PSTAT 131 - Homework Assignment 6

Akshat Ataliwala (7924145)

May 17, 2022

Tree Based Models

For this assignment, we will continue 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(ggplot2)
library(corrr)
library(klaR)
library(glmnet)
library(MASS)
library(discrim)
library(poissonreg)
library(janitor)
tidymodels_prefer()
```

Exercise 1

Read in the data and set things up as in Homework 5:

```
data <- read_csv("data/pokemon.csv")
data %>% head(5)
```

```
## # A tibble: 5 x 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>
```

- Use `clean_names()`
- Filter out the rarer Pokémon types
- Convert `type_1` and `legendary` to factors

```
# Clean Names
data <- data %>% clean_names()

# Filter out rarer Pokemon types
data <- data %>% filter(type_1 == "Bug" | type_1 == "Fire" |
                          type_1 == "Grass" | type_1 == "Normal" |
                          type_1 == "Water" | type_1 == "Psychic")

# Convert type_1, legendary, generation to factors
data$type_1 <- as.factor(data$type_1)
data$generation <- as.factor(data$generation)
data$legendary <- as.factor(data$legendary)


data %>% head(5)
```

```
## # A tibble: 5 x 13
##   number name       type_1 type_2 total    hp attack defense sp_atk sp_def speed
##    <dbl> <chr>      <fct>  <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 VenusaurM~ Grass  Poison   625    80    100     123    122    120    80
## 5      4 Charmander Fire   <NA>     309    39     52      43     60     50    65
## # ... with 2 more variables: generation <fct>, legendary <fct>
```

Do an initial split of the data; you can choose the percentage for splitting. Stratify on the outcome variable.

Fold the training set using *v*-fold cross-validation, with `v = 5`. Stratify on the outcome variable.

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.

Exercise 2

Create a correlation matrix of the training set, using the `corrplot` package. *Note: You can choose how to handle the continuous variables for this plot; justify your decision(s).*

What relationships, if any, do you notice? Do these relationships make sense to you?

Exercise 3

First, set up a decision tree model and workflow. Tune the `cost_complexity` hyperparameter. Use the same levels we used in Lab 7 – that is, `range = c(-3, -1)`. Specify that the metric we want to optimize is `roc_auc`.

Print an `autoplot()` of the results. What do you observe? Does a single decision tree perform better with a smaller or larger complexity penalty?

Exercise 4

What is the `roc_auc` of your best-performing pruned decision tree on the folds? *Hint: Use `collect_metrics()` and `arrange()`.*

Exercise 5

Using `rpart.plot`, fit and visualize your best-performing pruned decision tree with the *training* set.

Exercise 5

Now set up a random forest model and workflow. Use the `ranger` engine and set `importance = "impurity"`. Tune `mtry`, `trees`, and `min_n`. Using the documentation for `rand_forest()`, explain in your own words what each of these hyperparameters represent.

Create a regular grid with 8 levels each. You can choose plausible ranges for each hyperparameter. Note that `mtry` should not be smaller than 1 or larger than 8. **Explain why not. What type of model would `mtry = 8` represent?**

Exercise 6

Specify `roc_auc` as a metric. Tune the model and print an `autoplot()` of the results. What do you observe? What values of the hyperparameters seem to yield the best performance?

Exercise 7

What is the `roc_auc` of your best-performing random forest model on the folds? *Hint: Use `collect_metrics()` and `arrange()`.*

Exercise 8

Create a variable importance plot, using `vip()`, with your best-performing random forest model fit on the *training* set.

Which variables were most useful? Which were least useful? Are these results what you expected, or not?

Exercise 9

Finally, set up a boosted tree model and workflow. Use the `xgboost` engine. Tune `trees`. Create a regular grid with 10 levels; let `trees` range from 10 to 2000. Specify `roc_auc` and again print an `autoplot()` of the results.

What do you observe?

What is the `roc_auc` of your best-performing boosted tree model on the folds? *Hint: Use `collect_metrics()` and `arrange()`.*

Exercise 10

Display a table of the three ROC AUC values for your best-performing pruned tree, random forest, and boosted tree models. Which performed best on the folds? Select the best of the three and use `select_best()`, `finalize_workflow()`, and `fit()` to fit it to the *testing* set.

Print the AUC value of your best-performing model on the testing set. Print the ROC curves. Finally, create and visualize a confusion matrix heat map.

Which classes was your model most accurate at predicting? Which was it worst at?