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(glmnet)
```

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 <- read_csv("data/Pokemon.csv") %>%
 clean_names()
```

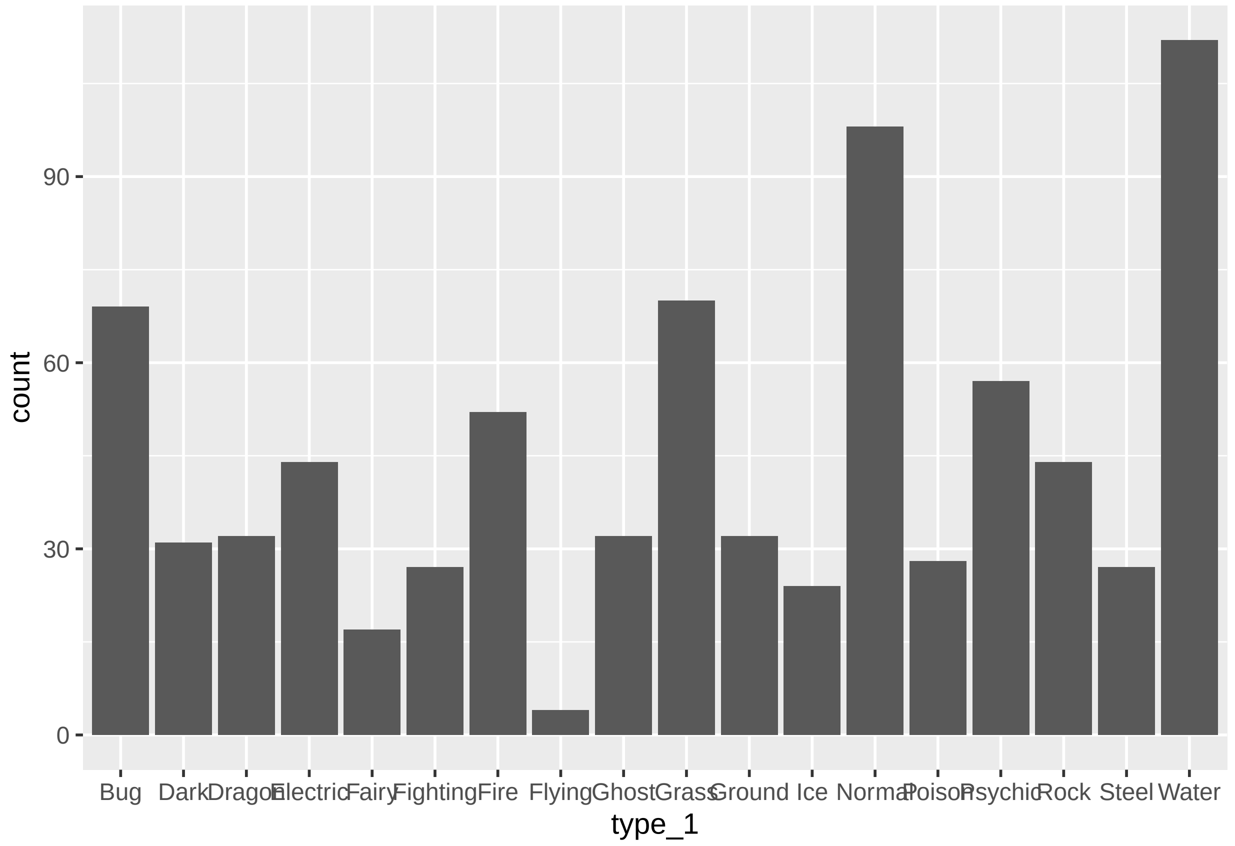
`clean_names()` make all our variable names consistent in one line of code. And it automatically converts all our variable names to lower case and puts underscores between the gaps (snake case). It's useful when we are dealing with large amounts of variables with various naming styles.

