Homework 5

PSTAT 131 John Wei

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.

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(discrim)
library(poissonreg)
library(ISLR)
library(ISLR2)
library(corrr)
library(klaR)
library(glmnet)
tidymodels_prefer()
set.seed(4167)
```

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)
```

```r
pokemon <- read_csv(file = "Pokemon.csv")
pokemon <- clean_names(pokemon)
pokemon <- as_tibble(pokemon)
pokemon
```

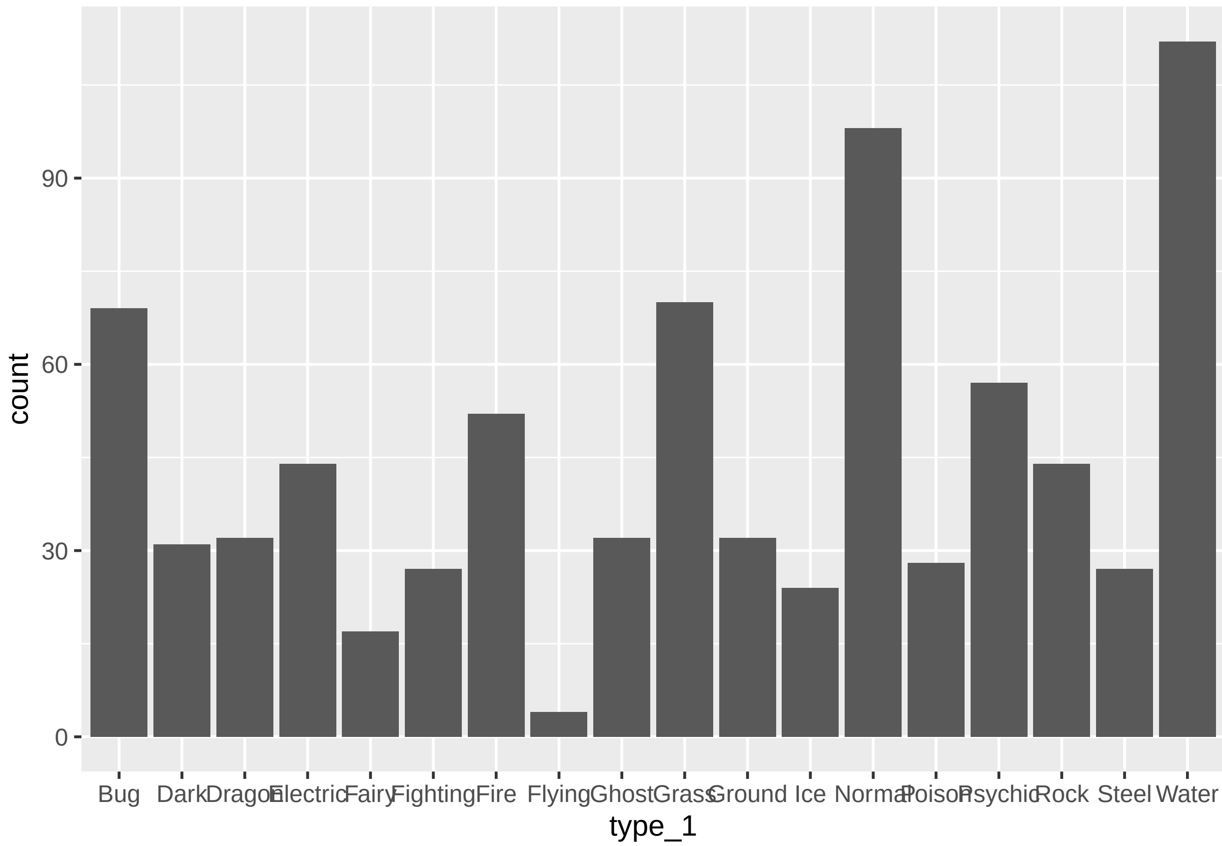
```
## # A tibble: 800 x 13
##    number name      type_1 type_2 total    hp attack defense sp_atk sp_def speed
##     <dbl> <chr>     <chr>  <chr>  <dbl> <dbl>  <dbl>   <dbl>  <dbl>  <dbl> <dbl>
##  1      1 Bulbasaur Grass  Poison   318    45     49      49     65     65    45
##  2      2 Ivysaur   Grass  Poison   405    60     62      63     80     80    60
##  3      3 Venusaur  Grass  Poison   525    80     82      83    100    100    80
##  4      3 Venusaur~ Grass  Poison   625    80    100     123    122    120    80
##  5      4 Charmand~ Fire   <NA>     309    39     52      43     60     50    65
##  6      5 Charmele~ Fire   <NA>     405    58     64      58     80     65    80
##  7      6 Charizard Fire   Flying   534    78     84      78    109     85   100
##  8      6 Charizar~ Fire   Dragon   634    78    130     111    130     85   100
##  9      6 Charizar~ Fire   Flying   634    78    104      78    159    115   100
## 10      7 Squirtle  Water  <NA>     314    44     48      65     50     64    43
## # ... with 790 more rows, and 2 more variables: generation <dbl>,
## #   legendary <lgl>
```

