PSTAT131_HW5

Immad Ali

5/15/2022

Elastic Net Tuning

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(ggplot2)
library(corrr)
library(klaR)
library(glmnet)
library(MASS)
library(discrim)
library(poissonreg)
tidymodels_prefer()
set.seed(999)
data <- read_csv("pokemon.csv")
data %>% head(5)
```

```
## # A tibble: 5 × 13
##     `#` Name    `Type 1` `Type 2` Total    HP Attack Defense `Sp. Atk` `Sp. Def`
##   <dbl> <chr>   <chr>    <chr>    <dbl> <dbl>  <dbl>   <dbl>     <dbl>     <dbl>
## 1     1 Bulbas… Grass    Poison     318    45     49      49        65        65
## 2     2 Ivysaur Grass    Poison     405    60     62      63        80        80
## 3     3 Venusa… Grass    Poison     525    80     82      83       100       100
## 4     3 Venusa… Grass    Poison     625    80    100     123       122       120
## 5     4 Charma… Fire     <NA>       309    39     52      43        60        50
## # … with 3 more variables: Speed <dbl>, Generation <dbl>, Legendary <lgl>
```

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?

```
#install.packages('janitor')
library(janitor)

data <- data %>%
  clean_names()
```

The clean_names() function works to tidy the names of all variables in the even there are special characters or repeating variables.

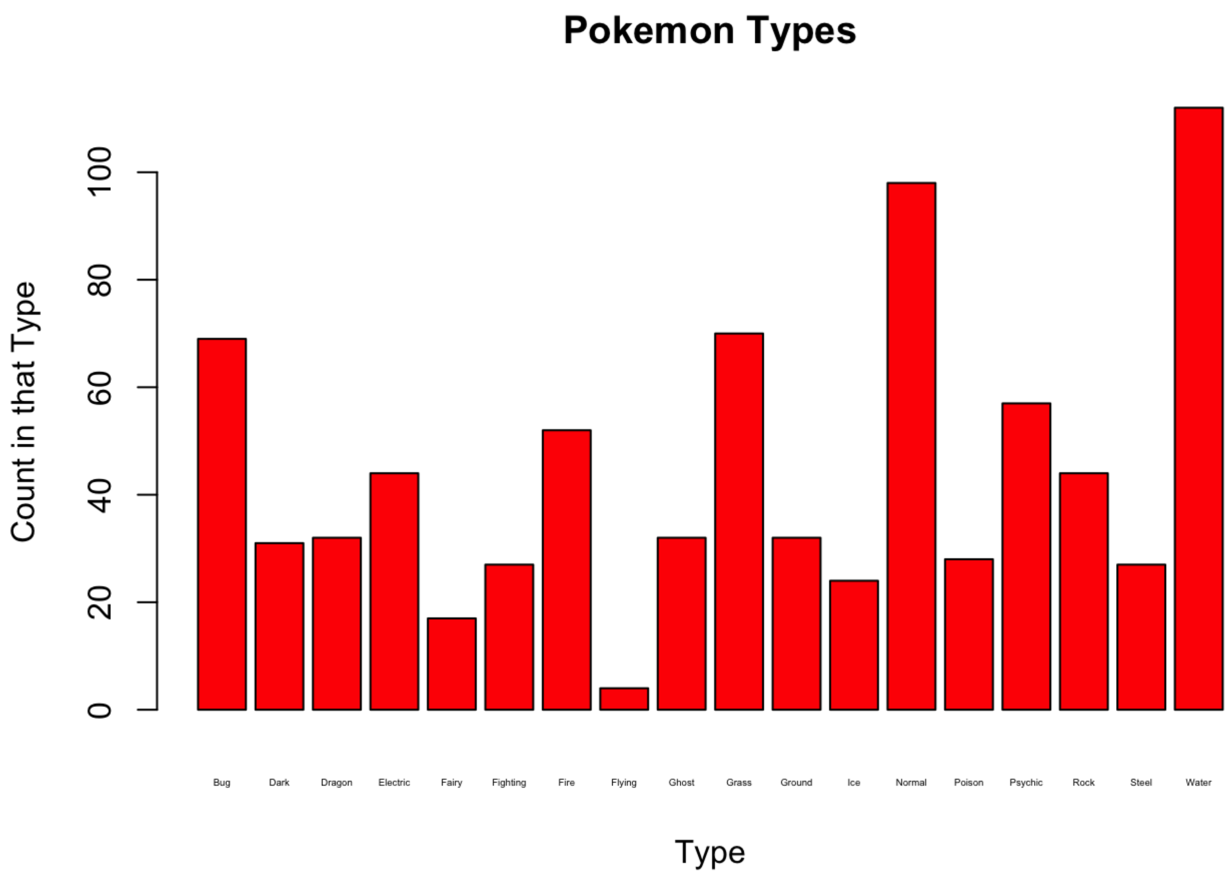
Exercise 2

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` and `legendary` to factors.

