PSTAT 131/231 Homework 5

William Long

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`.

```
pokemon <- read.csv("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_clean <- clean_names(pokemon)
```

A: The clean_names() function stripped all of the column names to consist of only underscores, numbers, and letters. The default arguments supplied also changed all the letters to be lowercase. For example, "Type.1" got changed to "type_1". This function helps keep variable names consistent which makes the dataframe more readable overall and makes coding with this dataset easier as you are less prone to making small mistakes when referencing the wrong variable name.

Exercise 2

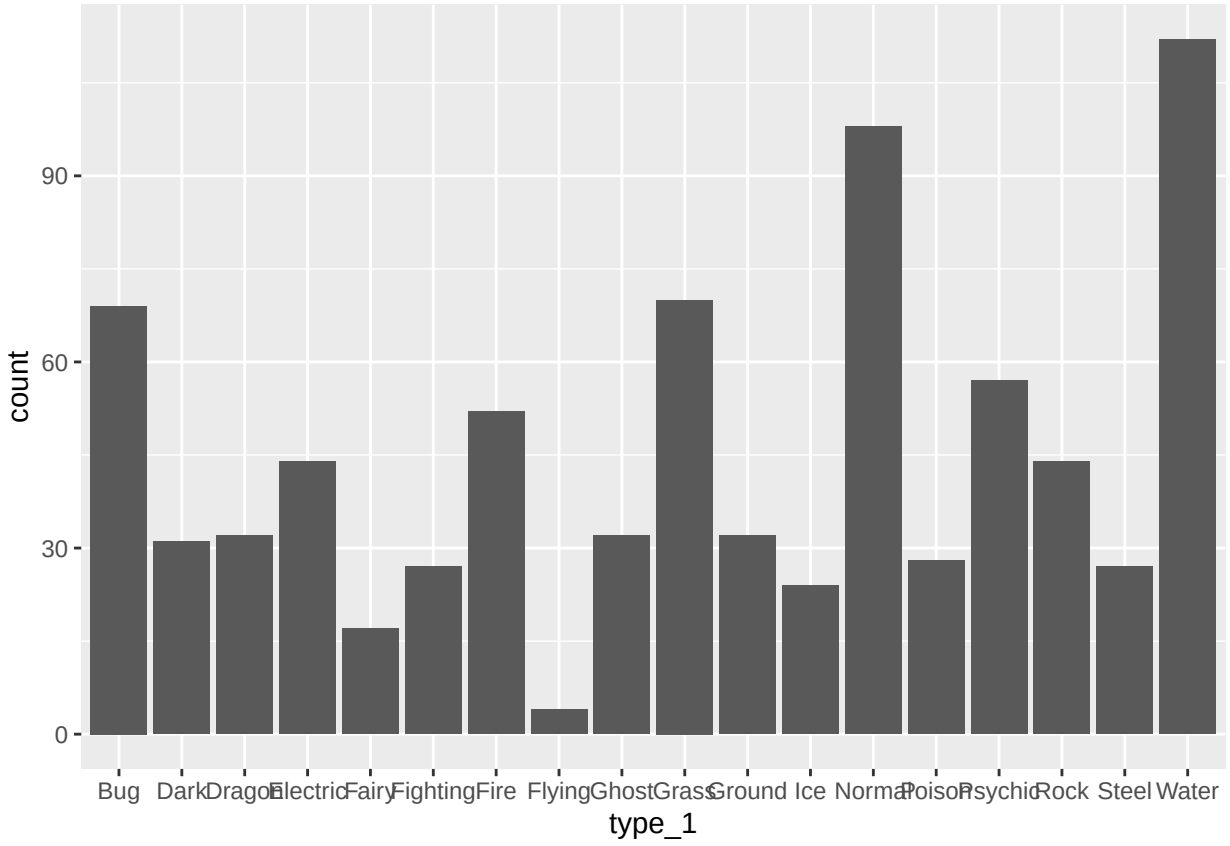
Using the entire data set, create a bar chart of the outcome variable, `type_1`.

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

```
#Bar chart of "type_1" with all types
pokemon_clean %>%
  ggplot(aes(x=type_1)) + geom_bar()
```




```
#Filtering out rarer types
pokemon_filter <- pokemon_clean %>%
  filter(type_1 == "Bug" | type_1 == "Fire" | type_1 == "Grass" | type_1 == "Normal" |
           type_1 == "Water" | type_1 == "Psychic")


#Converting "type_1" , "legendary", and "generation" to factors
pokemon_filter$type_1 <- factor(pokemon_filter$type_1)
pokemon_filter$legendary <- factor(pokemon_filter$legendary)
pokemon_filter$generation <- factor(pokemon_filter$generation)
```

A: There are 18 classes of "type_1" with Flying type as the most uncommon with only 4 Pokemon. Every other type seems to have at least 10+ Pokemon in its group but Fairy, Fighting, Ice, Poison, and Steel have less than 30 Pokemon as well.

