Homework 5

Mason Wong

2022-11-19

Exercise 1

```
library(janitor)
```

```
## Warning: package 'janitor' was built under R version 4.2.2
```

```
##
## Attaching package: 'janitor'
```

```
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
```

```
## v tibble  3.1.8     v purrr   0.3.4
## v tidyr   1.2.1     v stringr 1.4.1
## v readr   2.1.2     v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.2.2
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-4
```

```
library(tune)
library(tidymodels)
```

```
## -- Attaching packages --------------------------------------- tidymodels 1.0.0 --
## v broom        1.0.1     v recipes      1.0.1
## v dials        1.0.0     v rsample      1.1.0
## v infer        1.0.3     v workflows    1.1.0
## v modeldata    1.0.1     v workflowsets 1.0.0
## v parsnip      1.0.2     v yardstick    1.1.0
## -- Conflicts ------------------------------------------ tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x Matrix::expand()  masks tidyr::expand()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x Matrix::pack()    masks tidyr::pack()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## x Matrix::unpack()  masks tidyr::unpack()
## x recipes::update() masks Matrix::update(), stats::update()
## * Use tidymodels_prefer() to resolve common conflicts.
```

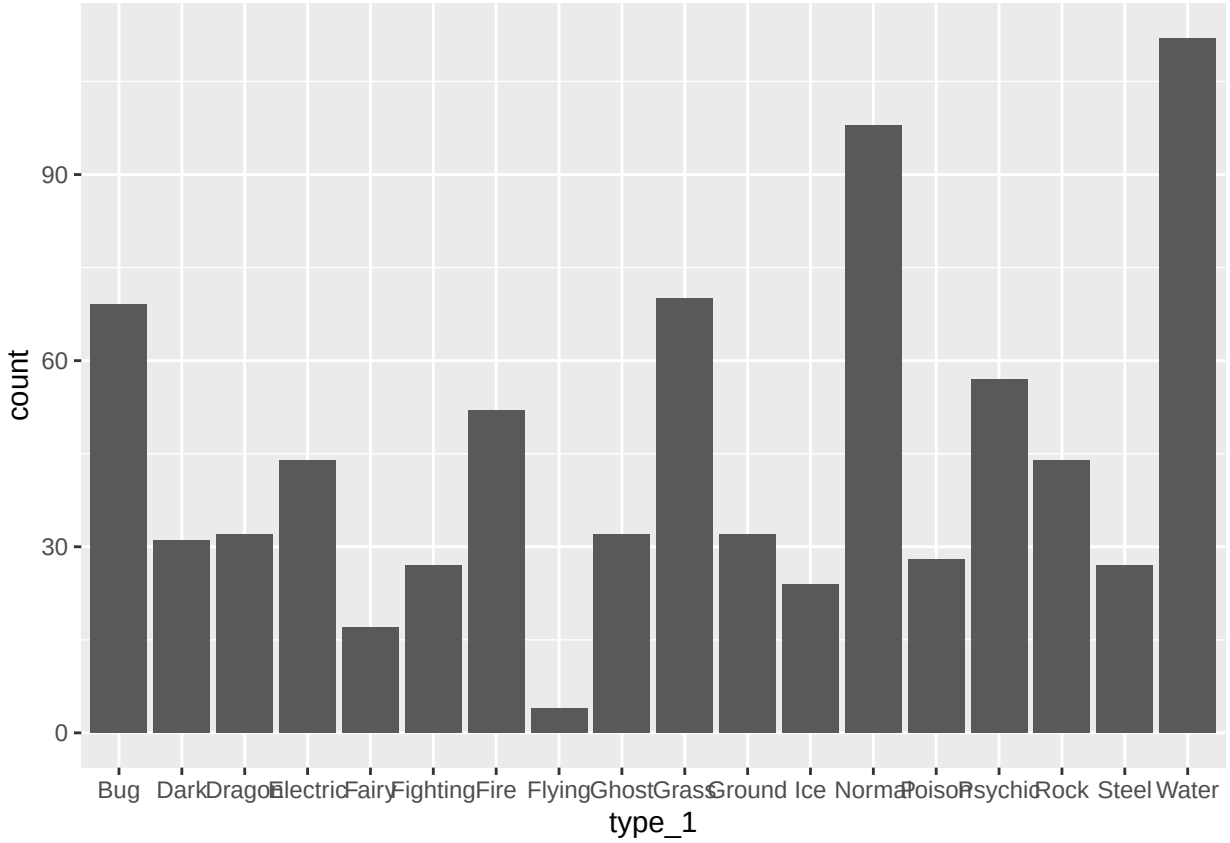
```
pokemon <- read.csv("Pokemon.csv")
set.seed(100)

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

clean_names() made all of the column names lowercase and turned any empty spaces or periods into underscores

This is useful because it helps to make our variable names uniform and prevent us from forgetting if we have capital letters in our variables when we are processing a long chunk of code

Exercise 2