```
type_1 <- table(data$type_1)
type_1
```

```
##
##      Bug     Dark   Dragon Electric    Fairy Fighting     Fire   Flying
##       69       31       32       44       17       27       52        4
##    Ghost    Grass   Ground      Ice   Normal   Poison  Psychic     Rock
##       32       70       32       24       98       28       57       44
##    Steel    Water
##       27      112
```

```
barplot(type_1, xlab = "Type", ylab = "Count in that Type", main = "Pokemon Types", widt
h = 0.2, cex.names = 0.3, col = 'red')
```



From the graph, we can see that there are 18 different groups of pokemon types. Each of the groups have differing number of counts in each type, for example, in the flying types, there seems to be very few Pokemon compared to others, like normal.

```
#Selects data that only contains Pokemon types Bug, Fire, Grass, Normal, Water, Psychic
data <- data %>% filter((type_1 == "Bug" | type_1 == "Fire" | type_1 == "Grass" | type_1
== "Normal" | type_1 == "Water" | type_1 == "Psychic"))
```

```
#Converts to factors

data$type_1 <- as.factor(data$type_1)

data$generation <- as.factor(data$generation)

data$legendary <- as.factor(data$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?

```
# Splitting data set testing and training sets
data_split <- initial_split(data, prop = .8, strata = type_1)
train <- training(data_split)
test <- testing(data_split)

#Verifying data dimensions are the same still
dim(train)
```

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

```
dim(test)
```

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

```
folds <- vfold_cv(data = train, v = 4, strata = type_1)
folds
```

```
## #  4-fold cross-validation using stratification
## # A tibble: 4 × 2
##   splits           id
##   <list>           <chr>
## 1 <split [270/94]> Fold1
## 2 <split [273/91]> Fold2
## 3 <split [274/90]> Fold3
## 4 <split [275/89]> Fold4
```

It is important to stratify the folds as it makes sure there is a balanced distribution across the types, in other words, representative of the whole sample.

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.

```
recipe_pokemon <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + def
ense + hp + sp_def, data = train) %>%
  step_dummy(legendary) %>%
  step_dummy(generation) %>%
  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
#inital model with parameters tune()
pokemon_net <- multinom_reg(penalty = tune(), mixture = tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")
```

```r
#workflow with recipe and model
pokemon_wrkflow <- workflow() %>%
  add_recipe(recipe_pokemon) %>%
  add_model(pokemon_net)
```

