Homework 5

Kabir Snell

2022-11-27

```r
# Loading Libraries
library(tidymodels)
library(ISLR)
library(ISLR2)
library(tidyverse)
library(glmnet)
library(janitor)

tidymodels_prefer()
```

```r
# Loading Data
pokemon <- read.csv('data/Pokemon.csv')
```

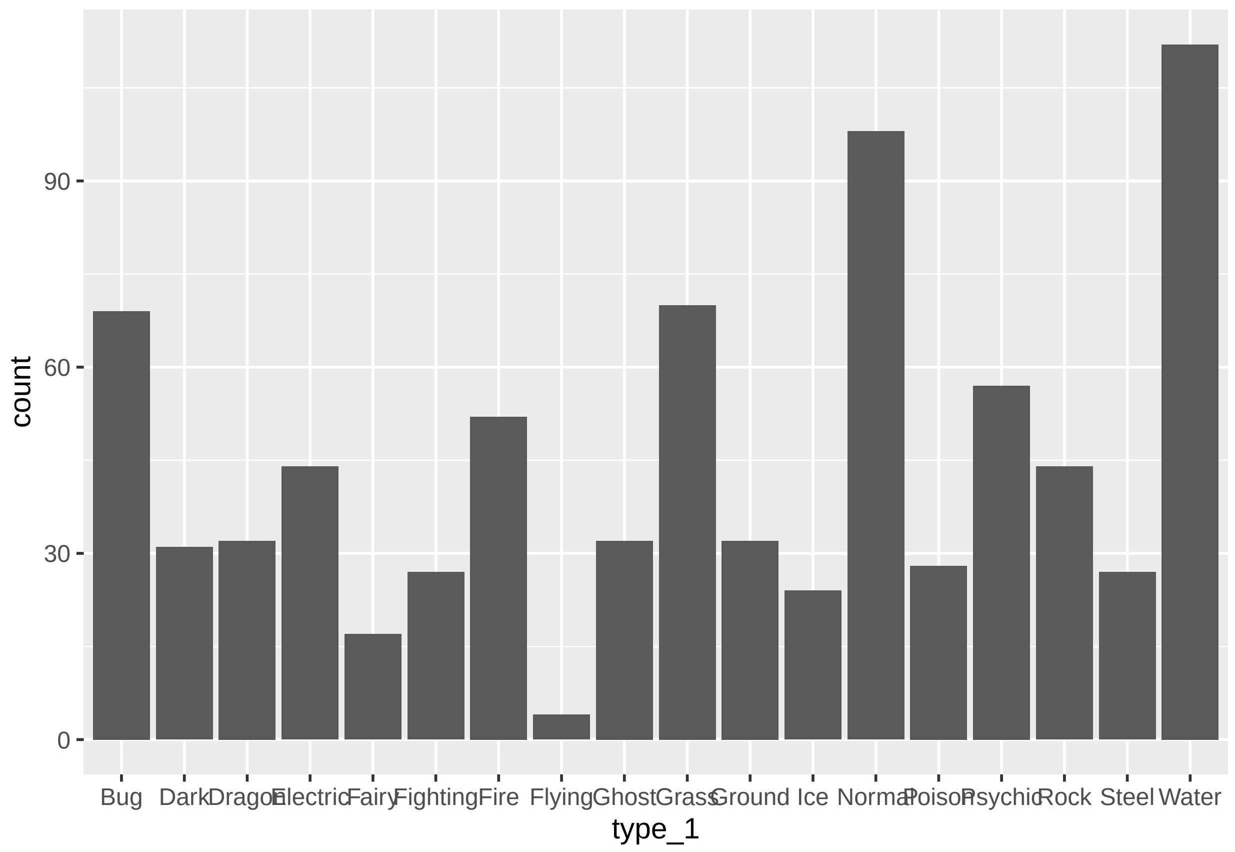
Question 1

```r
pokemon <- pokemon %>%
  clean_names()
```

The variable names for the data set became much more standardized. For example, Sp..Atk and Sp..Def became sp_atk and sp_def. Additionally, all of the first letters of the variable names became lowercase.

