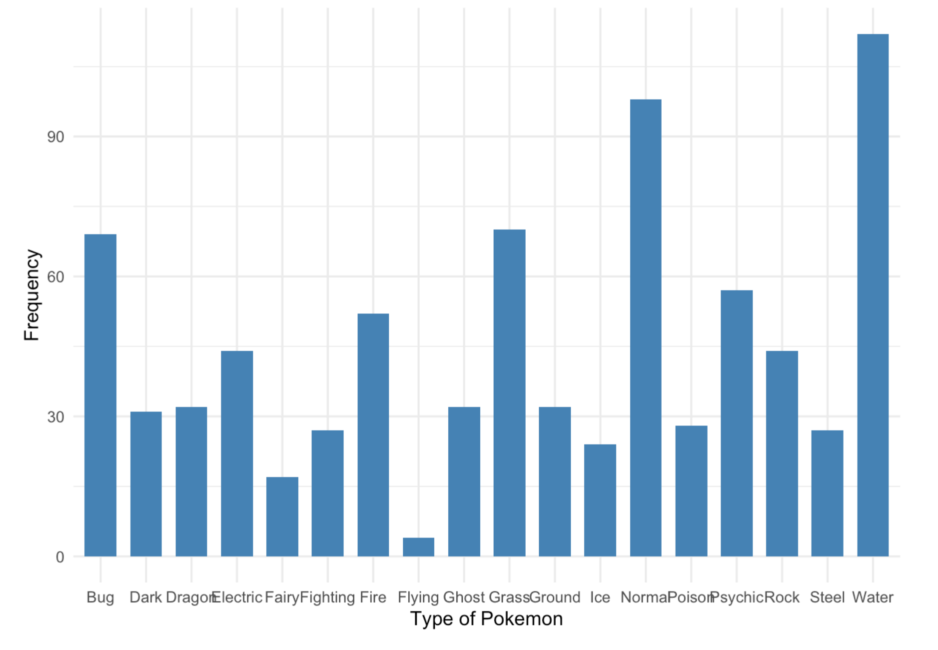
Exercise 2

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?

Hide

```
 ggplot(data = new_Pokemon, aes(x=type_1))+
  geom_histogram(stat="count", width=0.7, fill="steelblue")+
  theme_minimal()+ labs(x = "Type of Pokemon", y = "Frequency")
```



We have 18 different classes of Pokemon in our dataset. The pokemon types of Flying and Fairy all contain very few Pokemon in particular.

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.

Hide

```
 new_Pokemon <- new_Pokemon  %>% filter(type_1 == "Bug" | type_1 == "Grass" | type_1 == "Fire" |  type_
1 == "Normal" | type_1 == "Water" | type_1 == "Psychic")
#new_Pokemon
```

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

Hide

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

Hide

```
set.seed(3435)

Pokemon_split <- initial_split(new_Pokemon, prop = 0.80,
                                strata = type_1)
Pokemon_train <- training(Pokemon_split)
Pokemon_test <- testing(Pokemon_split)

dim(Pokemon_train)
```

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

Hide

```
dim(Pokemon_test)
```

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

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?

Hide

```
Pokemon_folds <- vfold_cv(Pokemon_train, v = 5, strata = type_1)
Pokemon_folds
```

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 × 2
##   splits           id
##   <list>           <chr>
## 1 <split [289/75]> Fold1
## 2 <split [291/73]> Fold2
## 3 <split [291/73]> Fold3
## 4 <split [292/72]> Fold4
## 5 <split [293/71]> Fold5
```

Stratifying on folds allows each fold to be representative of the data as a whole. This will consequently ensure that our cross-validation results are more accurate than they would have been otherwise.

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.

Hide

```
Pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed
                            + defense + hp + sp_def, data = Pokemon_train) %>%
  step_dummy(legendary) %>%
  step_dummy(generation) %>%
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).

Hide

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

Hide

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

Hide

```
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?

We will be fitting 500 models when we fit the models to the folded data.

Exercise 6

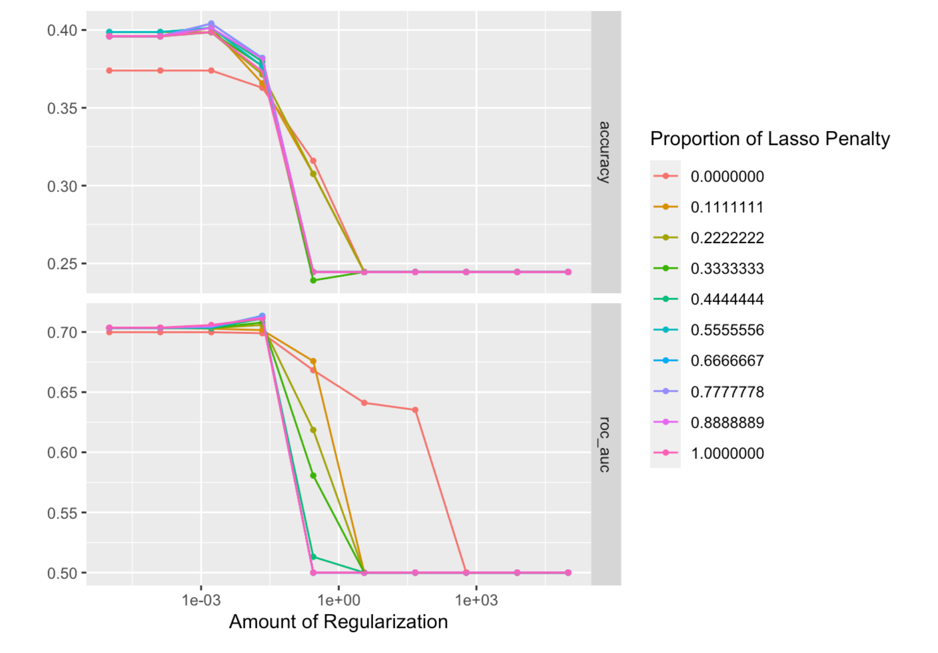
Fit the models to your folded data using `tune_grid()`.

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

Hide