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

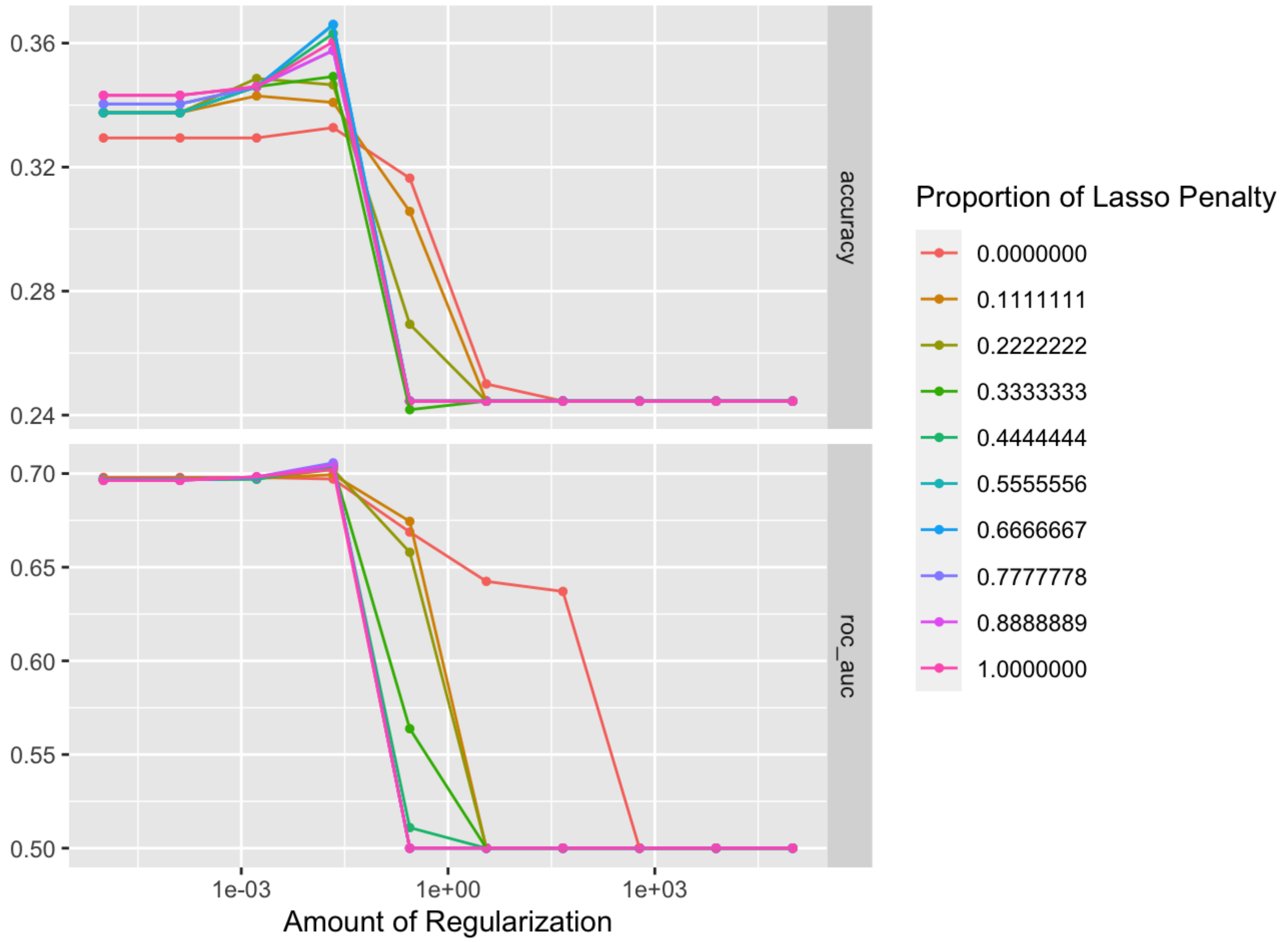
```
## # A tibble: 100 × 2
##          penalty mixture
##            <dbl>   <dbl>
##  1      0.00001        0
##  2      0.000129       0
##  3      0.00167        0
##  4      0.0215         0
##  5      0.278          0
##  6      3.59           0
##  7     46.4            0
##  8    599.             0
##  9   7743.             0
## 10 100000              0
## # … with 90 more rows
```

We will have a total of 500 models as there are 100 * 5.

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
pokemone_tunegrid <- tune_grid(object = pokemon_wrkflow, resamples = folds, grid = penalty_grid)
autoplot(pokemone_tunegrid)
```



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.

```
#model selection
best_penalty <- select_best(pokemone_tunegrid, metric = "roc_auc")
best_penalty
```

```
## # A tibble: 1 × 3
##   penalty mixture .config
##     <dbl>   <dbl> <chr>
## 1  0.0215   0.778 Preprocessor1_Model074
```

```
# finalizing workflow and fitting best model on the training set
pokemon_final <- finalize_workflow(pokemon_wrkflow, best_penalty)
pokemon_final_fit <- fit(pokemon_final, data = train)
```

```
# evaluating best model on the test set
final_model_accuracy <- augment(pokemon_final_fit, new_data = test) %>%
  accuracy(truth = type_1, estimate = .pred_class)
final_model_accuracy
```

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.394
```

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?

