Homework 5

PSTAT 131/231

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. 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(tidymodels)
library(tidyverse)
library(ISLR) # For the Smarket data set
library(ISLR2) # For the Bikeshare data set
library(discrim)
```

```r
library(poissonreg)
library(corrr)
library(klaR) # for naive bayes
library(forcats)
library(corrplot)
library(pROC)
library(recipes)
library(rsample)
library(parsnip)
library(workflows)
library(janitor)
library(glmnet)
tidymodels_prefer()
set.seed(2022)
```

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?

```r
# install.packages("janitor")
pokemon_raw <- read.csv("Pokemon.csv")
head(pokemon_raw)
```

```
##   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
```

```r
pokemon1 <- clean_names(pokemon_raw)
head(pokemon1)
```

```
##   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
```

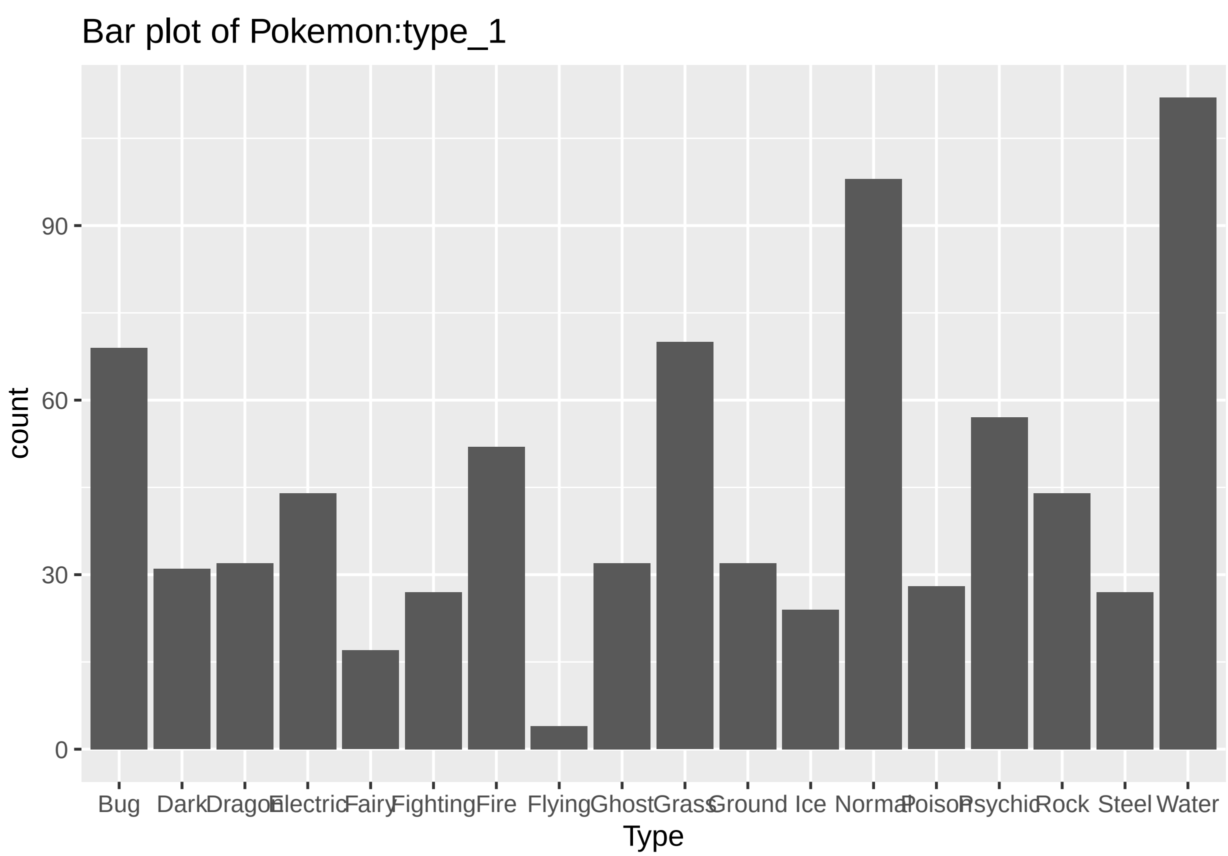
```
## 1    45          1     False
## 2    60          1     False
## 3    80          1     False
## 4    80          1     False
## 5    65          1     False
## 6    80          1     False
```

All column name convert to lower case and all of them are unique, also the name consist '.' convert to '_'. *Resulting names are unique and consist only of the "_" character, numbers, and letters which is more effciency.*

Exercise 2

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

```
ggplot(pokemon1, aes(x= type_1)) +
  geom_bar(stat = "count") +
  ggtitle("Bar plot of Pokemon:type_1") +
  xlab("Type")
```



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

Class: 18.
The flying type with few Pokemon.

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.