```
ggplot(data=pokemon_clean, aes(type_1)) + geom_bar()
```



There are 18 possible types for Type 1, with the fewest types being Flying and Fairy

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

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

```
pokemon_split <- initial_split(pokemon_filter, prop = 0.7, strata = type_1)

pokemon_train <- training(pokemon_split)

pokemon_test <- testing(pokemon_split)

poke_fold <- vfold_cv(pokemon_train, v=5, strata = type_1)
```

Stratifying is useful because it ensures that the data is evenly distributed and we don't have large variances between our data.

Exercise 4

```
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed +
                           defense + hp + sp_def, data = pokemon_train) %>%
  step_dummy(legendary)

pokemon_recipe <- step_dummy(recipe = pokemon_recipe, generation)

pokemon_recipe <- step_center(recipe = pokemon_recipe, sp_atk, attack,
                              speed, defense, hp, sp_def)

pokemon_recipe <- step_scale(recipe = pokemon_recipe, sp_atk, attack,
                             speed, defense, hp, sp_def)
```

Exercise 5

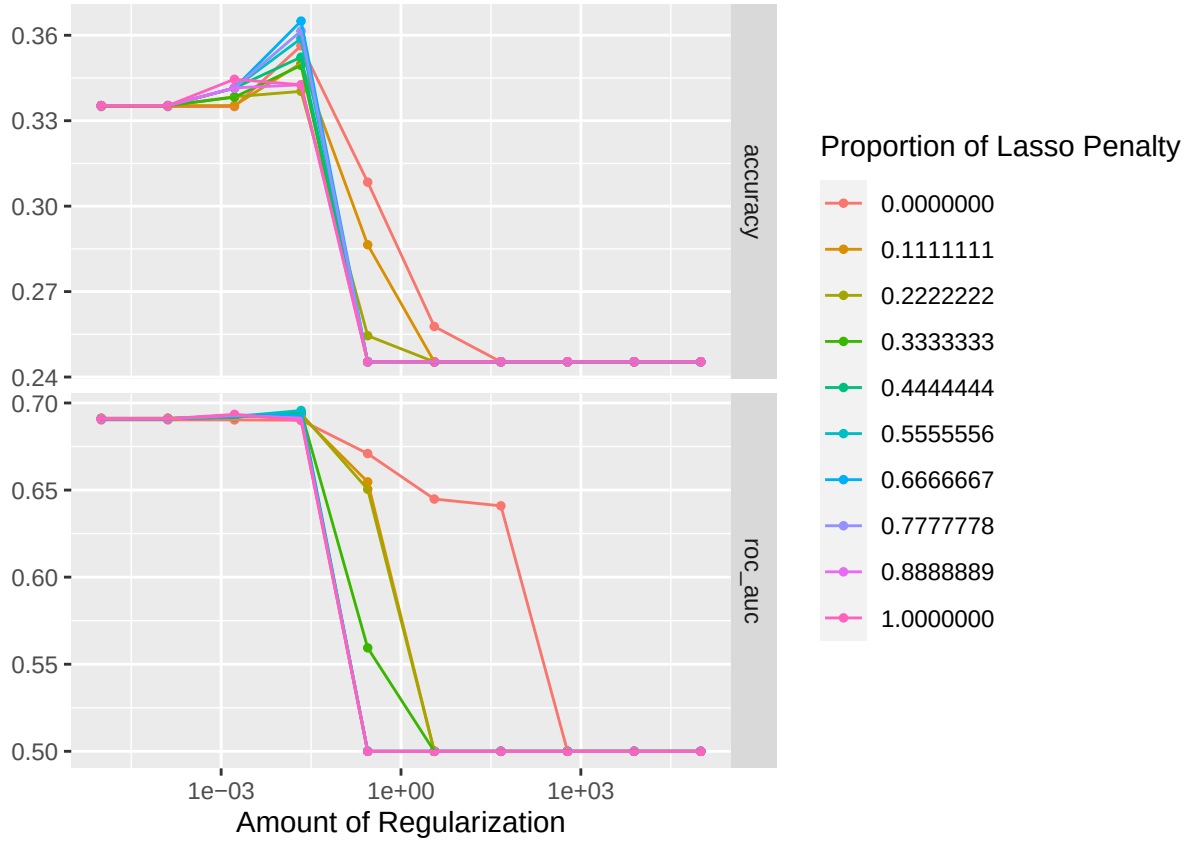
```
pokemon_multi <- multinom_reg() %>%
  set_engine("glmnet")

