PSTAT 131 HW 5

Raymond Lee

2022-05-11

```
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 0.2.0 --

## v broom        0.8.0     v recipes      0.2.0
## v dials        0.1.1     v rsample      0.1.1
## v dplyr        1.0.8     v tibble       3.1.6
## v ggplot2      3.3.5     v tidyr        1.2.0
## v infer        1.0.0     v tune         0.2.0
## v modeldata    0.1.1     v workflows    0.2.6
## v parsnip      0.2.1     v workflowsets 0.2.1
## v purrr        0.3.4     v yardstick    0.0.9

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x purrr::discard() masks scales::discard()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x recipes::step()  masks stats::step()
## * Use tidymodels_prefer() to resolve common conflicts.
```

```
tidymodels_prefer()
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v readr   2.1.2     v forcats 0.5.1
## v stringr 1.4.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x readr::col_factor() masks scales::col_factor()
## x purrr::discard()    masks scales::discard()
## x dplyr::filter()     masks stats::filter()
## x stringr::fixed()    masks recipes::fixed()
## x dplyr::lag()        masks stats::lag()
## x readr::spec()       masks yardstick::spec()
```

```
library(janitor)
library(glmnet)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-4
```

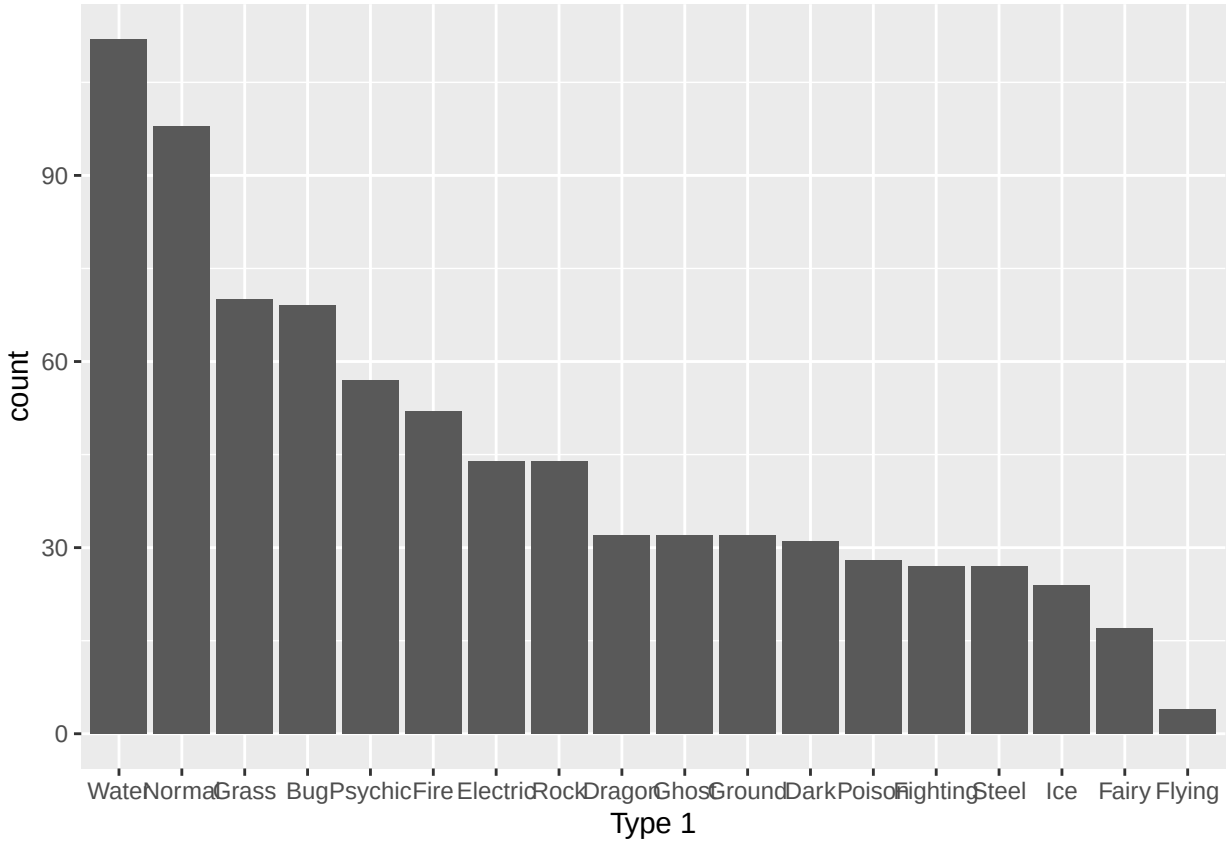
1.

```
pokemon = read.csv("pokemon.csv")
pokemon = clean_names(pokemon)
```

The names of the data frame were cleaned, resulting in names that are unique and consist of only underscores, numbers, and letters. This makes it easier to work with the dataset because we do not have to manually clean each column name, and the names are easier to read, recognize, and access.

