HW5

Ben Hertzberg

2022-11-12

```
library(janitor)
library(ggplot2)
library(tidymodels)
library(tidyverse)
library(glmnet)
```

Exercise 1

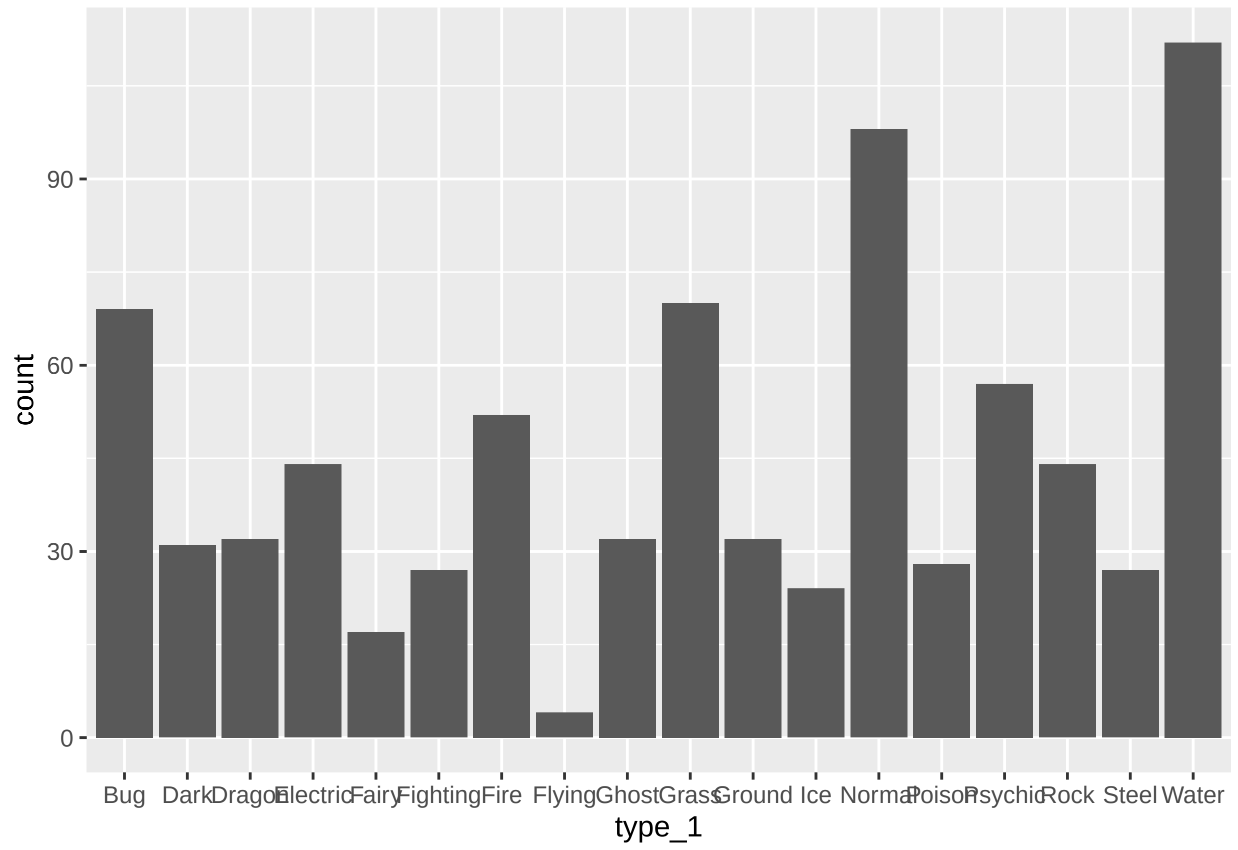
```
pok <- read.csv('./data/Pokemon.csv')
library(janitor)
```

```
pok <- clean_names(pok)
```

The variable names in the data set were all standardized. This function is useful because inconsistent formats for variables can lead to messy, less readable code and can potentially introduce bugs.