Question 2

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



There are 18 Pokemon types in this data set, with flying having the fewest observations.

```
pokemon <- pokemon %>%
  filter(type_1 =='Bug' | type_1 == 'Fire' | type_1 == 'Grass' | type_1 == 'Normal' | type_1 == 'Water'

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

Question 3

```
set.seed(727)

pokemon_split <- initial_split(pokemon, prop = 0.80, strata = type_1)

pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)

pokemon_split
```

```
## <Training/Testing/Total>
## <364/94/458>
```

```r
pokemon_fold <- vfold_cv(pokemon_train, v = 5)
```

Stratifying the folds may be useful as it offers a form of cross validation that we wouldn't otherwise have had.

Question 4

```r
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_c
  step_dummy(legendary) %>%
  step_dummy(generation) %>%
  step_normalize(all_predictors())
```

Question 5

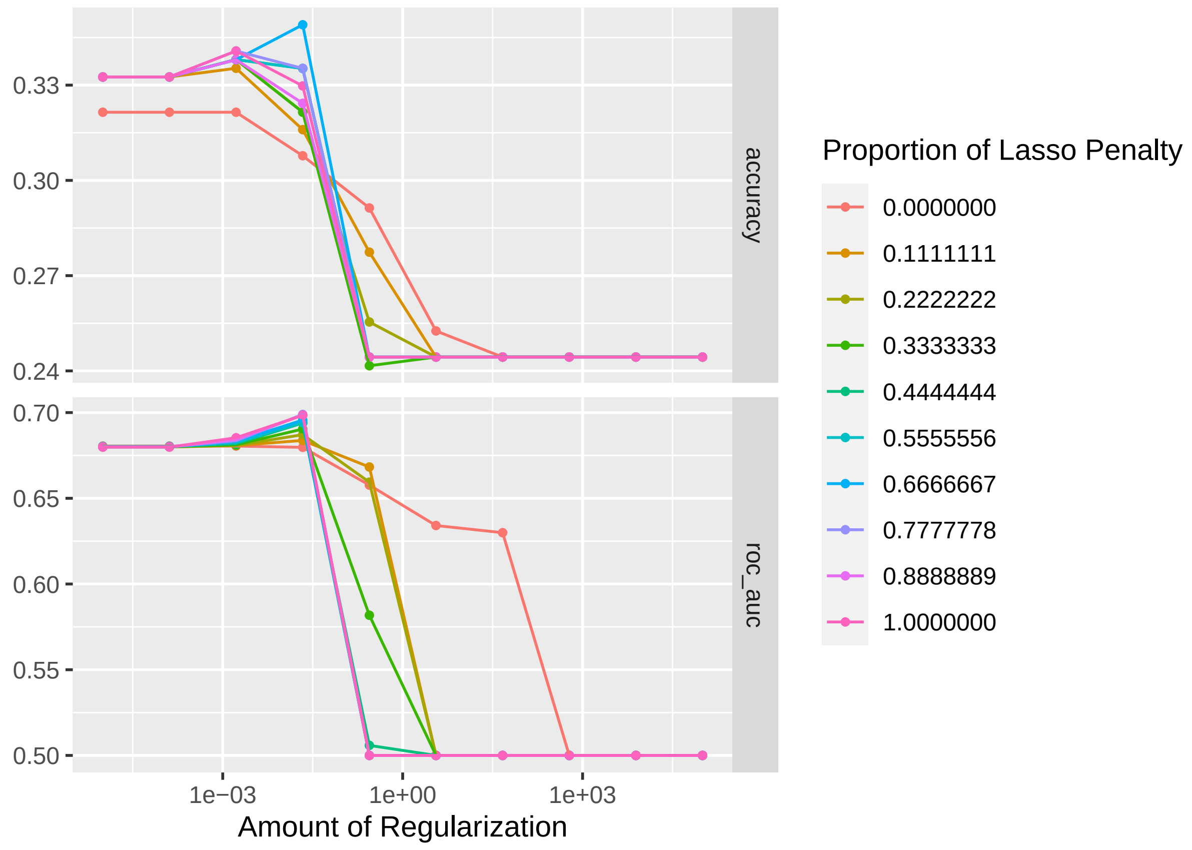
In total, we will be fitting 500 models

```r
pokemon_spec <- multinom_reg(mode = "classification",
  engine = "glmnet",
  penalty = tune(),
  mixture = tune())