2.

```
ggplot(pokemon, aes(x=fct_infreq(type_1))) + geom_bar() + labs(x = 'Type 1')
```



```
n_distinct(pokemon['type_1'])
```

```
## [1] 18
```

There are 18 type 1s. Fairy and flying types have the least amount of pokemon.

```
pokemon = filter(pokemon, type_1 == 'Bug' | type_1 == 'Fire' | type_1 == 'Grass' | type_1 == 'Normal' |
                   type_1 == 'Water' | type_1 == 'Psychic')
pokemon$type_1 = factor(pokemon$type_1, levels = c('Bug', 'Fire', 'Grass', 'Normal', 'Water', 'Psychic'
pokemon$legendary = factor(pokemon$legendary, levels = c('True', 'False'))
```

3.

```
set.seed(1114)
pokemon_split = initial_split(pokemon, prop = .70, strata = type_1)
pokemon_train = training(pokemon_split)
pokemon_test = testing(pokemon_split)

pokemon_folds = vfold_cv(pokemon_train, v = 5, strata = type_1)
```

Stratifying the folds may be useful when there is an unbalanced amount of data available for each level of a variable. This ensures that the folds are similar to each other in terms of the kinds of data that they consist of.

4.

```
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())
```

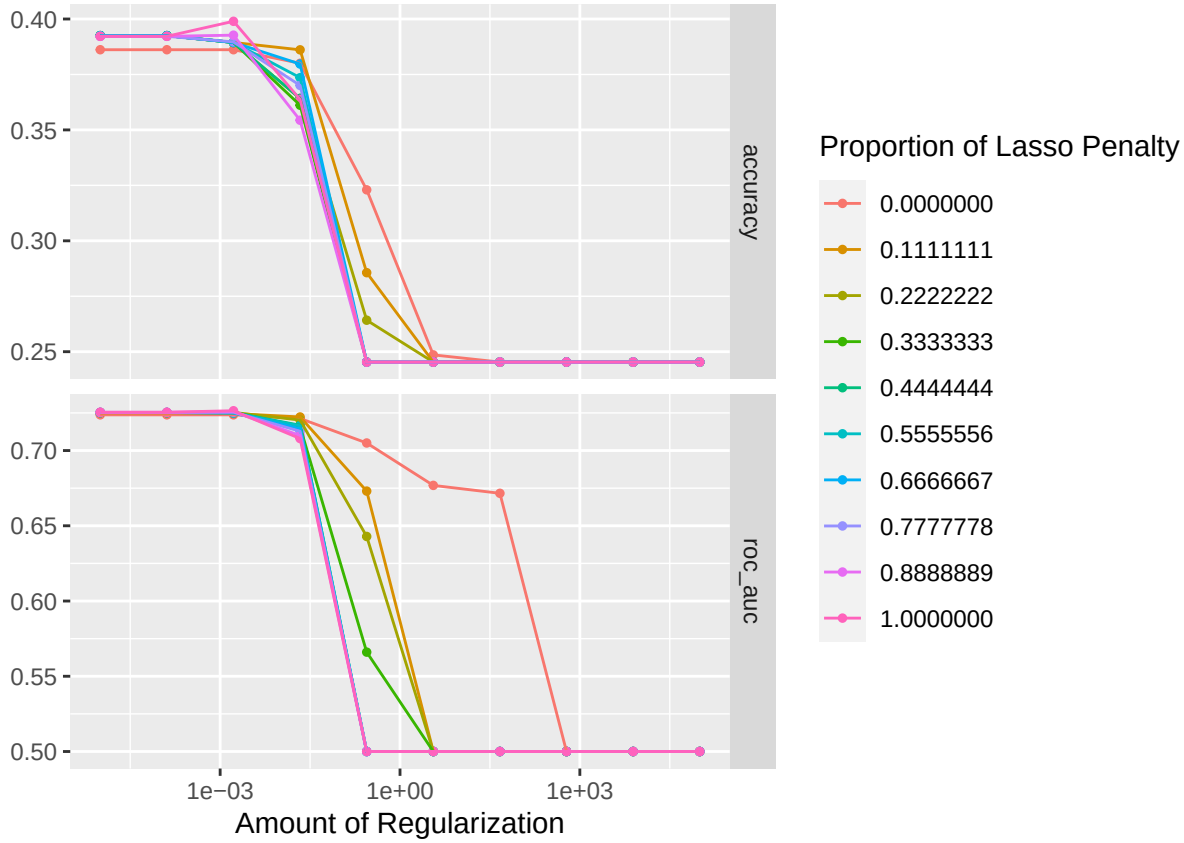
5.

