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(tidyverse)
library(tidymodels)
library(ggplot2)
library(tune)
```

```
library(glmnet)
library(yardstick)
tidymodels_prefer()
```

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/finnianstack/Desktop/Pokemon.csv")
```

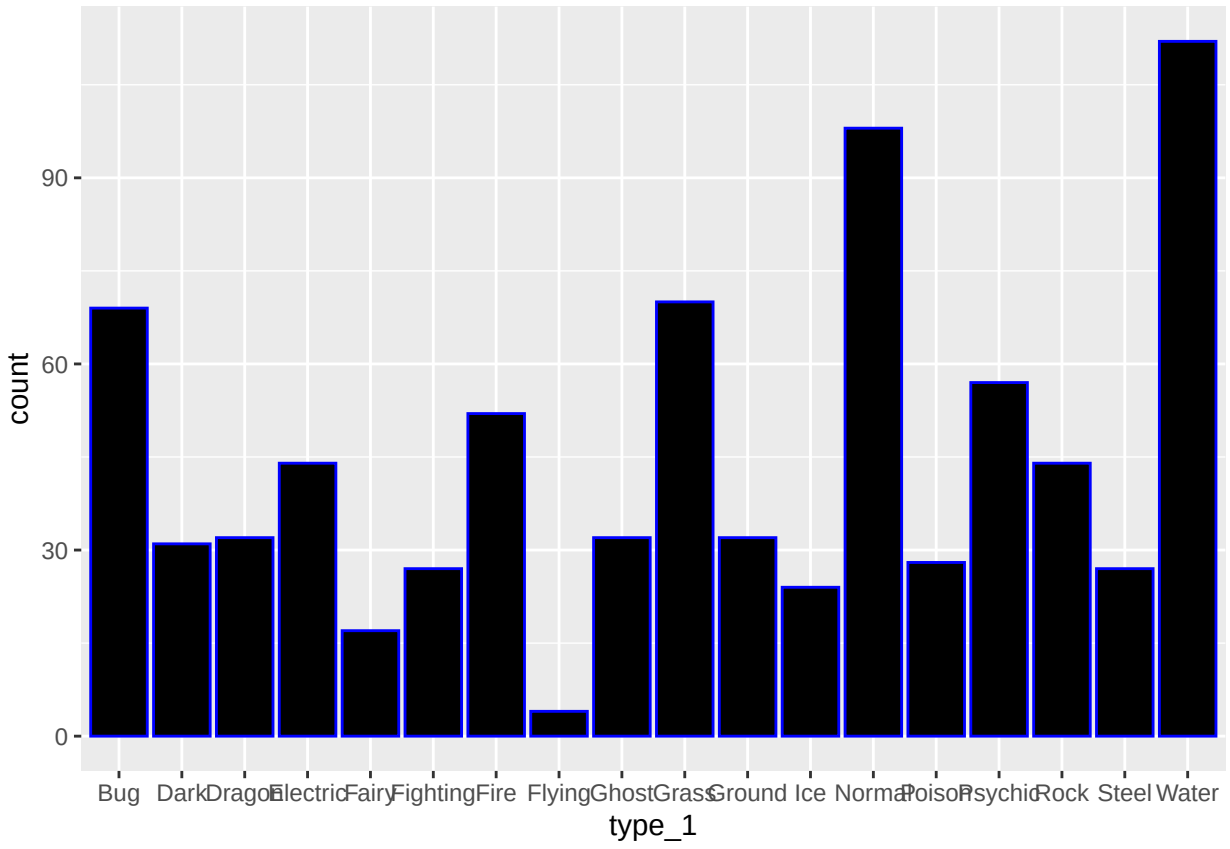
```
library(janitor)
pokemon <- clean_names(pokemon)
```

Using clean_names() changed the variables to lowercase and added under scores. Using this function is useful because it standardizes the data.

Exercise 2

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

```
library(ggplot2)
ggplot(pokemon) + geom_bar(aes(x = type_1), col = 'blue', fill = "black")
```



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

There are 18 classes of outcome. The type with the fewest pokemon is the flying class and second is the fairy type.