pokemon_wrkflw <- workflow() %>%
  add_model(pokemon_multi %>% set_args(mixture = tune(), penalty = tune())) %>%
  add_recipe(pokemon_recipe)

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

We will be fitting 100 models for each fold in our v fold, thus we are fitting 500 models.

Exercise 6

```
pokemon_mr <- tune_grid(
  pokemon_wrkflw,
  resamples = poke_fold,
  grid = comb_grid
)

autoplot(pokemon_mr)
```



It seems smaller values of penalty and mixture produce more accurate results

Exercise 7

```
best_model <- select_best(pokemon_mr)

pokemon_final <- finalize_workflow(pokemon_wrkflw, best_model)

pokemon_fit <- fit(pokemon_final, data = pokemon_train)

augment(pokemon_fit, new_data = pokemon_test) %>%
  accuracy(truth = type_1, estimate = .pred_class)
```

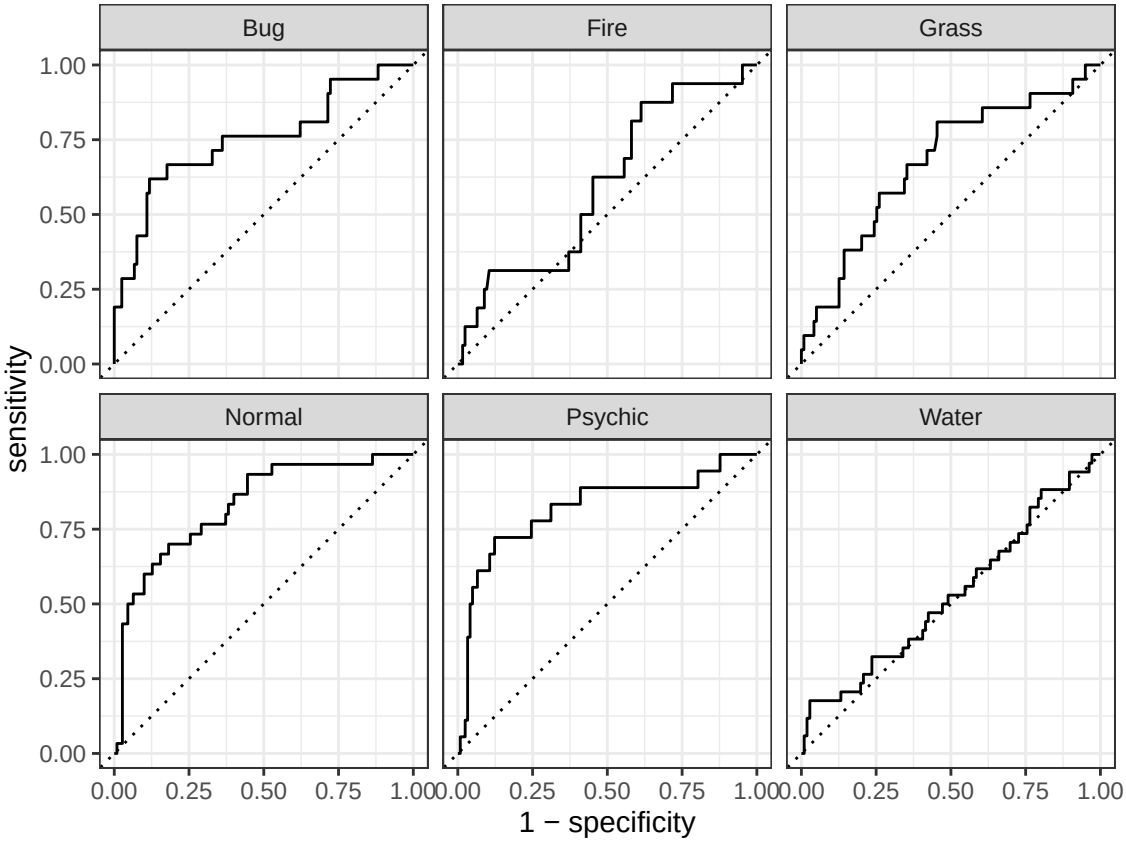
```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.357
```