```
Pokemon_res <- tune_grid(
  Pokemon_workflow,
  resamples = Pokemon_folds,
  grid = penalty_grid
)

autoplot(Pokemon_res)
```



The visualization above indicates that smaller values `penalty` and `mixture` produce 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.

Hide

```
best <- select_best(Pokemon_res, metric = "roc_auc")

Pokemon_final <- finalize_workflow(Pokemon_workflow, best)

Pokemon_final_fit <- fit(Pokemon_final, data = Pokemon_train)

modelaccuracy<- augment(Pokemon_final_fit, new_data = Pokemon_test) %>%
  accuracy(truth = type_1, estimate = .pred_class)
modelaccuracy
```

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.340
```

Exercise 8

Calculate the overall ROC AUC on the testing set.

Hide

```
augment(Pokemon_final_fit, new_data = Pokemon_test, metric='roc_auc')
```

```
## # A tibble: 94 × 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     3 VenusaurM… Grass  "Pois…   625    80    100     123    122    120    80
##  2     9 Blastoise… Water  ""       630    79    103     120    135    115    78
##  3    15 Beedrill   Bug    "Pois…   395    65     90      40     45     80    75
##  4    15 BeedrillM… Bug    "Pois…   495    65    150      40     15     80   145
##  5    20 Raticate   Normal ""       413    55     81      60     50     70    97
##  6    38 Ninetales  Fire   ""       505    73     76      75     81    100   100
##  7    40 Wigglytuff Normal "Fair…   435   140     70      45     85     50    45
##  8    60 Poliwag    Water  ""       300    40     50      40     40     40    90
##  9    84 Doduo      Normal "Flyi…   310    35     85      45     35     35    75
## 10    85 Dodrio     Normal "Flyi…   460    60    110      70     60     60   100
## # … with 84 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>
```

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

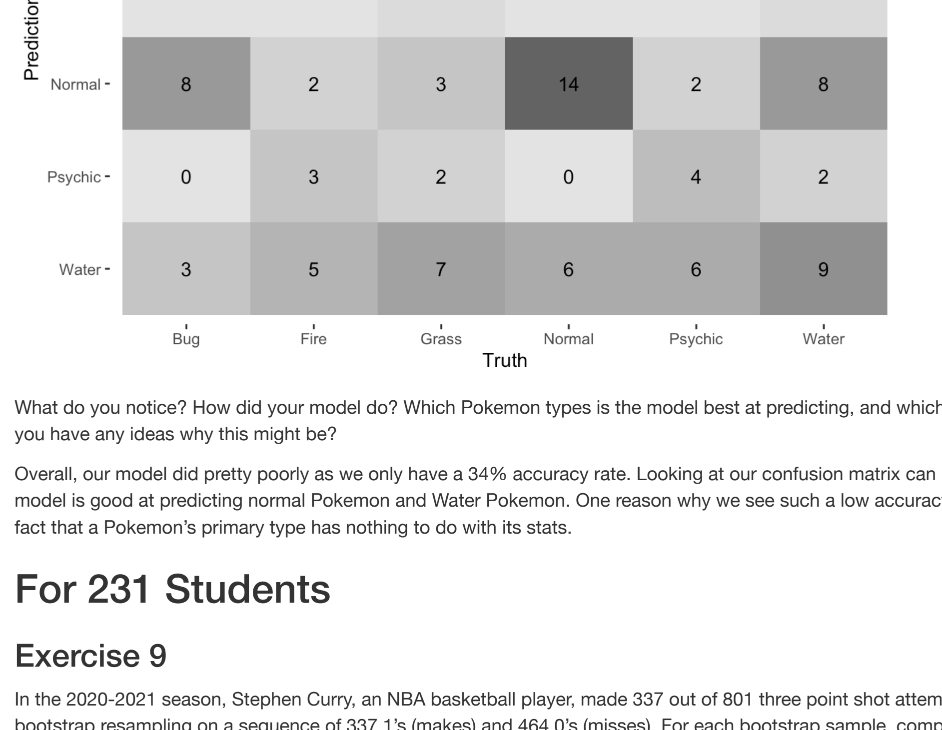
Hide

```
augment(Pokemon_final_fit, new_data = Pokemon_test, metric='roc_auc') %>%
  roc_curve(type_1, estimate = c(.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Water, .pred_P
sychic))
```

```
## # A tibble: 576 × 4
##    .level .threshold specificity sensitivity
##    <chr>       <dbl>       <dbl>       <dbl>
##  1 Bug     -Inf           0                1
##  2 Bug        0.00140     0                1
##  3 Bug        0.00715     0.0125           1
##  4 Bug        0.0115      0.025            1
##  5 Bug        0.0327      0.0375           1
##  6 Bug        0.0357      0.05             1
##  7 Bug        0.0374      0.0625           1
##  8 Bug        0.0430      0.075            1
##  9 Bug        0.0452      0.0875           1
## 10 Bug        0.0457      0.1              1
## # … with 566 more rows
```

Hide

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



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?

Overall, our model did pretty poorly as we only have a 34% accuracy rate. Looking at our confusion matrix can indicate that the model is good at predicting normal Pokemon and Water Pokemon. One reason why we see such a low accuracy rate is due to the fact that a Pokemon's primary type has nothing to do with its stats.

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.