PSTAT131HW05

Yifei Zhang

2022-05-10

```
library(ggplot2)
library(tidyverse)
library(tidymodels)
library(corrplot)
library(klaR)
library(glmnet)
tidymodels_prefer()
Pokemon <- read_csv("Pokemon.csv")
pokemon <- readr::spec(Pokemon)
pokemon
```

```
## cols(
##   `#` = col_double(),
##   Name = col_character(),
##   `Type 1` = col_character(),
##   `Type 2` = col_character(),
##   Total = col_double(),
##   HP = col_double(),
##   Attack = col_double(),
##   Defense = col_double(),
##   `Sp. Atk` = col_double(),
##   `Sp. Def` = col_double(),
##   Speed = col_double(),
##   Generation = col_double(),
##   Legendary = col_logical()
## )
```

```
library(janitor)
```

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?

```
cleaned <- clean_names(Pokemon)
cleaned
```

```
## # 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>
```

As the function name suggests, it cleaned the names of our Pokemon data, made the names unique. I think clean_names() is useful, because sometime we may have variables with names that are very similar to each other, and have space as a part of the name, which can cause a lot of problems, and by using clean_names() we can avoid these problems, and we can change the letter case all at once.

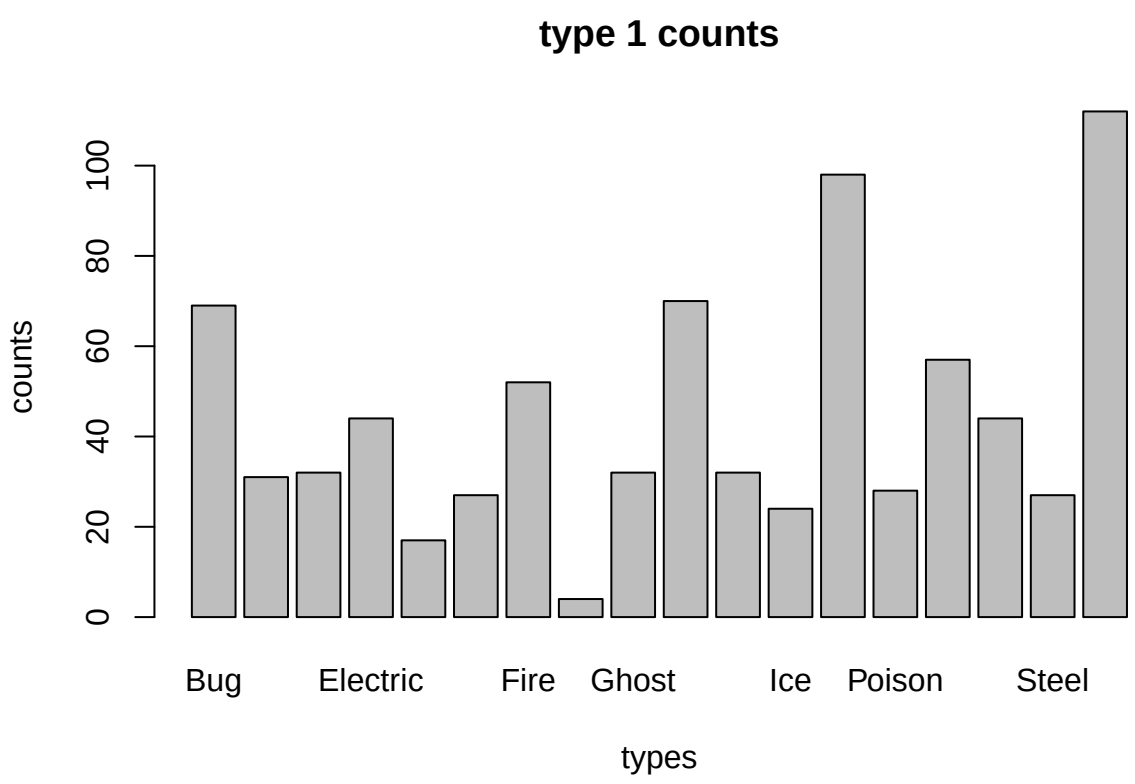
Exercise 2

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

```
count <- table(cleaned$type_1)
count
```

```
##
##      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(count, xlab = "types", ylab = "counts", main ="type 1 counts")
```




```
# not all the type names are shown in the xlab because thats too long
```

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

```
There are 18 class of outcomes. There are Pokémon types with very few Pokémon. Such as the  Flying class
```

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.

```
filtered <- cleaned %>% filter(
  type_1 == "Bug" | type_1 == "Fire" | type_1 == "Grass"
  |type_1 == " Normal" | type_1 == "Water" | type_1 == "Psychic"
  )
filtered
```

```
## # A tibble: 360 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 350 more rows, and 2 more variables: generation <dbl>,
## #   legendary <lgl>
```

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

```
data <- filtered %>%
  mutate(type_1 = factor(type_1),
         legendary = factor(legendary),
         generation = factor(generation)
         )
data
```

```
## # A tibble: 360 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 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 350 more rows, and 2 more variables: generation <fct>,
## #   legendary <fct>
```

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 <- data %>%
  initial_split(strata = type_1, prop = 0.7)
pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)
dim(pokemon_train)
```

```
## [1] 250  13
```

```
There are 250 observations in our training set, which is roughly 70 percent of our filtered overall 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_folds <- vfold_cv(pokemon_train, v = 5, strata = 'type_1')
pokemon_folds
```

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 x 2
##   splits           id
##   <list>           <chr>
## 1 <split [198/52]> Fold1
## 2 <split [199/51]> Fold2
## 3 <split [199/51]> Fold3
## 4 <split [201/49]> Fold4
## 5 <split [203/47]> Fold5
```

Stratifying the folds can be useful useful because we want to make sure the cross validation reaches the result as realistic as possible.