```
pokemon <- pokemon1[which(pokemon1$type_1 == "Bug"| pokemon1$type_1 =="Fire"|
                     pokemon1$type_1 =="Grass"| pokemon1$type_1 =="Normal"|
                     pokemon1$type_1 =="Water"| pokemon1$type_1 =="Psychic"), ]
pokemon <- pokemon %>%
              mutate(type_1 = factor(type_1),
                  legendary =factor(legendary))
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
```

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.

```
pokemon_split <- pokemon %>%
  initial_split(strata = type_1, prop = 0.7)
pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)
dim(pokemon_train)
```

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

```
dim(pokemon_test)
```

```
## [1] 140  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?

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

Each resample is created within the stratification variable that each fold is an appropriate representative of the original data.

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, pokemon_train) %>%
  step_dummy(legendary,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_spec <- multinom_reg(mixture = tune(), penalty = tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

pokemon_workflow <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(pokemon_spec)

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

There will be 500 models in total.

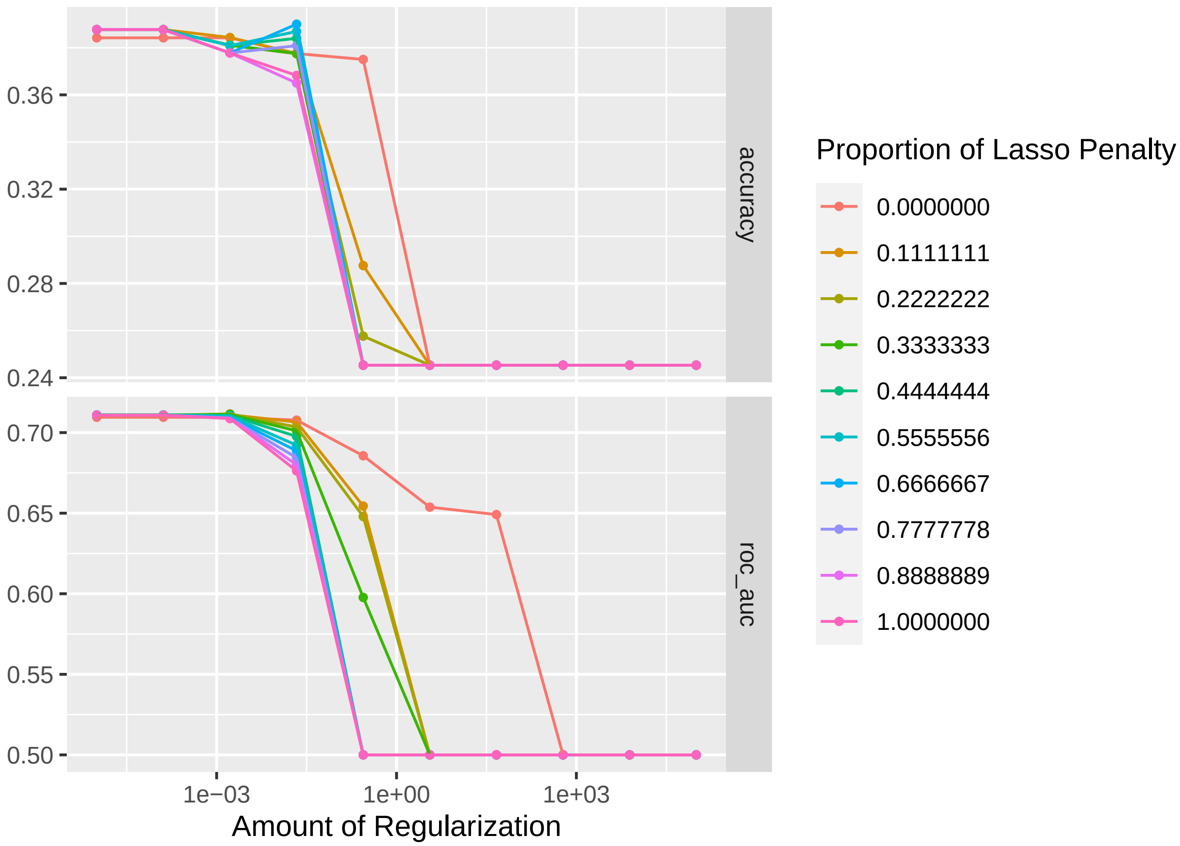
Exercise 6

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(
  pokemon_workflow,
  resamples = pokemon_fold,
  grid =  penalty_grid
)

autoplot(tune_res)
```



I notice that less lasso penalty produce higher accuracy and roc_auc. And smaller 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.

```
best_model <- select_best(tune_res, metric = "roc_auc")
pokemon_final <- finalize_workflow(pokemon_workflow, best_model)
pokemon_final_fit <-  fit(pokemon_final, data = pokemon_train)
```

```
predicted_data <- augment(pokemon_final_fit, new_data = pokemon_test) %>%
                  select(type_1,starts_with(".pred"))
```

Exercise 8

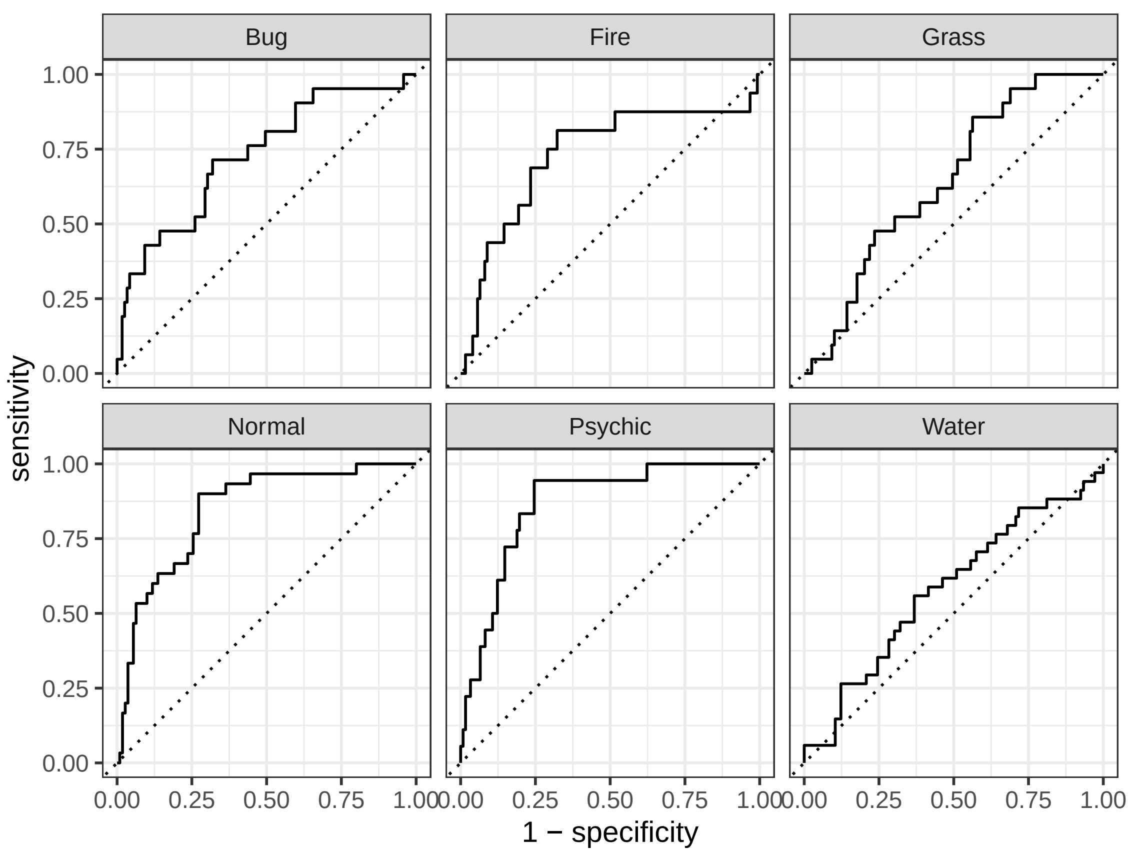
Calculate the overall ROC AUC on the testing set.