The function clean_names() is useful because it makes each variable name unique and standarizes them - it replaces spaces with underscores and changes the letter casing.

Exercise 2

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

```r
library(ggplot2)
ggplot(pokemon, aes(x = type_1)) + geom_bar()
```



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 the outcome, with flying type having very few Pokemon. The next fewest one is fairy.

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[pokemon$type_1 %in% c("Bug","Fire","Grass","Normal","Water","Psychic"), ]
```

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

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

```
pokemon_split <- initial_split(pokemon, prop = 0.8, strata = type_1)
pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)
```

```
dim(pokemon_train)
```

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

```
dim(pokemon_test)
```

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

364 + 94 = 458, which is the number of total observations in the filtered pokemon data.

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)
```

Stratifying the folds is useful because it keeps the ratios of the classes in 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_
  step_dummy(all_nominal_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_center(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?

```
elastic_spec <- multinom_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet") %>%
  set_mode("classification")
elastic_grid <- grid_regular(penalty(c(-5, 5)), mixture(c(0,1)), levels = 10)
elastic_workflow <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(elastic_spec)
```

100 models will be fitted to each fold, for a total of 500 total models.

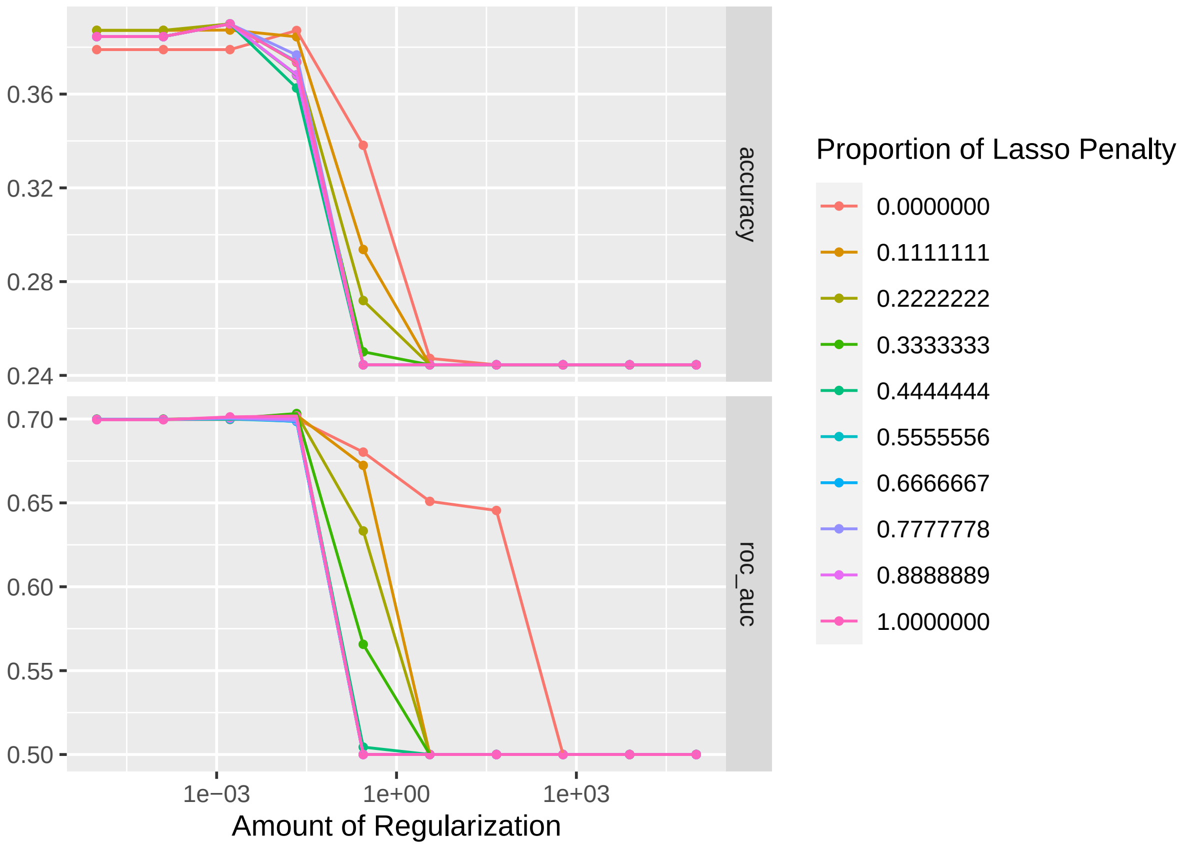
Exercise 6

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

```r
tune_res <- tune_grid(elastic_workflow, resamples = pokemon_fold, 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?

```r
autoplot(tune_res)
```



Smaller values of '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.

```r
pokemon_penalty <- select_best(tune_res, metric = "roc_auc")
```

```r
final = elastic_workflow %>%
  finalize_workflow(pokemon_penalty) %>%
  fit(pokemon_train) %>%
  augment(pokemon_test)
accuracy(final, truth = type_1, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.383
```

Exercise 8

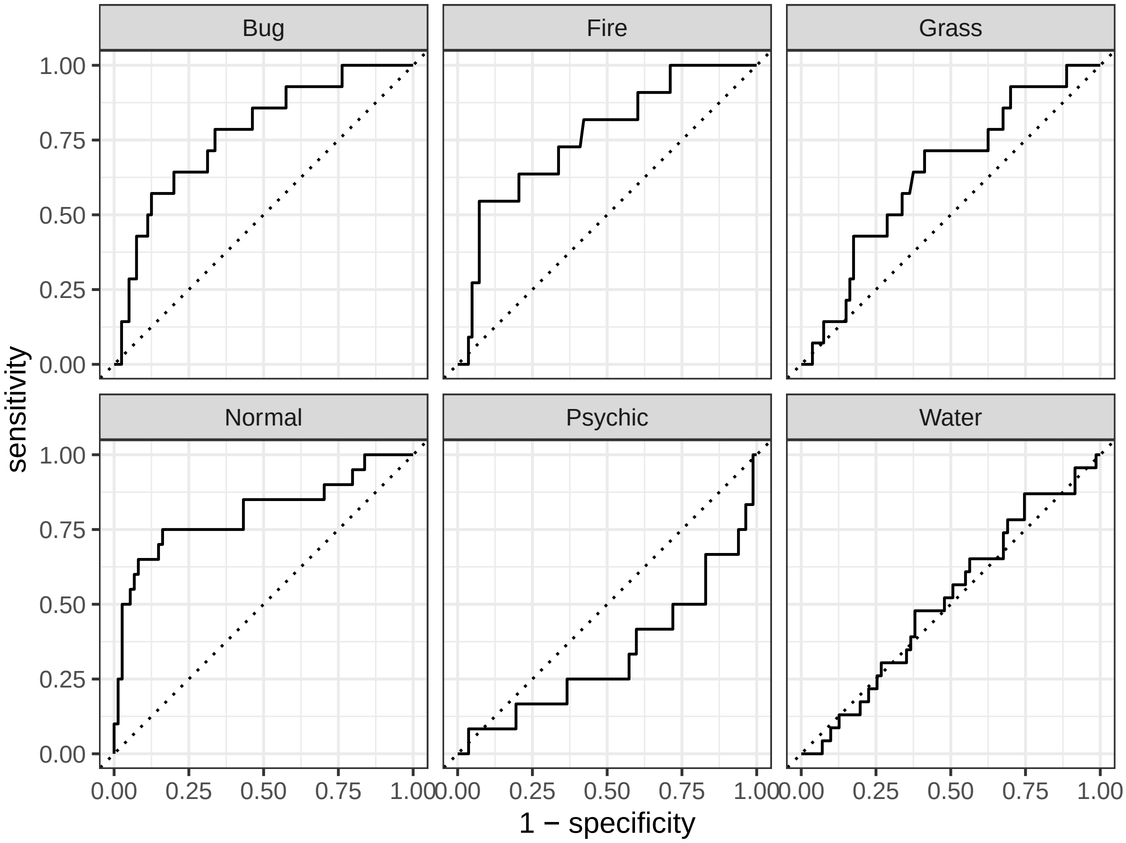
Calculate the overall ROC AUC on the testing set.