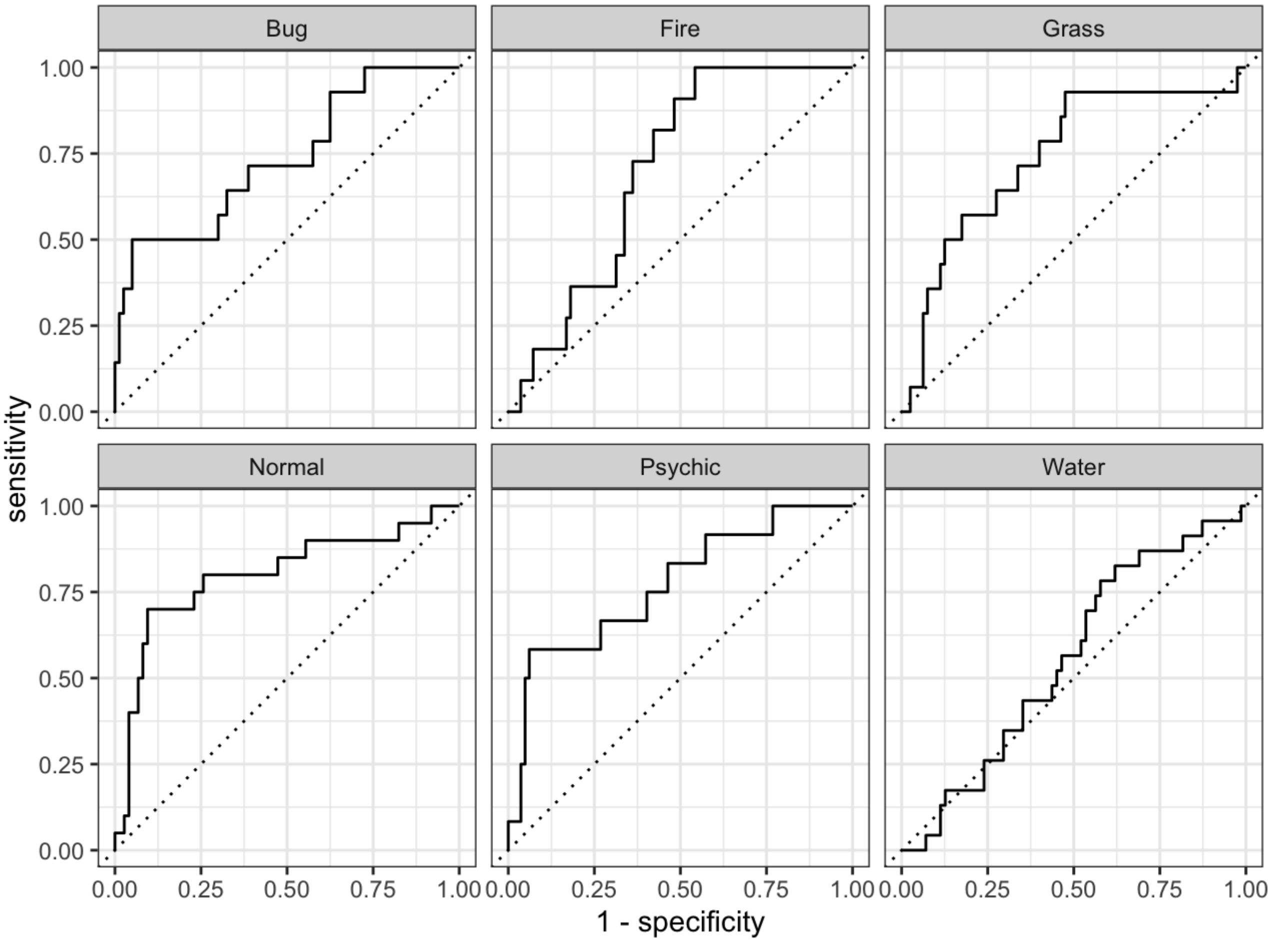
```
totalroc_auc <- augment(pokemon_final_fit, new_data = test) %>%
  roc_auc(truth = type_1, estimate = c(.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal,
.pred_Psychic, .pred_Water))

totalroc_auc
```