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

en_wf = workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(elastic_net)

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

We will be fitting 500 models to the folded data.

6.

```
tune_res = tune_grid(en_wf, resamples = pokemon_folds, grid = en_grid)
autoplot(tune_res)
```



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

7.

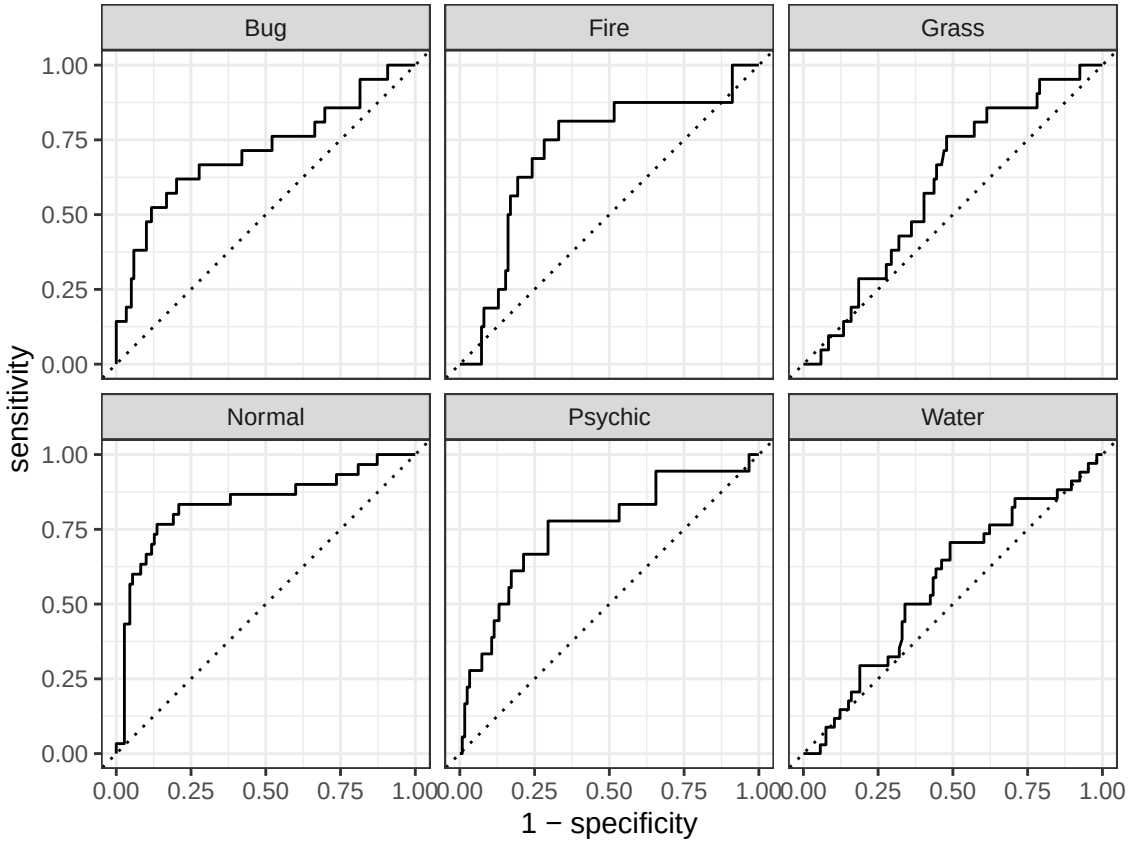
```
best_model = select_best(tune_res, metric = 'roc_auc')
en_final = finalize_workflow(en_wf, best_model)
en_final_fit = fit(en_final, data = pokemon_train)
predicted_data = augment(en_final_fit, new_data = pokemon_test) %>%
  select(type_1, starts_with('.pred'))
```

8.