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(type_1 == "Bug" |
                                type_1 == "Fire" |
                                type_1 == "Grass" |
                                type_1 == "Normal" |
                                type_1 == "Water" |
                                type_1 == "Psychic")
```

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

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

```
set.seed(11)
pokemon_split <- initial_split(pokemon, prop = 0.8, strata = type_1)
pokemon_training <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)
nrow(pokemon_training)
```

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

```
nrow(pokemon_test)
```

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

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

Since there are different amounts of pokemon in each outcome, by stratifying the folds, this will preserve the original ratios of the classes into each fold.

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) %>%
  step_dummy(legendary, generation) %>%
  step_center(starts_with("legendary"), starts_with("generation"), sp_atk, attack, speed,
              defense, hp, sp_def) %>%
  step_scale(starts_with("legendary"), starts_with("generation"), sp_atk, attack, speed,
             defense, hp, sp_def)
```

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).

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

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

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

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

We have 100 total models

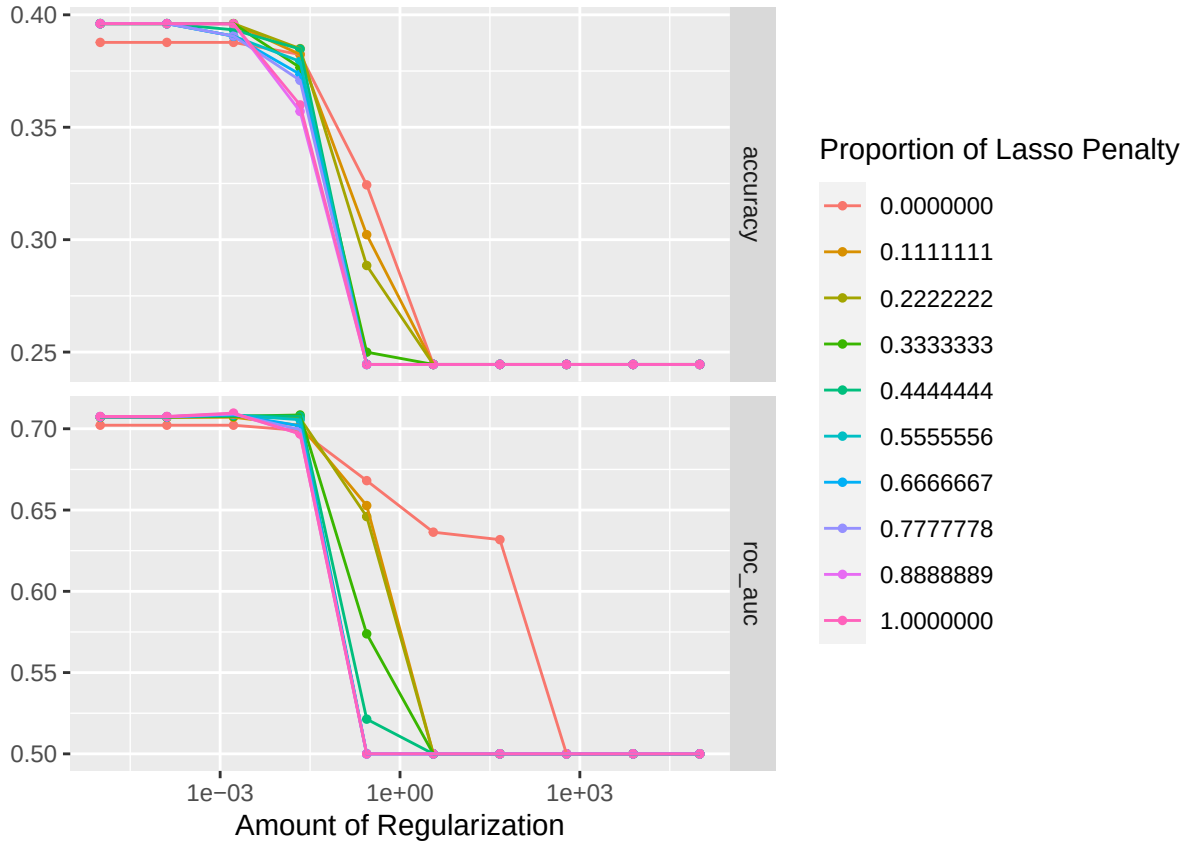
Exercise 6

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

```
tune_pokemon <- tune_grid(pokemon_workflow,
                          resamples = pokemon_folds,
                          grid = elastic_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?

```
autoplot(tune_pokemon)
```



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.