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

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

pokemon_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
```

Question 6

```
tune_res <- tune_grid(
  pokemon_wkflow,
  resamples = pokemon_fold,
  grid = pokemon_grid)

autoplot(tune_res)
```



Generally smaller values of penalty produced better accuracy and roc_auc. For mixture, generally larger values prouced better results for accuracy and roc_auc. However, the peak values for penalty were slightly below the middle and the best value for mixture was .555555

Question 7

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

pokemon_final <- finalize_workflow(pokemon_wkflow, best_penalty)

pokemon_final_fit <- fit(pokemon_final, data = pokemon_train)

augment(pokemon_final_fit, new_data = pokemon_test) %>%
  roc_curve(truth = type_1, estimate = c(.pred_Bug,.pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic
```

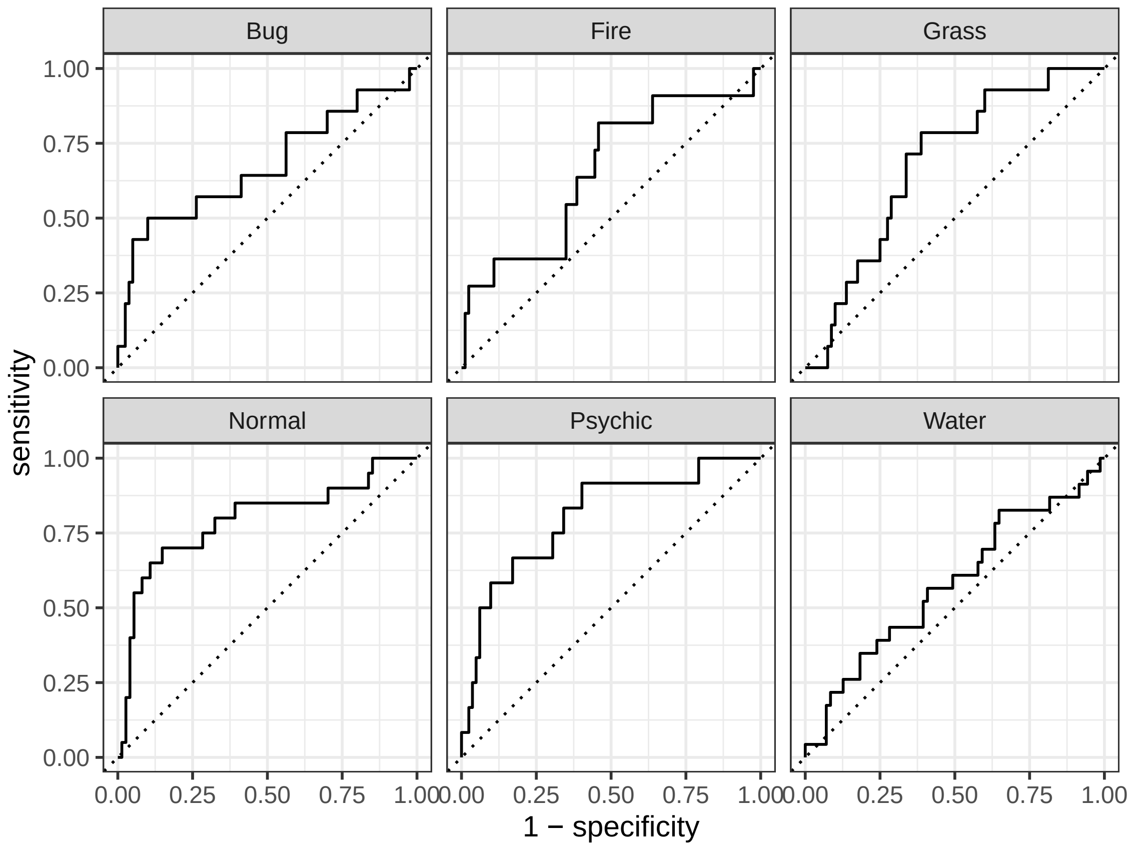
```
## # A tibble: 576 x 4
##    .level .threshold specificity sensitivity
##    <chr>       <dbl>       <dbl>       <dbl>
##  1 Bug    -Inf             0           1
##  2 Bug       0.00762      0           1
##  3 Bug       0.0142       0.0125      1
##  4 Bug       0.0227       0.0250      1
##  5 Bug       0.0250       0.0250      0.929
##  6 Bug       0.0250       0.0375      0.929
##  7 Bug       0.0292       0.0500      0.929
##  8 Bug       0.0303       0.0625      0.929
##  9 Bug       0.0304       0.0750      0.929
## 10 Bug       0.0307       0.0875      0.929
## # ... with 566 more rows
```

