131-Homework5

Caleb Mazariegos

2022-05-13

```
pokemon_codebook <- read.csv("/Users/calebmazariegos/Desktop/homework-5/Pokemon.csv")
head(pokemon_codebook)
```

```
##   X.                  Name Type.1 Type.2 Total HP Attack Defense Sp..Atk
## 1  1             Bulbasaur  Grass Poison   318 45     49      49      65
## 2  2               Ivysaur  Grass Poison   405 60     62      63      80
## 3  3              Venusaur  Grass Poison   525 80     82      83     100
## 4  3 VenusaurMega Venusaur  Grass Poison   625 80    100     123     122
## 5  4            Charmander   Fire          309 39     52      43      60
## 6  5            Charmeleon   Fire          405 58     64      58      80
##   Sp..Def Speed Generation Legendary
## 1      65    45          1     False
## 2      80    60          1     False
## 3     100    80          1     False
## 4     120    80          1     False
## 5      50    65          1     False
## 6      65    80          1     False
```

Exercise 1

```
# loading the janitor package

pokemon_codebook <- clean_names(pokemon_codebook)

head(pokemon_codebook)
```

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?

```
##   x                  name type_1 type_2 total hp attack defense sp_atk sp_def
## 1 1             Bulbasaur  Grass Poison   318 45     49      49     65     65
## 2 2               Ivysaur  Grass Poison   405 60     62      63     80     80
## 3 3              Venusaur  Grass Poison   525 80     82      83    100    100
## 4 3 VenusaurMega Venusaur  Grass Poison   625 80    100     123    122    120
## 5 4            Charmander   Fire          309 39     52      43     60     50
## 6 5            Charmeleon   Fire          405 58     64      58     80     65
##   speed generation legendary
```

```
## 1    45          1     False
## 2    60          1     False
## 3    80          1     False
## 4    80          1     False
## 5    65          1     False
## 6    80          1     False
```

clean_names() changed the variable names to lowercase, and added underscores instead of the period that was in the variable name before. I think this is useful because it makes them easier to code, and makes the code more readable and understandable.

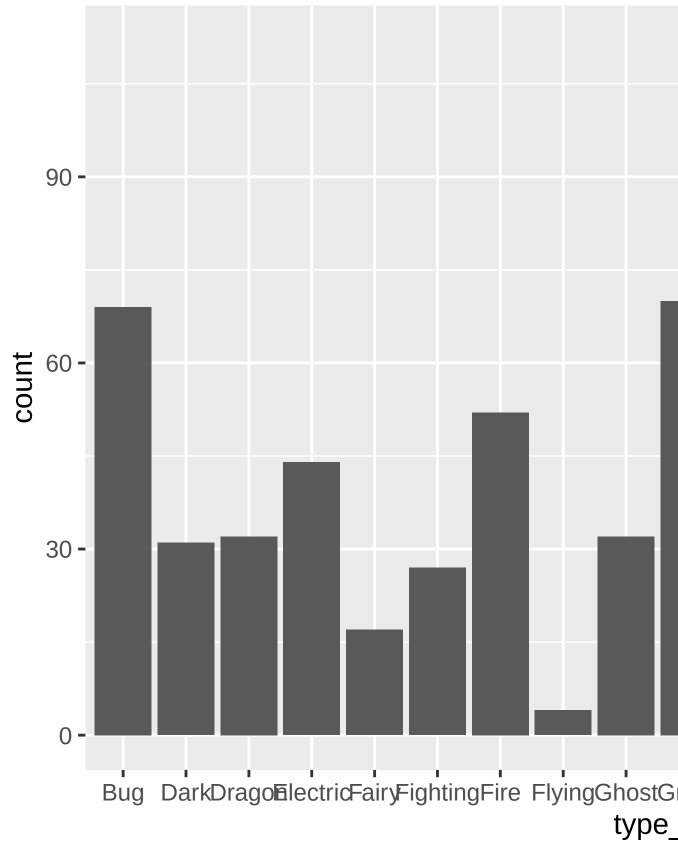
Exercise 2

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.

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



After filtering, convert type_1 and legendary to factors.