```
lowest_auc <- select_best(tune_pokemon, metric = "roc_auc")

final_workflow <- finalize_workflow(pokemon_workflow, lowest_auc)

pokemon_fit <- fit(final_workflow, pokemon_training)

prediction_result <- augment(pokemon_fit, new_data = pokemon_test) %>%
  accuracy(truth = type_1, estimate = .pred_class)
```

```
pokemon_final <- finalize_workflow(pokemon_workflow, lowest_auc)
pokemon_final_fit <- fit(pokemon_final, data = pokemon_training)
```

```
prediction_result <- augment(pokemon_final_fit, new_data = pokemon_test) %>%
    select(type_1, .pred_class, .pred_Bug, .pred_Fire, .pred_Grass,
           .pred_Normal, .pred_Psychic, .pred_Water)
accuracy(prediction_result, type_1, .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
```

```
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.340
```

Exercise 8

Calculate the overall ROC AUC on the testing set.

```
roc_aunp(prediction_result, type_1, .pred_Bug:.pred_Water)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 roc_aunp macro          0.679
```

```
roc_auc(prediction_result, type_1, .pred_Bug, .pred_Fire, .pred_Grass,
        .pred_Normal, .pred_Psychic, .pred_Water, estimator = "macro_weighted")
```

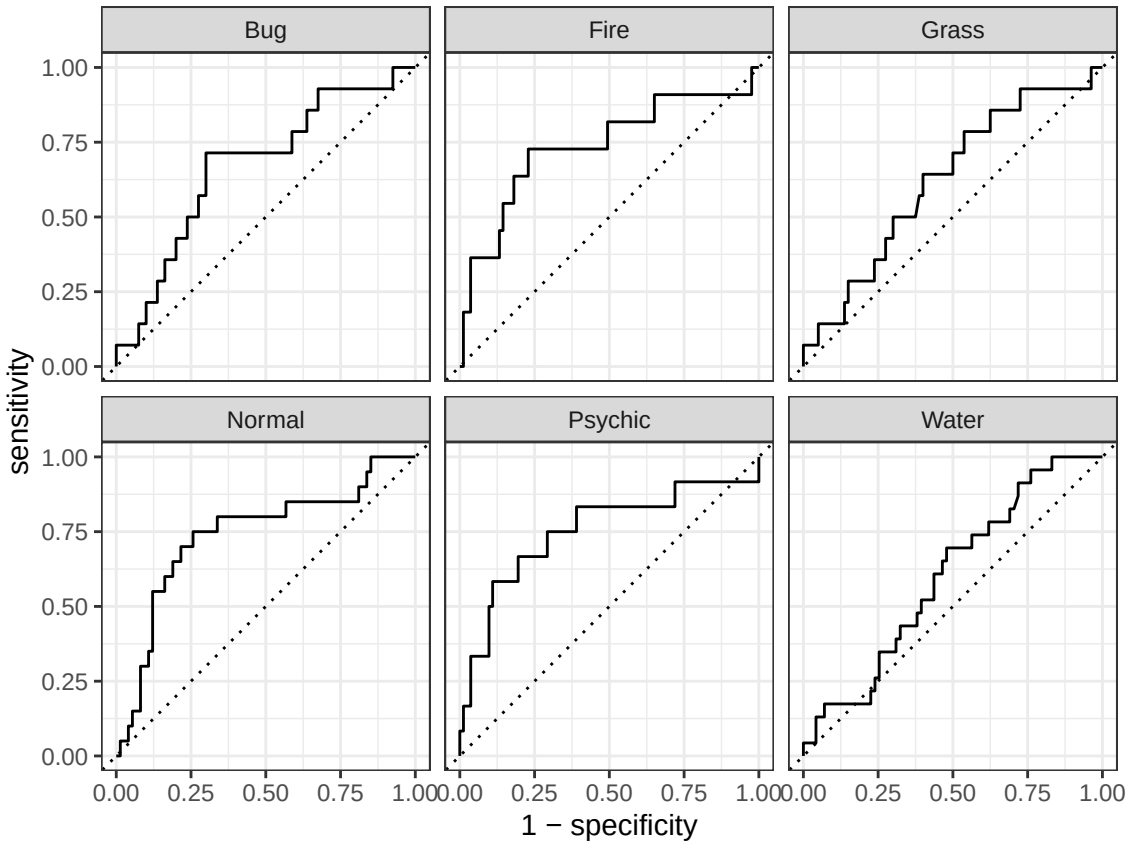
```
## # A tibble: 1 x 3
##   .metric .estimator     .estimate
##   <chr>   <chr>              <dbl>
## 1 roc_auc macro_weighted     0.679
```