Question 8

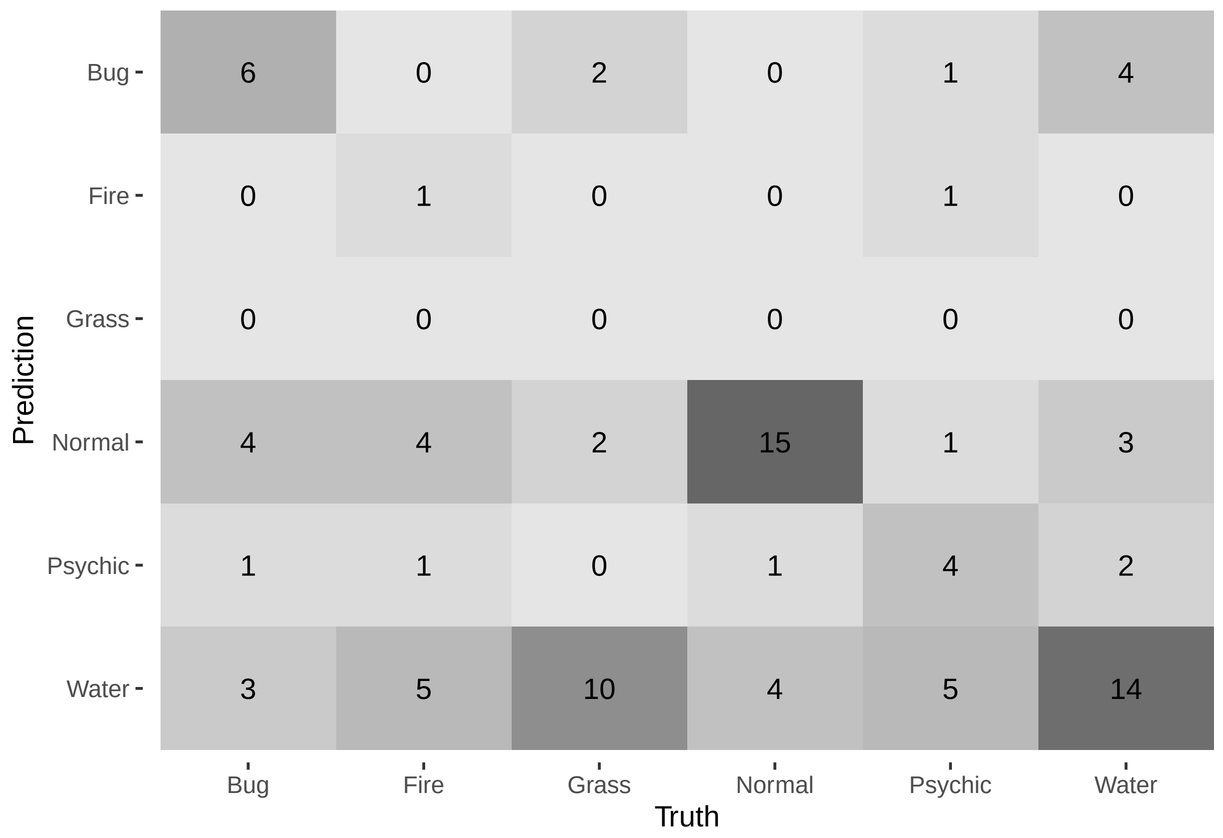
```
augment(pokemon_final_fit, new_data = pokemon_test) %>%
roc_auc(truth = type_1, estimate = c(.pred_Bug,.pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic, .p
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.695
```

```
augment(pokemon_final_fit, new_data = pokemon_test) %>%
  roc_curve(truth = type_1, estimate = c(.pred_Bug,.pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic
  autoplot()
```



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



The model did well for some pokemon types while it did pretty bad for others. It was able to predict normal, psychic and bug pokemon pretty well, but struggled heavily with grass and fire.

One idea as to that might be is that normal type pokemon generally have specific characteristics such as being physical attackers (high attack and low sp_atk), as well as being generally defensive. Just looking at the numbers it may be easy to discern normal type pokemon from the rest. Although all pokemon have tendencies for stats, it may be harder to categorize groups of pokemon by looking at the stats.

Normal, bug and psychic pokemon are unique in their attributes, making it easy to categorize them, while the other types of pokemon are more vague.