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(2012)
```

```
pokemon_split <- initial_split(pokemon_filter, prop = 0.7, strata = type_1)
pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)

dim(pokemon_filter)  #458 total observations after filtering
```

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

```
dim(pokemon_train)   #458(0.7) = 320.6
```

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

```
dim(pokemon_test)    #458(0.3) = 137.4
```

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

```
#Actual total dimensions: 318+140 = 458
#The training and testing data have approximately the desired number of observations. The difference be

#Creating folds on the training set
pokemon_folds <- vfold_cv(pokemon_train, v=5, strata = type_1)
```

If we don't stratify our folds, it is possible that our folds can be randomly skewed in outcome variable distribution even though our original training and testing splits were stratified equally.

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_
  step_dummy(legendary) %>%    #Dummy-coding categorical predictors
  step_dummy(generation) %>%
  step_normalize(all_predictors()) #Centering and scaling
```

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(mode = "classification", engine = "glmnet", penalty = tune(),
                            mixture = tune())

pokemon_wkflow <- 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)
```

Since I'm using 5 folds with 10 levels each of penalty and mixture, $5(10)(10) = 500$ total models that I'll be fitting.

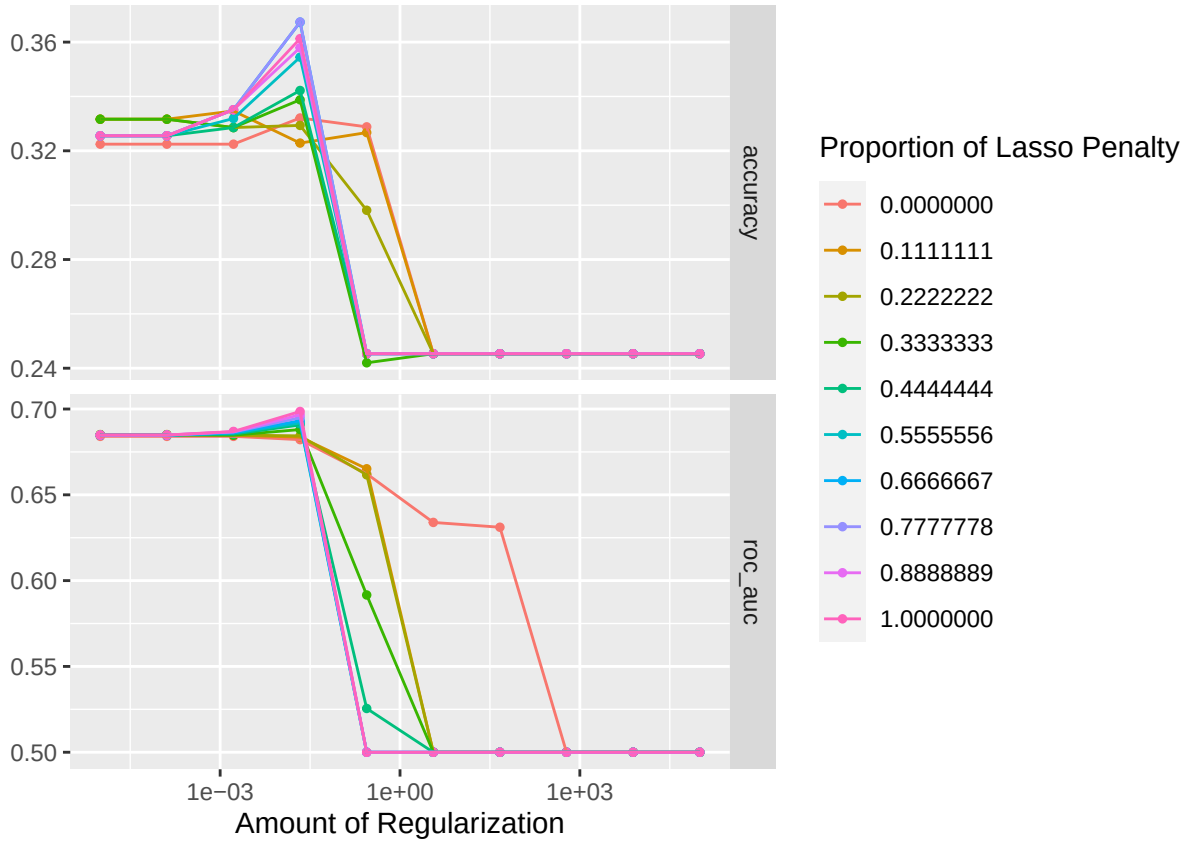
Exercise 6

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?