```
roc_auc(prediction_result, type_1, .pred_Bug, .pred_Fire, .pred_Grass,
        .pred_Normal, .pred_Psychic, .pred_Water, estimator = "macro_weighted")
```

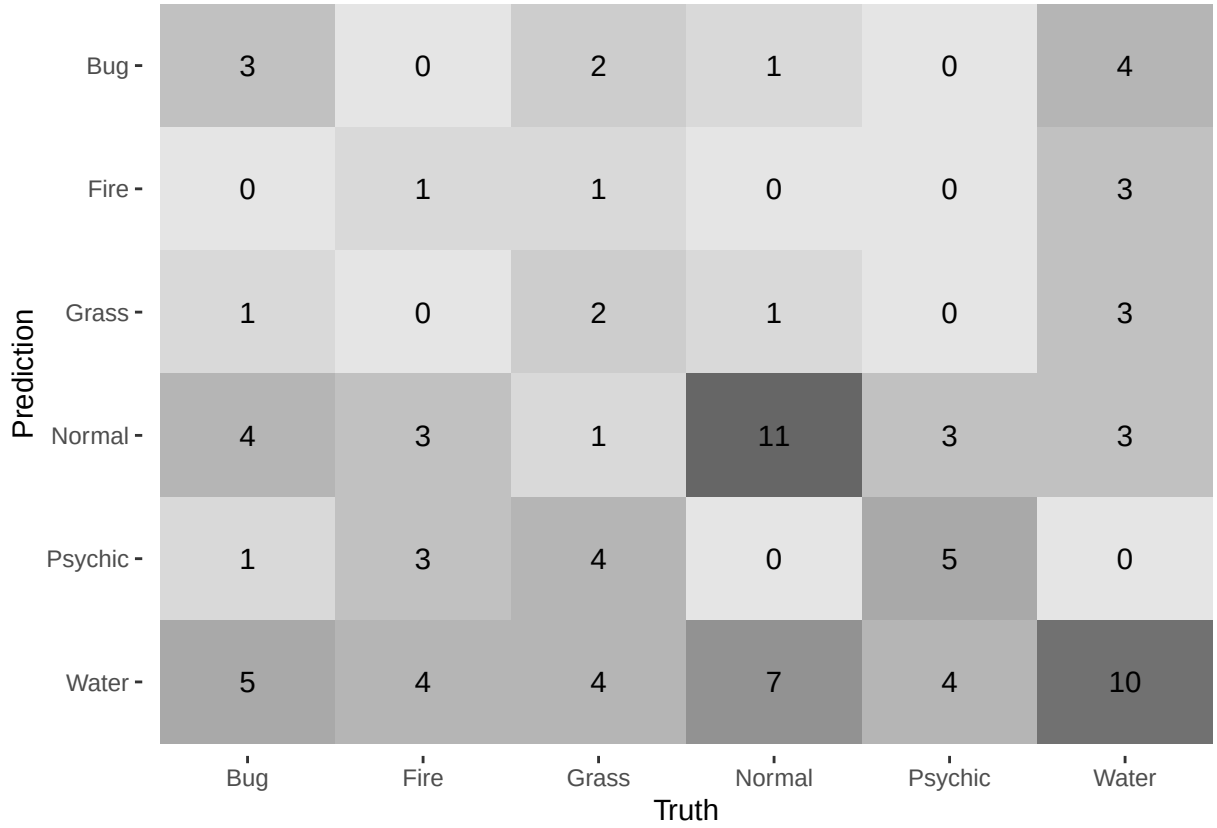
```
## # A tibble: 1 x 3
##   .metric .estimator     .estimate
##   <chr>   <chr>              <dbl>
## 1 roc_auc macro_weighted     0.679
```

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

```
prediction_result %>%
  roc_curve(type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal,
            .pred_Psychic, .pred_Water) %>%
  autoplot()
```




```
prediction_result %>%
  conf_mat(type_1, .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?

The accuracy and roc_auc are not very high, so the model is generally not doing very well. However, since the pokemon dataset is multiclass, the data set is actually doing reasonably well.

Our psychic pokemon model appears to be the best at predicting the model and the normal type also worked pretty well. The Fire type model performed the poorest at predicting the variables. This could be due to the imbalanced dataset. Some classes have much more observations than the other type which could cause variation in the prediction models.

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.