There are 18 classes of the outcome variable. The flying Pokémon type is has very few Pokémon.

```
# filtering out pokemon types that are not Bug, Fire, Grass, Normal, Water, or Psychic

pokemon_filter <- pokemon_codebook %>%
  filter((type_1 == "Bug" | type_1 == "Fire" | type_1 == "Grass" | type_1 == "Normal" | type_1 == "Wate

head(pokemon_filter)
```

```
##   x                  name type_1 type_2 total hp attack defense sp_atk sp_def
## 1 1             Bulbasaur  Grass Poison   318 45     49      49     65     65
## 2 2               Ivysaur  Grass Poison   405 60     62      63     80     80
## 3 3              Venusaur  Grass Poison   525 80     82      83    100    100
## 4 3 VenusaurMega Venusaur  Grass Poison   625 80    100     123    122    120
## 5 4            Charmander   Fire          309 39     52      43     60     50
## 6 5            Charmeleon   Fire          405 58     64      58     80     65
##   speed generation legendary
## 1    45          1     False
## 2    60          1     False
## 3    80          1     False
## 4    80          1     False
## 5    65          1     False
## 6    80          1     False
```

```r
# converting type 1 and legendary in factors
pokemon_codebook$type_1 <- as.factor(pokemon_codebook$type_1)
pokemon_codebook$legendary <- as.factor(pokemon_codebook$legendary)
pokemon_codebook$generation <- as.factor(pokemon_codebook$generation)

pokemon_filter$type_1 <- as.factor(pokemon_filter$type_1)
pokemon_filter$legendary <- as.factor(pokemon_filter$legendary)
pokemon_filter$generation <- as.factor(pokemon_filter$generation)
```

Exercise 3

```r
# Setting the seed
set.seed(3465)

pokemon_split <- initial_split(pokemon_filter, prop = 0.70, stata = type_1)

pokemon_train <- training(pokemon_split)

pokemon_test <- testing(pokemon_split)
```

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?

```r
set.seed(234)
pokemon_folds <- vfold_cv(pokemon_train, v=5)
pokemon_folds
```

```
## #  5-fold cross-validation
## # A tibble: 5 x 2
##   splits           id
##   <list>           <chr>
## 1 <split [256/64]> Fold1
## 2 <split [256/64]> Fold2
## 3 <split [256/64]> Fold3
## 4 <split [256/64]> Fold4
## 5 <split [256/64]> Fold5
```

Excercise 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_d
  step_dummy(legendary) %>%
  step_dummy(generation) %>%
  step_center(all_predictors()) %>%
  step_scale(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?

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

pokemon_workflow <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(pokemon_spec)

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

head(penalty_grid)
```

```
## # A tibble: 6 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
```

We will be fitting 500 models when we fit these models to our folded data.

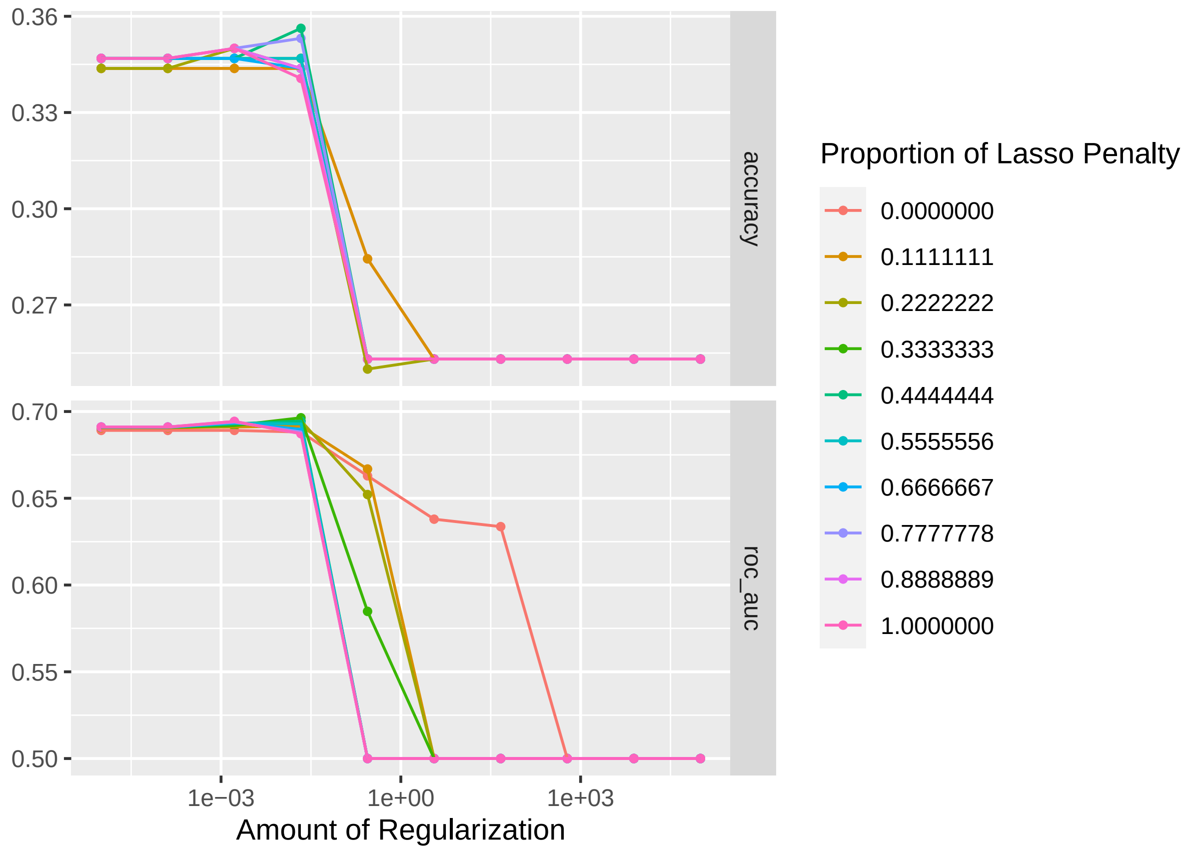
Excercise 6

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(
  pokemon_workflow,
  resamples = pokemon_folds,
  grid = penalty_grid)

autoplot(tune_res)
```