```
roc_auc(final, truth = type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Water, .pred_Psy
```

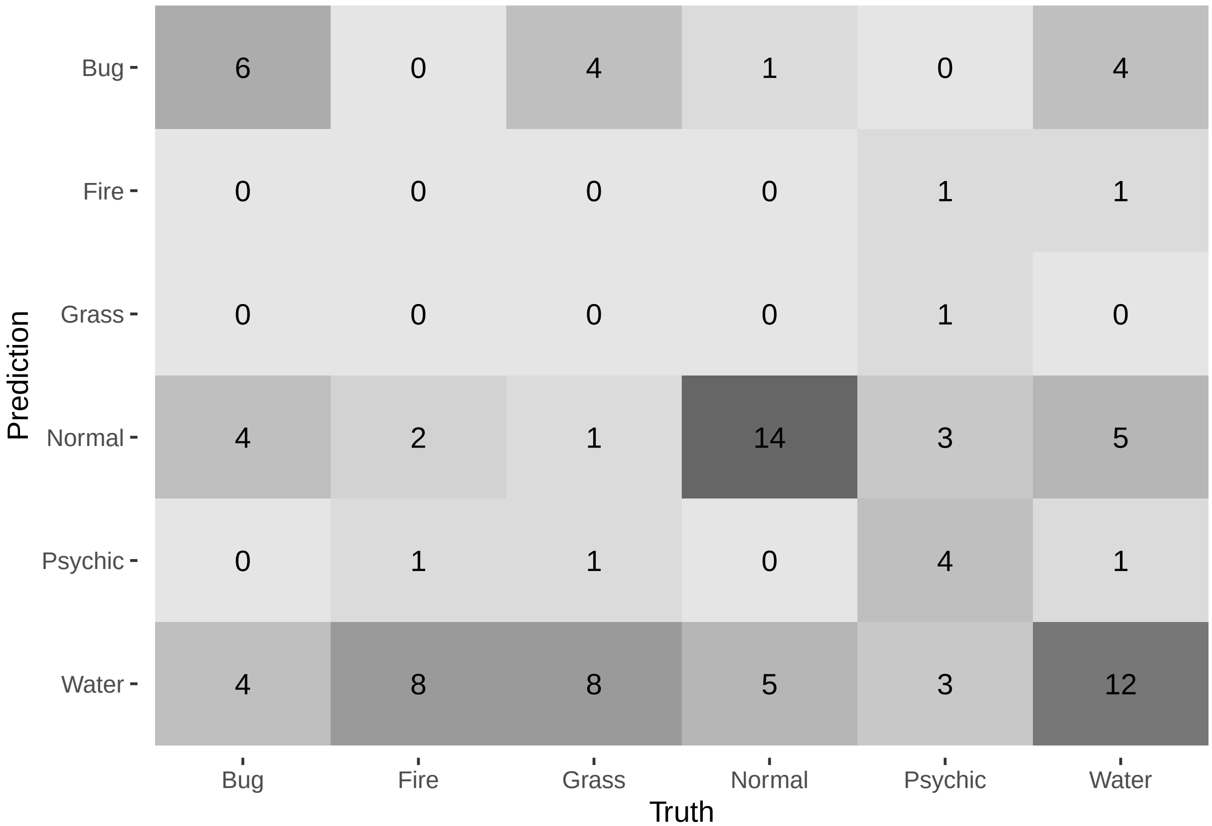
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.632
```

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

```
autoplot(roc_curve(final, truth = type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Water
```




```
conf_mat(final, 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?

The model didn't do well - an accuracy of .372 and ROC AUC of .577. Normal and fire types were the best at being predicted, with bug being an honorable mention. Grass did relatively mediocre, with water and especially psychic being the worst at being predicted. Water and bug pokemon were often classified as normal which makes sense since many of these types are "normal" (seals, bugs, fish etc) Water and psychic pokemon are the least predictable in my opinion because they tend to have a lot of variance of what they are.