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_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?

We will be fitting 500 models to our folded data.

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

elastic_workflow <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(elastic_spec)
```

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

```
## # A tibble: 100 x 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
```

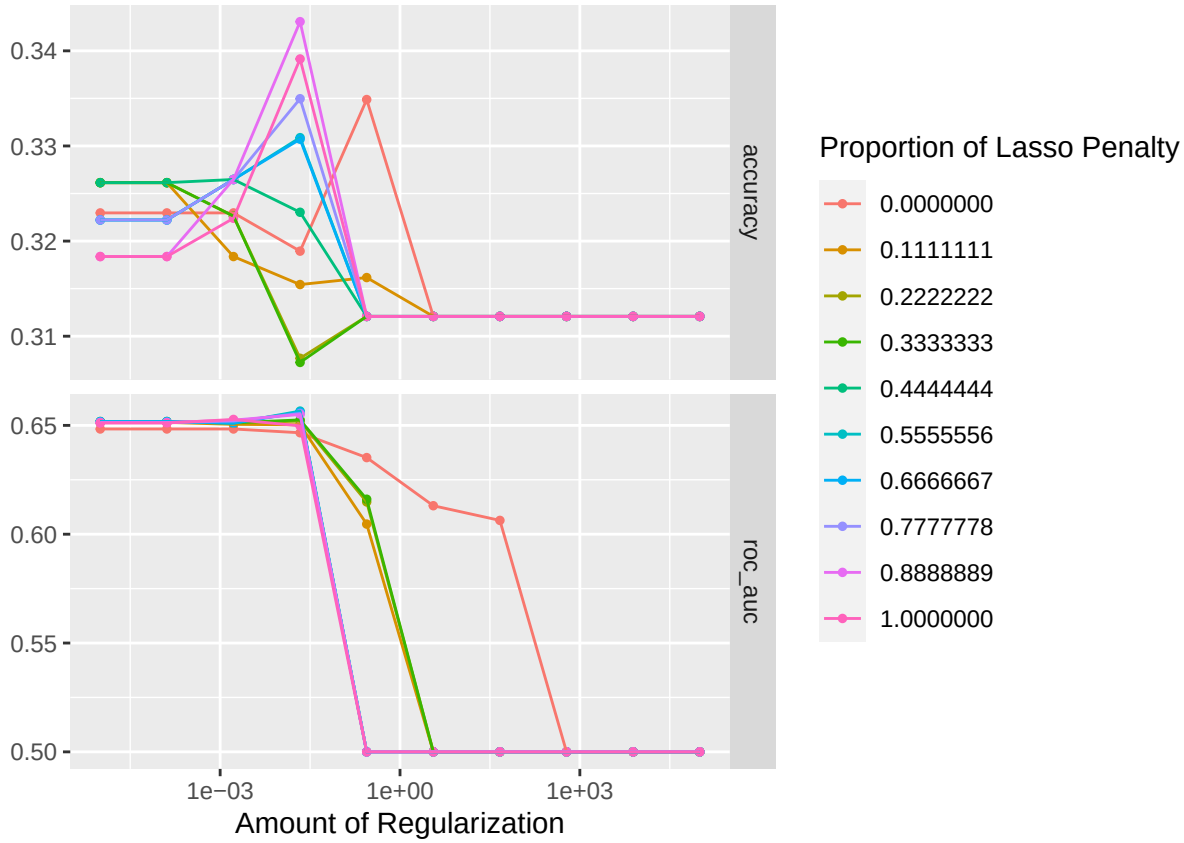
Exercise 6

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?

```
tune_res <- tune_grid(
  elastic_workflow,
  resamples = pokemon_folds,
  grid = penalty_grid
)
```

```
autoplot(tune_res)
```



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.

```
collect_metrics(tune_res)
```

```
## # A tibble: 200 x 8
##     penalty mixture .metric  .estimator  mean     n std_err .config
##       <dbl>   <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1 0.00001        0 accuracy multiclass 0.323     5  0.0219 Preprocessor1_Model~
##  2 0.00001        0 roc_auc  hand_till  0.648     5  0.0143 Preprocessor1_Model~
##  3 0.000129       0 accuracy multiclass 0.323     5  0.0219 Preprocessor1_Model~
##  4 0.000129       0 roc_auc  hand_till  0.648     5  0.0143 Preprocessor1_Model~
##  5 0.00167        0 accuracy multiclass 0.323     5  0.0219 Preprocessor1_Model~
##  6 0.00167        0 roc_auc  hand_till  0.648     5  0.0143 Preprocessor1_Model~
##  7 0.0215         0 accuracy multiclass 0.319     5  0.0221 Preprocessor1_Model~
##  8 0.0215         0 roc_auc  hand_till  0.647     5  0.0151 Preprocessor1_Model~
##  9 0.278          0 accuracy multiclass 0.335     5  0.0171 Preprocessor1_Model~
## 10 0.278          0 roc_auc  hand_till  0.635     5  0.0232 Preprocessor1_Model~
## # ... with 190 more rows
```

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

```
## # A tibble: 1 x 3
##   penalty mixture .config
##     <dbl>   <dbl> <chr>
## 1  0.0215   0.556 Preprocessor1_Model054
```

```
elastic_final <- finalize_workflow(elastic_workflow, best_penalty)

elastic_final_fit <- fit(elastic_final, data = pokemon_train)
```

augment(elastic_final_fit, new_data = pokemon_test) %>% rsq(truth = Salary, estimate = .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?