Exercise 2

```
type_1_plot<-ggplot(pok, aes(type_1)) +
  geom_bar()
type_1_plot
```



```
pok2 <- subset(pok, type_1 ==c('Bug','Fire','Grass',"Normal",'Water','Psychic'))
```

```
## Warning in type_1 == c("Bug", "Fire", "Grass", "Normal", "Water", "Psychic"):
## longer object length is not a multiple of shorter object length
```

```
pok2$type_1 <- as.factor(pok2$type_1)
pok2$legendary <- as.factor(pok2$legendary)
```

There are 18 classes of the type_1 variable. There are very few Flying and Fairy types

Exercise 3

```
set.seed(619)

pok_split <- initial_split(pok2, prop = 0.80, strata = type_1)
pok_train <- training(pok_split)
pok_test <- testing(pok_split)
dim(pok_train)
```

```
## [1] 63 13
```

```
dim(pok_test)
```

```
## [1] 19 13
```

```
pok_folds <- vfold_cv(pok_train, v = 5, strata = type_1)
pok_folds
```

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 x 2
##   splits          id
##   <list>          <chr>
## 1 <split [48/15]> Fold1
## 2 <split [49/14]> Fold2
## 3 <split [50/13]> Fold3
## 4 <split [52/11]> Fold4
## 5 <split [53/10]> Fold5
```

Stratifying the folds would be useful becuause some predictor possibilities occur rarely, and ina randomly selected sample without stratification, some of these possibilities might get passed over entirely and fail to assist in training.

Exercise 4

```
pok_rec <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_def, da

  step_dummy(legendary, generation) %>%
  step_normalize(all_numeric_predictors())
```

Exercise 5

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

net_wkflw <- workflow() %>%
  add_recipe(pok_rec) %>%
  add_model(net_mod)
```

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

I will be fitting 100 models to each of 5 folds, for a total of 500 model

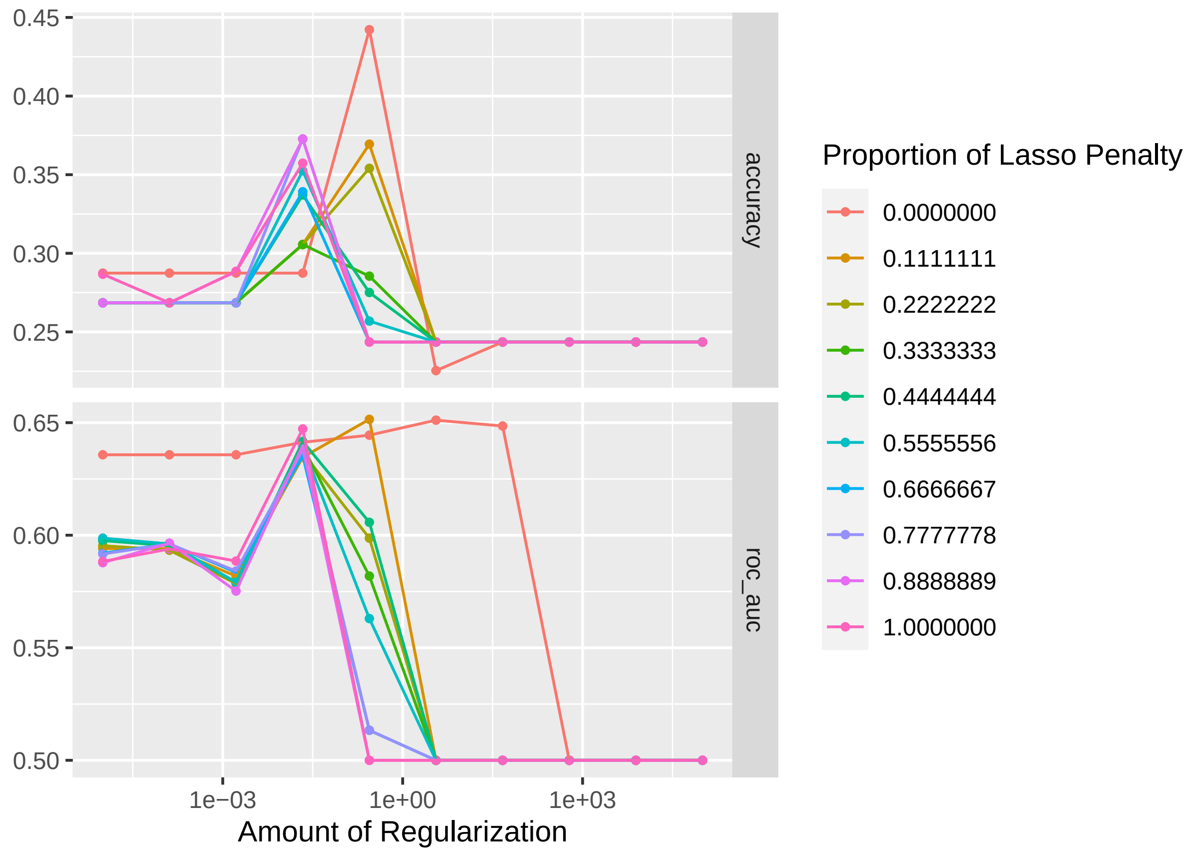
Exercise 6

```
tune_res <-tune_grid(
  net_wkflw,
  resamples = pok_folds,
  grid = net_grid
)
```

```
write_rds(tune_res, file = 'SavedModels/pok_tuned_res.rds')
```

```
pok_tune_res <- read_rds(file = 'SavedModels/pok_tuned_res.rds')
```

```
autoplot(pok_tune_res)
```



Medium to smaller values of penalty, and smaller values of mixture, produced better accuracy and ROC AUC.