```
results <- tune_grid(pokemon_wkflow, resamples = pokemon_folds, grid = pokemon_grid)

autoplot(results)
```



Larger values of penalty(>0.555) and lower values of mixture(<0.3) tended to produce the highest 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(results, metric = "roc_auc")

final_model <- finalize_workflow(pokemon_wkflow, best)

final_fit <- fit(final_model, data = pokemon_train)

augment(final_fit, new_data = pokemon_test) %>%
  roc_curve(truth = type_1, estimate = c(.pred_Bug,.pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic
```

```
## # A tibble: 852 x 4
##    .level .threshold specificity sensitivity
##    <chr>       <dbl>       <dbl>       <dbl>
##  1 Bug    -Inf           0                   1
##  2 Bug       0.00275     0                   1
##  3 Bug       0.00371     0.00840             1
##  4 Bug       0.00791     0.0168              1
##  5 Bug       0.0129      0.0252              1
##  6 Bug       0.0133      0.0336              1
##  7 Bug       0.0164      0.0420              1
##  8 Bug       0.0174      0.0504              1
##  9 Bug       0.0190      0.0588              1
## 10 Bug       0.0253      0.0672              1
## # ... with 842 more rows
```

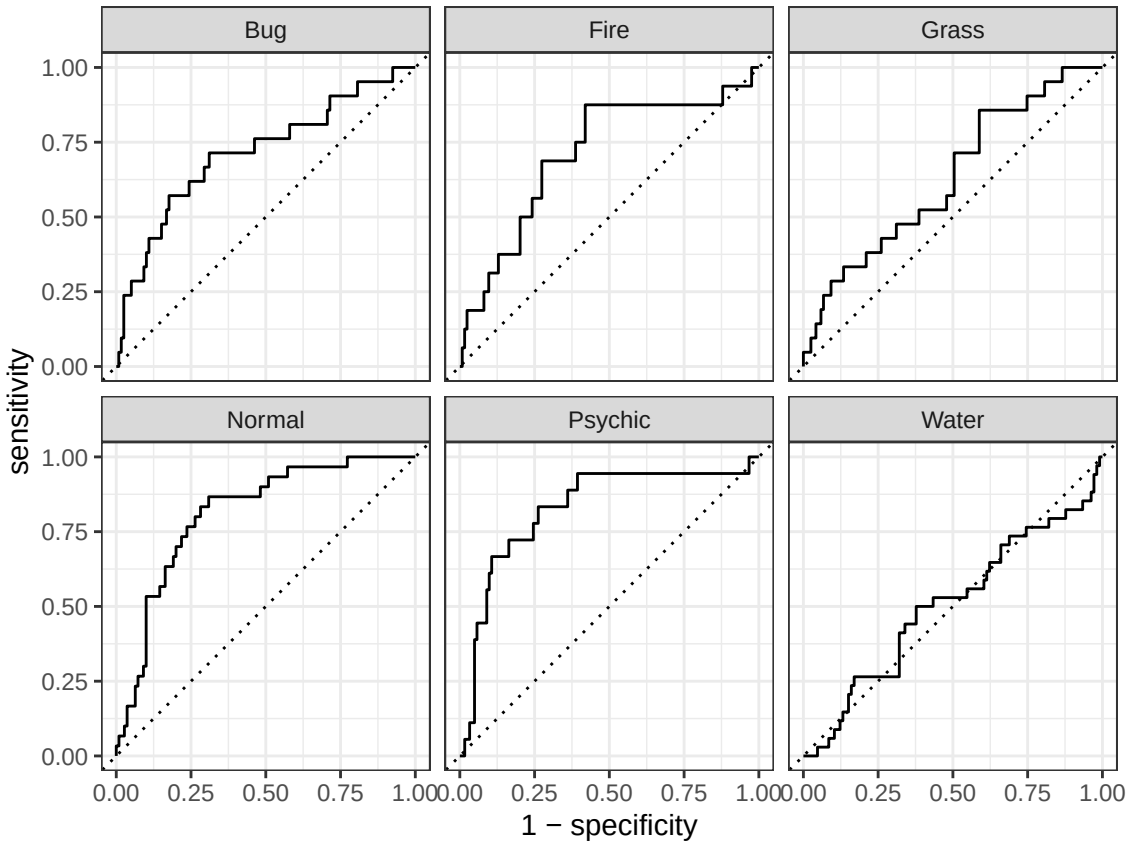
Exercise 8

Calculate the overall ROC AUC on the testing set.