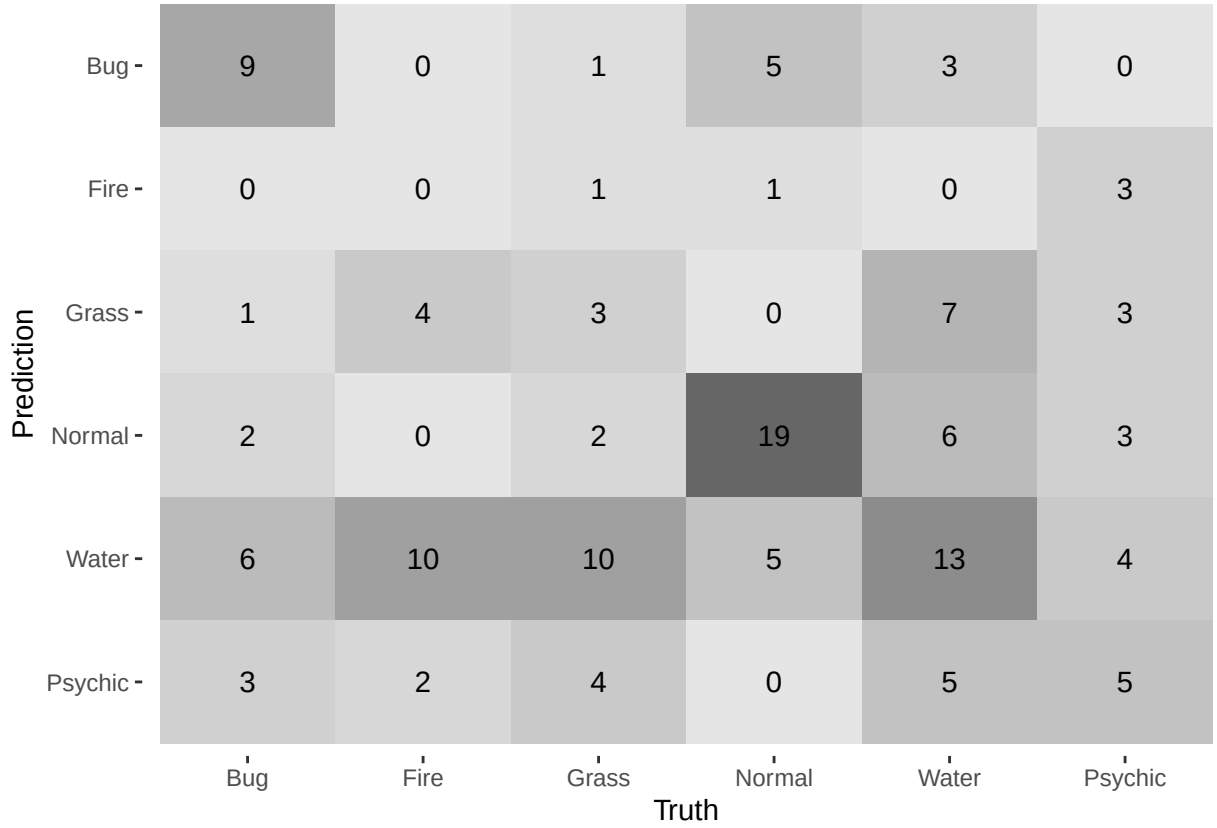
```
predicted_data %>% roc_auc(type_1, .pred_Bug:.pred_Psychic)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.694
```

```
predicted_data %>% roc_curve(type_1, .pred_Bug:.pred_Psychic) %>%
  autoplot()
```




```
predicted_data %>% conf_mat(truth = type_1, estimate = .pred_class) %>%
  autoplot(type = 'heatmap')
```



The model predicts normal types the best, but it predicts fire types the worst. Perhaps normal types have certain stats that are significantly different from other types in general while fire types have certain stats that are extremely similar to other types. It also seems like that the model tends to predict the type to be water regardless of the predictors.

```
#elastic net: combination of ridge and lasso
#range of penalties/norms that can be used
#mixture represents proportion of lasso and ridge penalty

#tuning: finding best value for input (like lambda)

#tune_grid tries out values for tuning
```