```
predicted_data %>% roc_auc(type_1, .pred_Bug:.pred_Water)
```

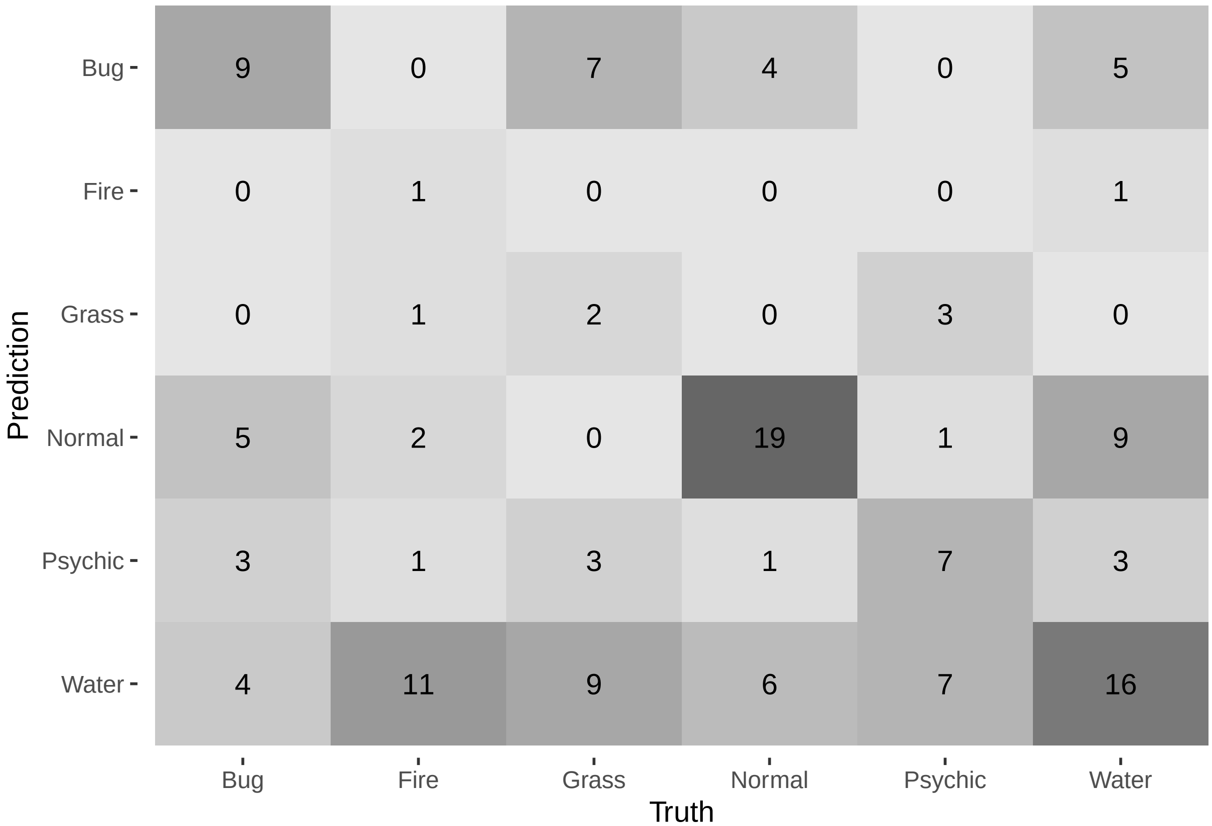
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.728
```

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

```
predicted_data %>% roc_curve(type_1, .pred_Bug:.pred_Water) %>%
  autoplot()
```




```
augment(pokemon_final_fit, new_data = pokemon_test) %>%
  conf_mat(truth = type_1, estimate =.pred_class)%>%
  autoplot("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?

The overall roc_auc is 0.70 which is not good enough. The model is best at predicting Normal type and water is the worst.This might due to the resample techniques.