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?

```
pokemon %>%
 ggplot(aes(x = type_1)) +
 geom_bar()
```



There are eighteen classes of the outcome in the data. The Flying and Fairy classes have the smallest numbers of 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 <- pokemon %>%
 filter(type_1 %in% c('Bug', 'Fire', 'Grass', 'Normal', 'Water', 'Psychic'))
```

```
pokemon <- pokemon %>%
 mutate(type_1 = factor(type_1), legendary = factor(legendary))
```

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(2727)
pokemon_split <- initial_split(pokemon, prop = 0.7, strata = type_1)
pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)
dim(pokemon_train)
```

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

```
dim(pokemon_test)
```

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

When the data set is relatively small (800 observations), 70:30 is suitable. More observations are in the training set than in the testing set. There are 318 observations in the training set, which is around 70% of the 458 observations in the Pokemon_split.

```
# 5 fold cross-validation
set.seed(2772)
pokemon_folds <- vfold_cv(data = pokemon_train, v = 5, strata = type_1)
```

Because it generally results in a more stable and less biased estimate of the model. It makes sure that the distribution remains the same across the folds.

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
pokemon_recipe <- recipe(type_1 ~ legendary + generation +
 sp_atk + attack + speed + defense +
 hp + sp_def, data = pokemon_train) %>%
 step_dummy(all_nominal_predictors()) %>%
 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).

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

```r
elastic_net_spec <- multinom_reg(penalty = tune(),
 mixture = tune()) %>%
 set_mode("classification") %>%
 set_engine("glmnet")
en_workflow <- workflow() %>%
 add_recipe(pokemon_recipe) %>%
 add_model(elastic_net_spec)
en_grid <- grid_regular(penalty(range = c(-5, 5)),
 mixture(range = c(0, 1)), levels = 10)
```

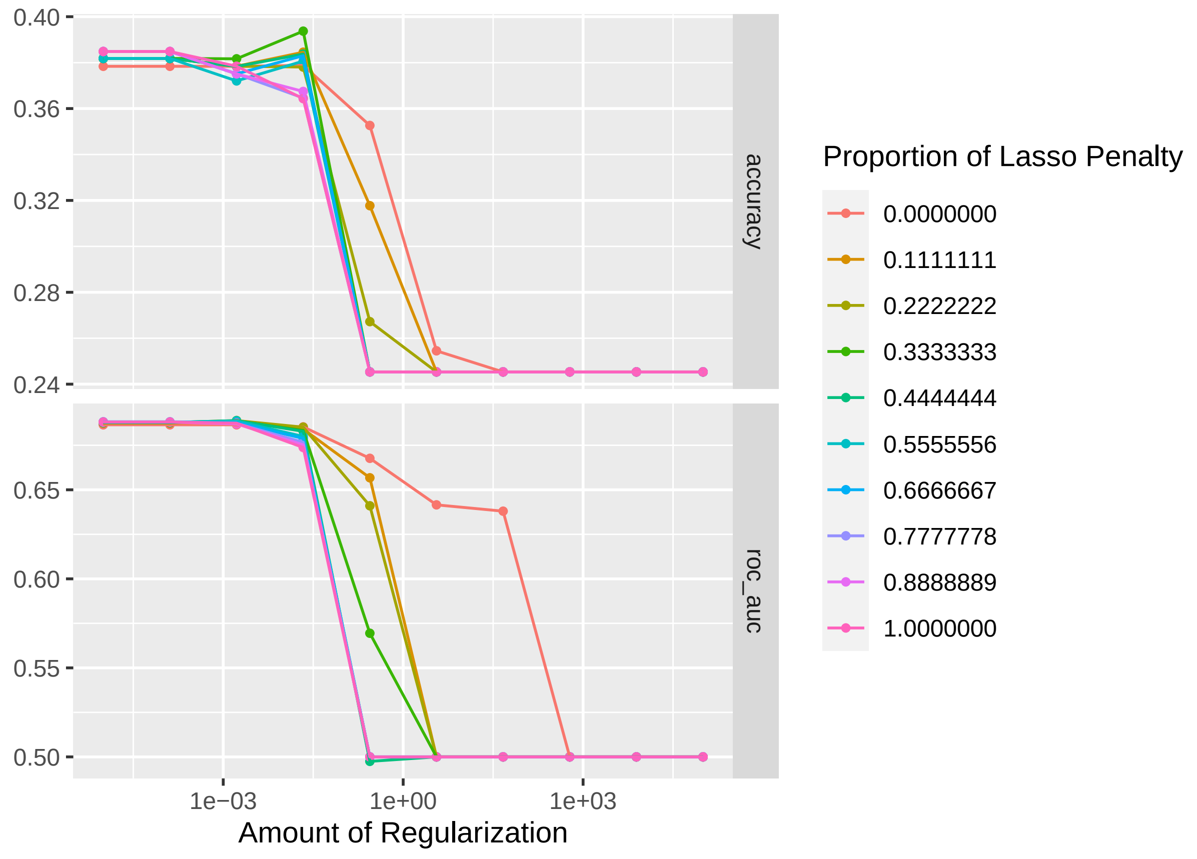
We will fit 100 models per fold, for a total of 500 models.