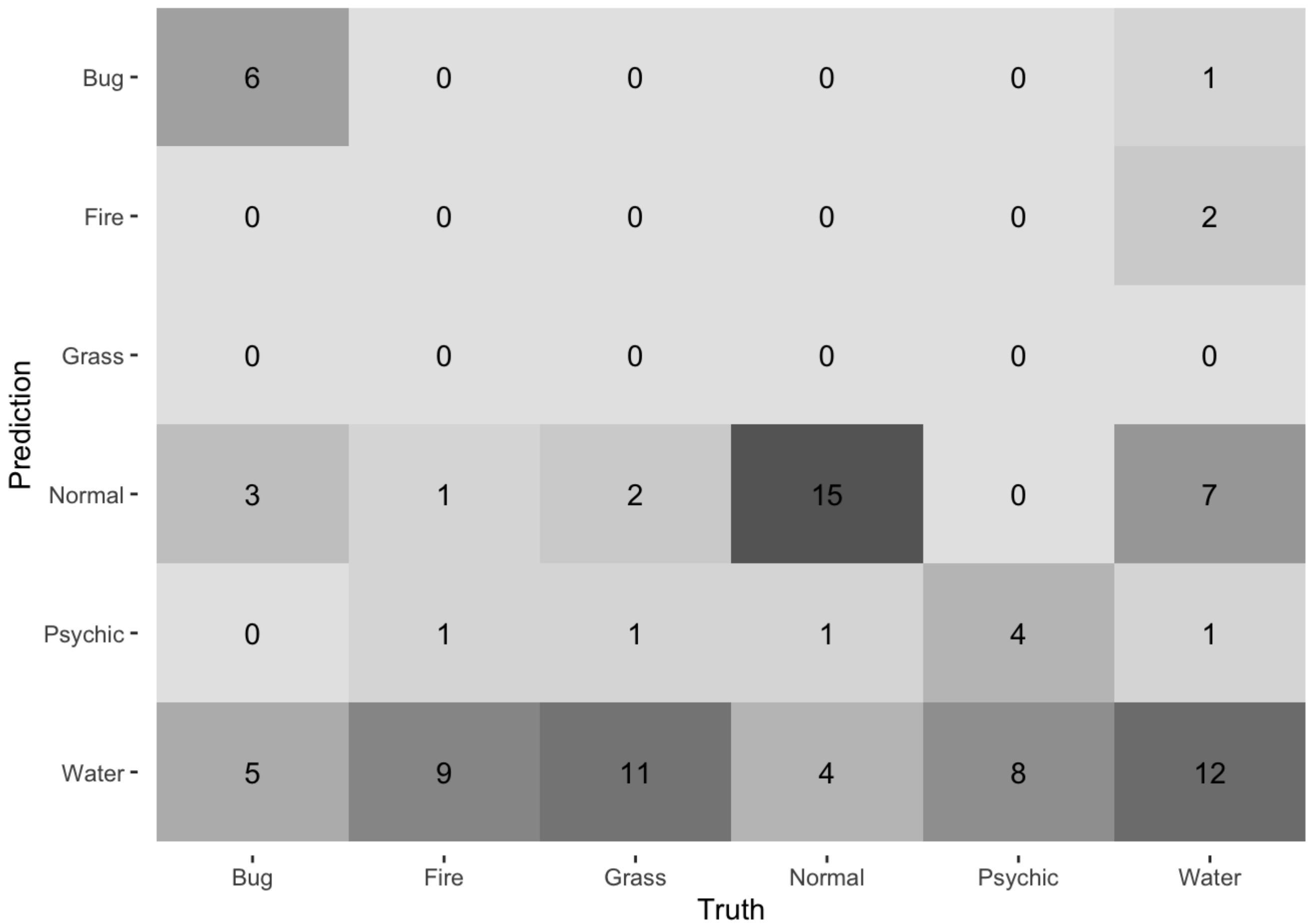
```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.712
```

```
roc_curves <- augment(pokemon_final_fit, new_data = test) %>%
  roc_curve(truth = type_1, estimate = c(.pred_Bug, .pred_Fire, .pred_Grass, .pred_Norma
l, .pred_Psychic, .pred_Water)) %>%
  autoplot()

roc_curves
```




```
final_model_conf <- augment(pokemon_final_fit, new_data = test) %>%
  conf_mat(truth = type_1, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
final_model_conf
```



I would say overall my model did decent. I had a ROC auc of 67% and a overall accuracy of .351. It seemed like my most accurate type being modeled was water and normal.