Smaller values of penalty and mixture produce better accuracy and ROC AUC.

Excercise 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_roc <- select_best(tune_res, metric = "roc_auc")
wkflw_final <- finalize_workflow(pokemon_workflow, best_roc)
final_fit <- fit(wkflw_final, data = pokemon_train)
aug <- augment(final_fit, new_data = pokemon_test) %>%
  accuracy(truth = type_1, estimate = .pred_class)

aug
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
```

```
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.341
```

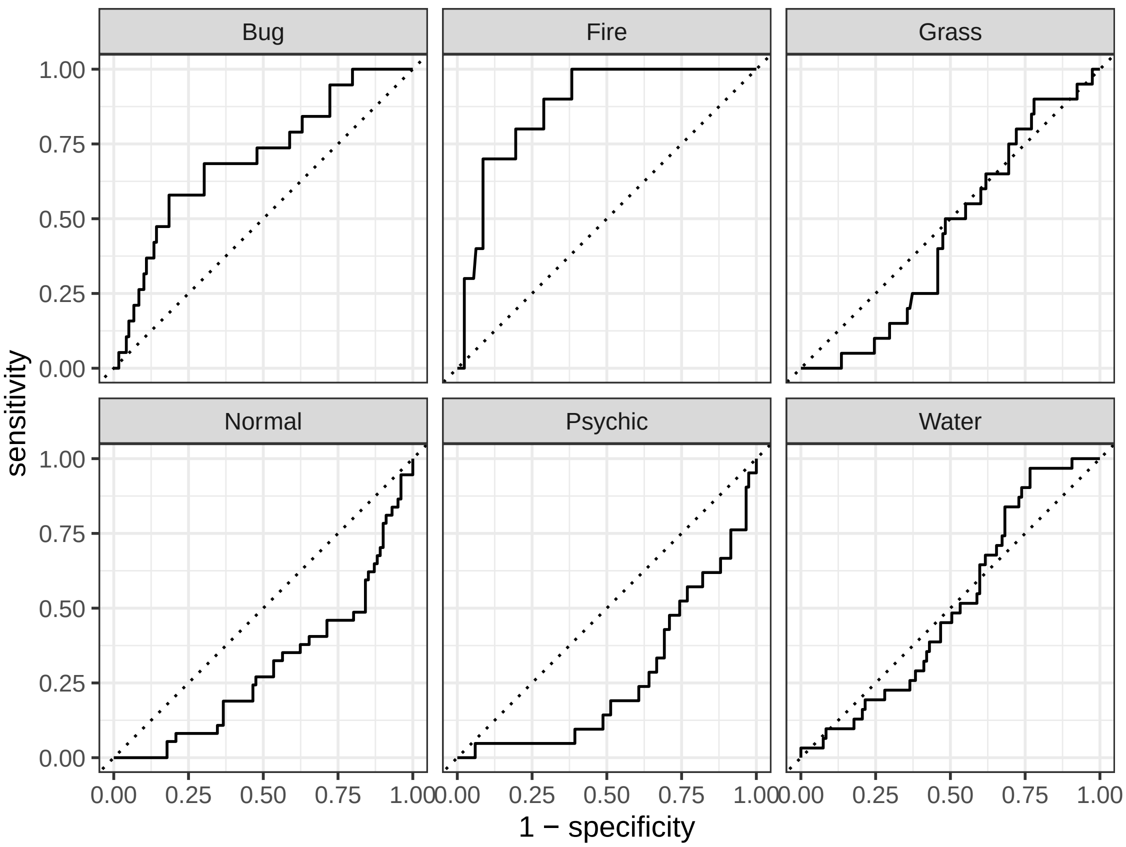
Excercise 8

Calculate the overall ROC AUC on the testing set.

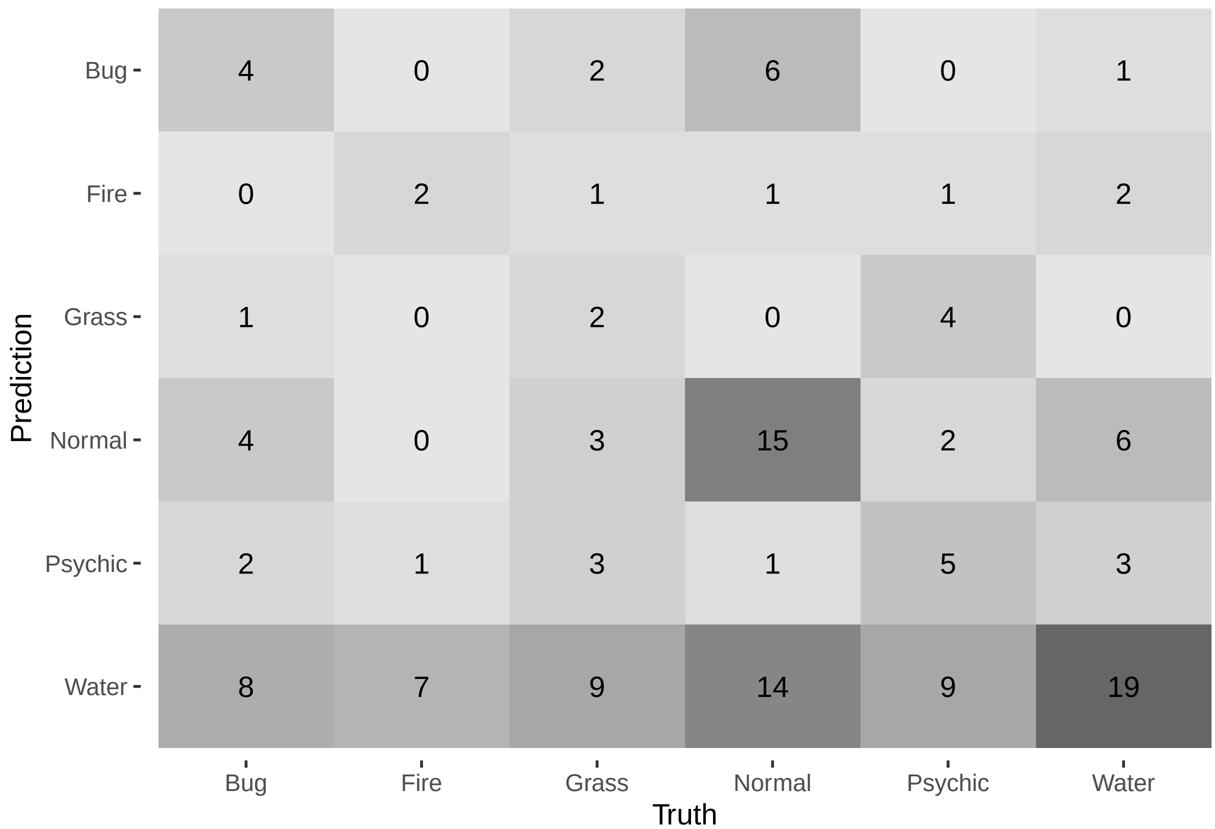
Then create plots of the different ROC curves, one per level of the outcome. Also make a heat map of the confusion matrix.

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?

```
augment(final_fit, new_data = pokemon_test) %>%
  roc_curve(type_1, estimate=c(.pred_Bug, .pred_Fire, .pred_Water, .pred_Grass, .pred_Normal, .pred_Psy
  autoplot()
```




```
augment(final_fit, new_data = pokemon_test) %>%
  conf_mat(truth = type_1, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```



I would say that my model performed relatively well. My model is best at predicting Bug and Fire pokemon types. It is worst at predicting Normal and Psychic pokemon.