Exercise 8

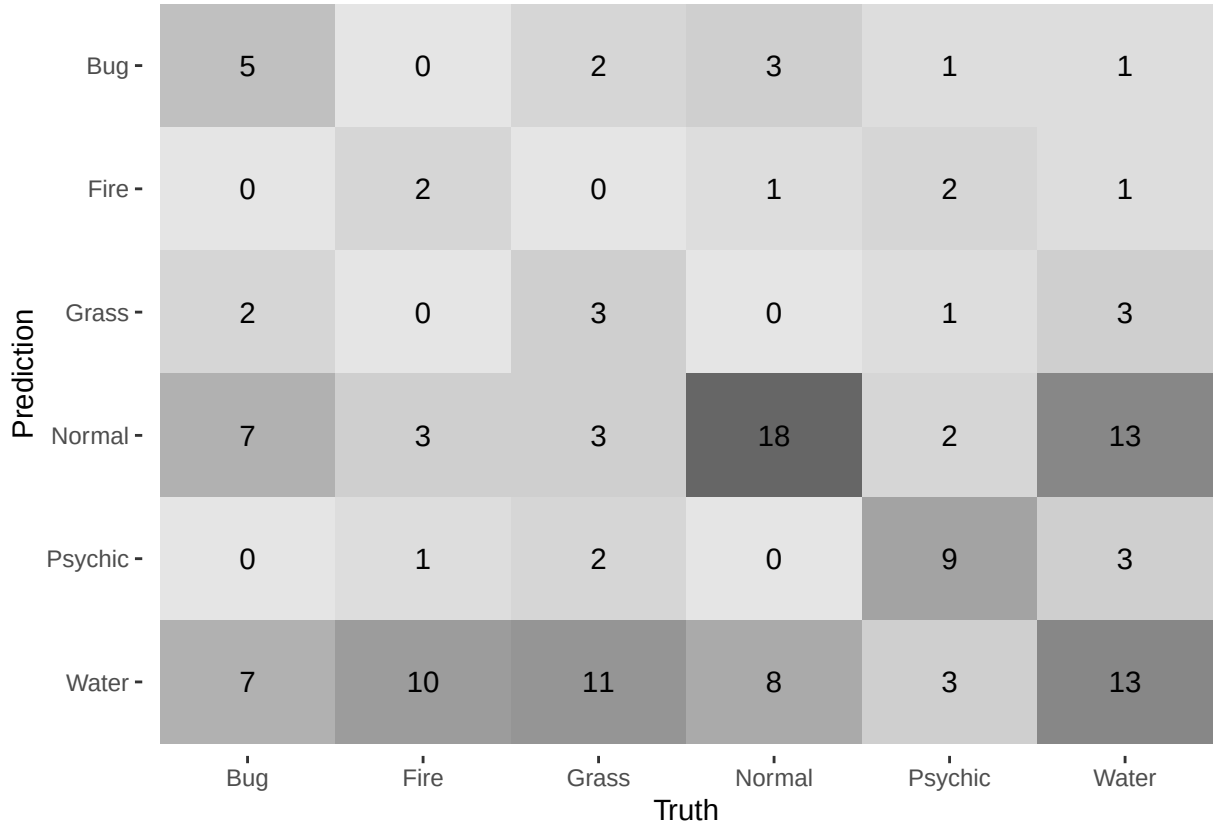
```
augment(pokemon_fit, new_data= pokemon_test) %>%
  roc_auc(truth = type_1, .pred_Bug:.pred_Water)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.700
```

```
augment(pokemon_fit, new_data= pokemon_test) %>%
  roc_curve(truth = type_1, .pred_Bug:.pred_Water) %>%
  autoplot()
```




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



The model was okay, the confusion matrix shows that it was really good at predicting Water and Normal types, but really bad at predicting Grass and fire types. On the contrary, the roc curve seems to suggest normal and psychic types were best predicted by this model, while water is the worst performing. I believe this slight inaccuracy is due to the fact that Normal and Water make up a majority of the pokemon that we were given in the data set. This would mean that our model has the most experience predicting water and normal types so it would be more likely to predict a water or normal type due to the bias in our data. This means that the best model that we have selected has the most experience predicting types based around water and normal, so there is a slight bias in the model's predictions.