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.

```
best_mod <- select_best(pok_tune_res, metric = 'roc_auc')
pok_final <- finalize_workflow(net_wkflw, best_mod)
pok_final_fit <- fit(pok_final, data = pok_train)
```

```
## Warning: The following variables are not factor vectors and will be ignored:
## ‘generation‘
```

```
## Warning in lognet(xd, is.sparse, ix, jx, y, weights, offset, alpha, nobs, : one
## multinomial or binomial class has fewer than 8 observations; dangerous ground
```

```
augment(pok_final_fit, new_data = pok_test) %>%
  conf_mat(truth = type_1, estimate = .pred_class)
```

```
##             Truth
## Prediction Bug Fire Grass Normal Psychic Water
##    Bug       0    0     0      0       0     0
##     Fire       0    0     0      0       0     0
##     Grass      0    0     0      0       0     0
##     Normal     2    1     1      1       1     4
##     Psychic    0    0     0      1       0     0
##     Water      1    1     2      2       2     0
```

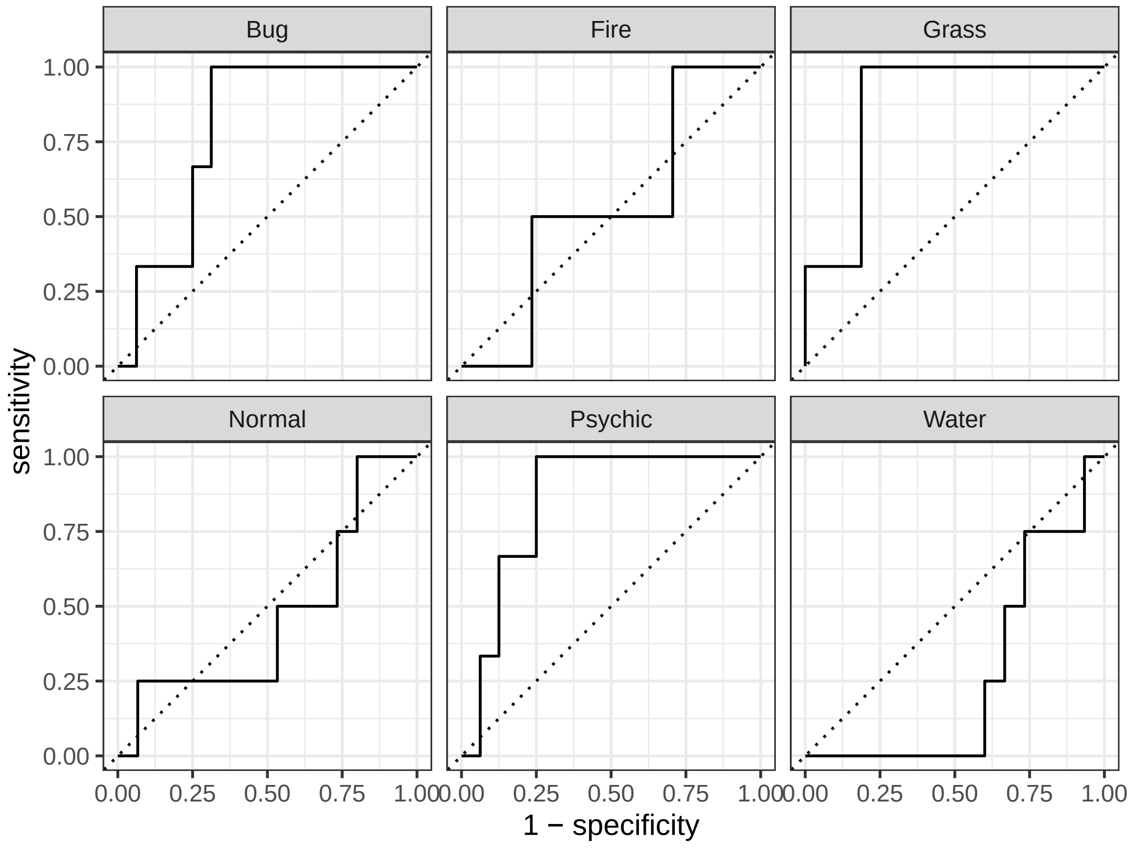
Exercise 8

```
predicted_data <- predict(pok_final_fit, new_data = pok_test, type = 'prob')
```