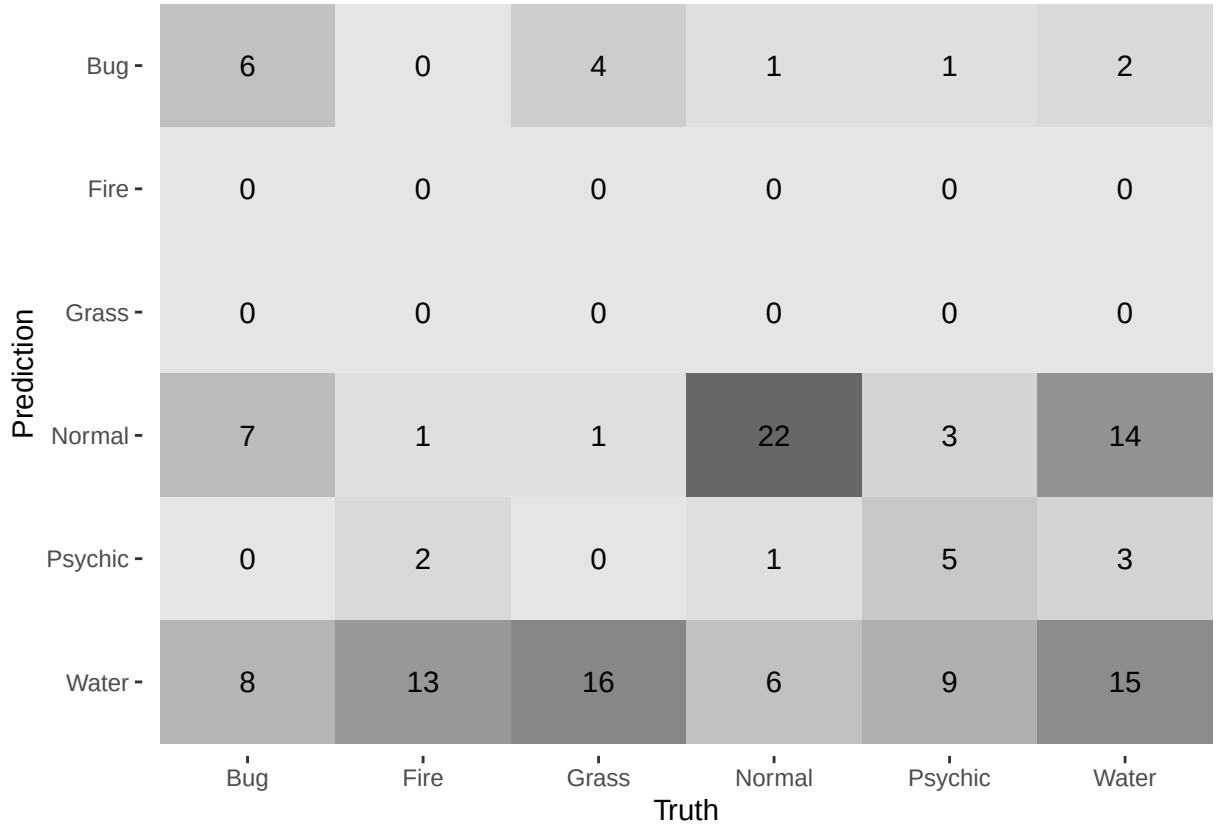
Then create plots of the different ROC curves, one per level of the outcome. Also make a heat map of the confusion matrix.

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?

```
#Plotting all the different ROC curves
augment(final_fit, new_data = pokemon_test) %>%
  roc_curve(truth = type_1, estimate = c(.pred_Bug,.pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic
  autoplot()
```




```
#Heat map of confusion matrix
augment(final_fit, new_data = pokemon_test) %>%
  conf_mat(truth = type_1, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```



My model was pretty good at predicting some types such as "Normal" and "Psychic" but had sub-par performance for most other types. For example, predicting a "Water" type was no better than random chance. Given my previous knowledge of the games, most Pokemon types do not tend to follow specific stat distribution patterns, but there are a few exceptions. I know that Psychic Pokemon tend to have high "sp_atk" and "sp_def" which is likely why the model was able to predict them better than most of the other types.

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.