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?

```r
tune_res <- tune_grid(
 en_workflow,
 resamples = pokemon_folds,
 grid = en_grid
)
autoplot(tune_res)
```



We can find that smaller values of penalty (proportion of lasso penalty) and smaller values of mixture (amount of regularization) will result in larger higher accuracy and ROC AUC values.

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")
en_final <- finalize_workflow(en_workflow, best_model)
en_final_fit <- fit(en_final, data = pokemon_train)
predicted_data <- augment(en_final_fit, new_data = pokemon_test) %>%
 select(type_1, starts_with(".pred"))
```

Exercise 8

Calculate the overall ROC AUC on the testing set.

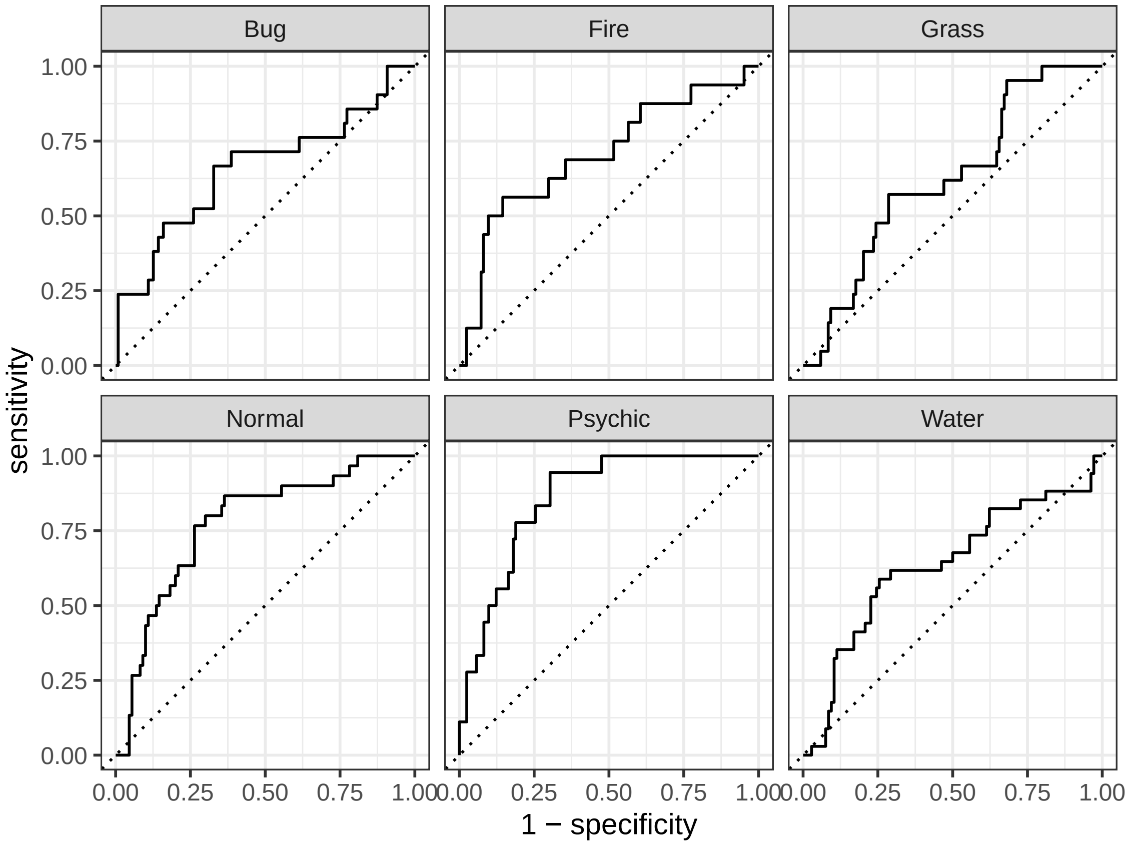
Then create plots of the different ROC curves, one per level of the outcome. Also make a heat map of the confusion matrix.

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?

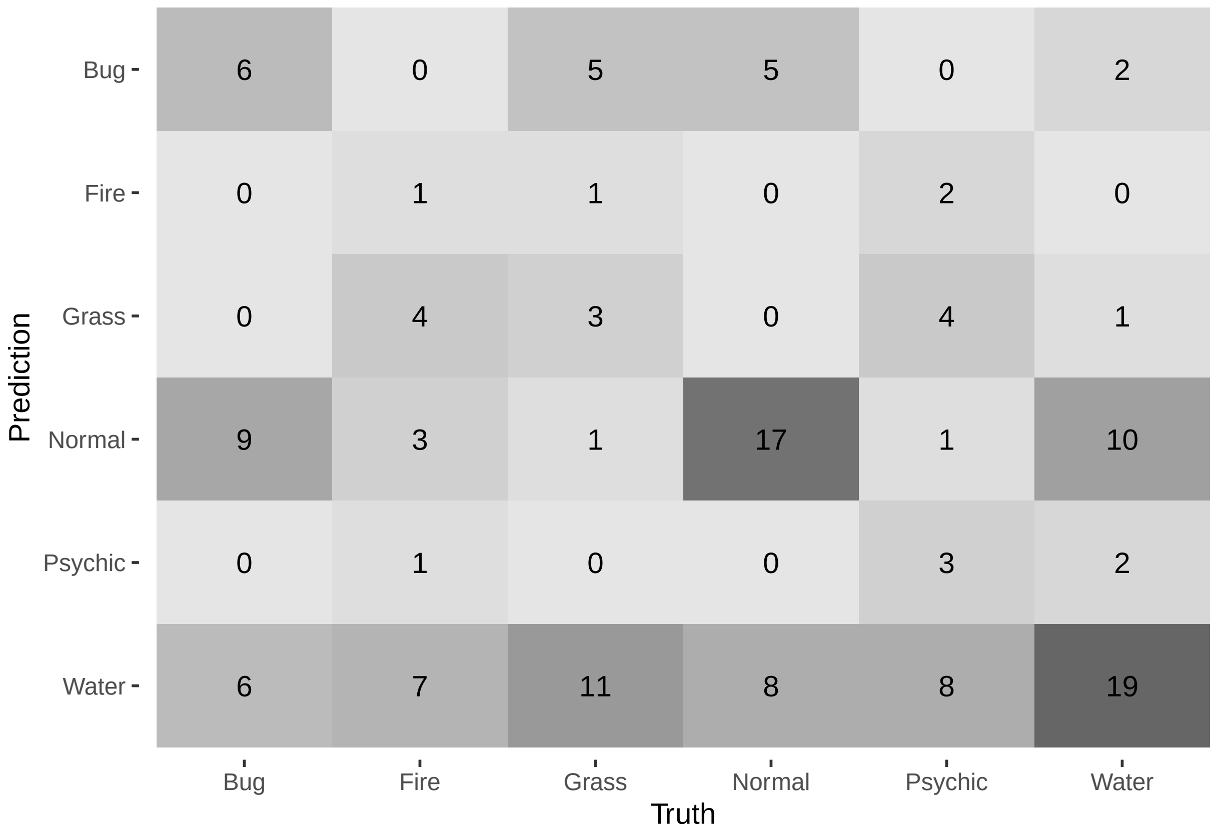
```
# AUC
predicted_data %>% roc_auc(type_1, .pred_Bug:.pred_Water)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.704
```

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



```
# heat map
predicted_data %>%
 conf_mat(truth = type_1, estimate = .pred_class) %>%
 autoplot(type = "heatmap")
```



Prediction \ Truth	Bug	Fire	Grass	Normal	Psychic	Water
Bug	6	0	5	5	0	2
Fire	0	1	1	0	2	0
Grass	0	4	3	0	4	1
Normal	9	3	1	17	1	10
Psychic	0	1	0	0	3	2
Water	6	7	11	8	8	19

We can notice that the model is worst at differentiating. From the ROC curve we can see, the model is worst at identifying Grass and Water type. But, it is good at identifying Psychic and Normal type. From the heatmap, we can find that it correctly identified 17/30 of the Normal type and 19/34 of the Water type. However, the model seems worst at identify the Water type. It always identified other types as the Water type, especially the Grass type. It might because the data is cpmplex but the model is too simple.