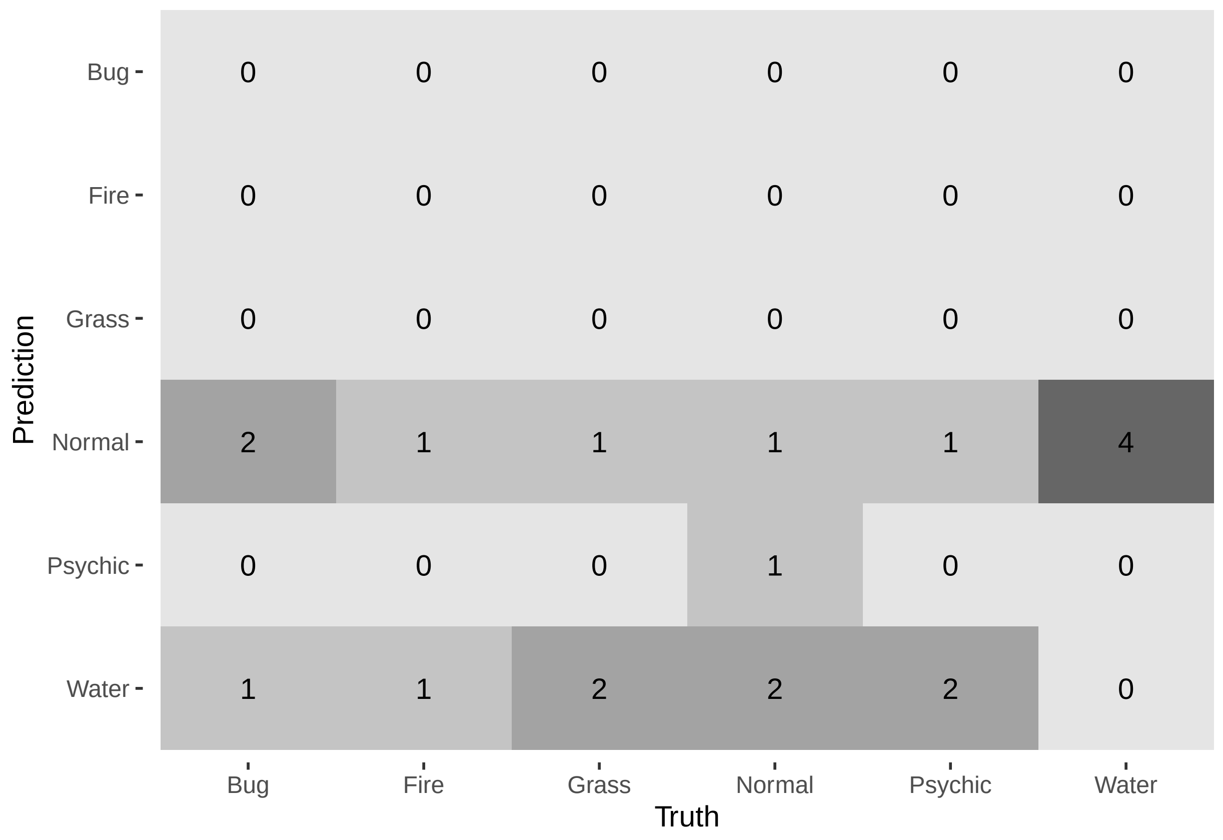
```
augment(pok_final_fit, new_data = pok_test) %>%
  roc_auc(type_1, .pred_Bug:.pred_Water)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.636
```

```
augment(pok_final_fit, new_data = pok_test) %>%
  roc_curve(type_1, .pred_Bug:.pred_Water) %>%
  autoplot()
```



```r
augment(pok_final_fit, new_data = pok_test) %>%
  conf_mat(truth = type_1, estimate = .pred_class) %>%
  autoplot(type = 'heatmap')
```



Prediction \ Truth	Bug	Fire	Grass	Normal	Psychic	Water
Bug	0	0	0	0	0	0
Fire	0	0	0	0	0	0
Grass	0	0	0	0	0	0
Normal	2	1	1	1	1	4
Psychic	0	0	0	1	0	0
Water	1	1	2	2	2	0

The overall ROC AUC is 0.636, which is not much better than a completely random model which would produce 0.5. However, the model did much better than this for specific pokemon types based on the ROC curves. The model is best at predicting grass, bug, and psychic types, and is worst at predicting water, fire, and normal types. This might be because bug, grass, and psychic would have more unique features that distinguish them into these unconventional types. Fire, normal, and water seem like more general terms, and could have fewer distinguishing features as a result.