PSTAT 131 Homework 5

Tammy Truong

2022-05-15

```
pokemon <- read_csv("Pokemon.csv")
```

```
## Rows: 800 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Name, Type 1, Type 2
## dbl (9): #, Total, HP, Attack, Defense, Sp. Atk, Sp. Def, Speed, Generation
## lgl (1): Legendary
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Elastic Net Tuning

Question 1

We install and use `janitor` to utilize the function `clean_names()` to standardize the dataset.

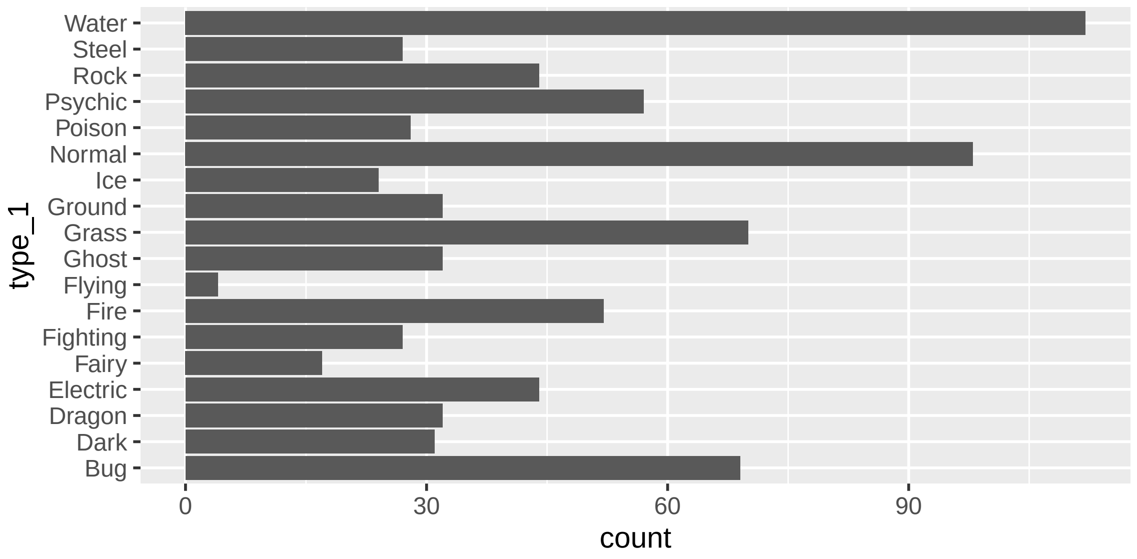
```
library(janitor)

pokemon <- clean_names(pokemon)
```

After using the `clean_names()` function, the data has changed the variables to lowercase and have added under scores such as "Type.1" to "type_1." This function is useful because it helps us to easily identify variables and standardize the whole data set.

Question 2

Using the entire data set, we create a bar chart of the outcome variable, `type_1`.

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



From the above results, there are 18 classes of the outcome. We see that there are very few flying type Pokemons.

We now filter the data set to contain only Pokemon whose `type_1` is Bug, Fire, Grass, Normal, Water, or Psychic.

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

After filtering, convert `type_1` and `legendary` to factors.

```
pokemon $ type_1 <- as.factor(pokemon $ type_1)
pokemon $ legendary <- as.factor(pokemon $ legendary)
pokemon $ generation <- as.factor(pokemon $ generation) # TA's recommendation
```

Question 3

Performing an initial split of the data and stratifying by the outcome variable.

```
set.seed(1004)
pokemon_split <- initial_split(pokemon, prop = 0.80, strata = type_1)
pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)

# verifying training and testing sets with desired number of obs.
nrow(pokemon_train)
```

```
## [1] 364
```

```
nrow(pokemon_test)
```

```
## [1] 94
```

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?

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

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 x 2
##   splits           id
##   <list>           <chr>
## 1 <split [289/75]> Fold1
## 2 <split [291/73]> Fold2
## 3 <split [291/73]> Fold3
## 4 <split [292/72]> Fold4
## 5 <split [293/71]> Fold5
```

Stratifying the folds may be useful because it helps the same ratios be consistent throughout each fold.

Question 4

Setting up a recipe to predict `type_1` with `legendary`, `generation`, `sp_atk`, `attack`, `speed`, `defense`, `hp`, and `sp_def`, including:

- Dummy-code `legendary` and `generation`;
- Center and scale all predictors.

```
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_
  step_dummy(all_nominal_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())
```

Question 5

We'll be fitting and tuning an elastic net, tuning `penalty` and `mixture` (use `multinom_reg` with the `glmnet` engine).

We set up this model and workflow by creating a regular grid for `penalty` and `mixture` with 10 levels each; `mixture` should range from 0 to 1. We'll let `penalty` range from -5 to 5 (it's log-scaled).

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

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

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

When we fit these models to the folded data, we have 100 models per fold, so a total of 500 models.

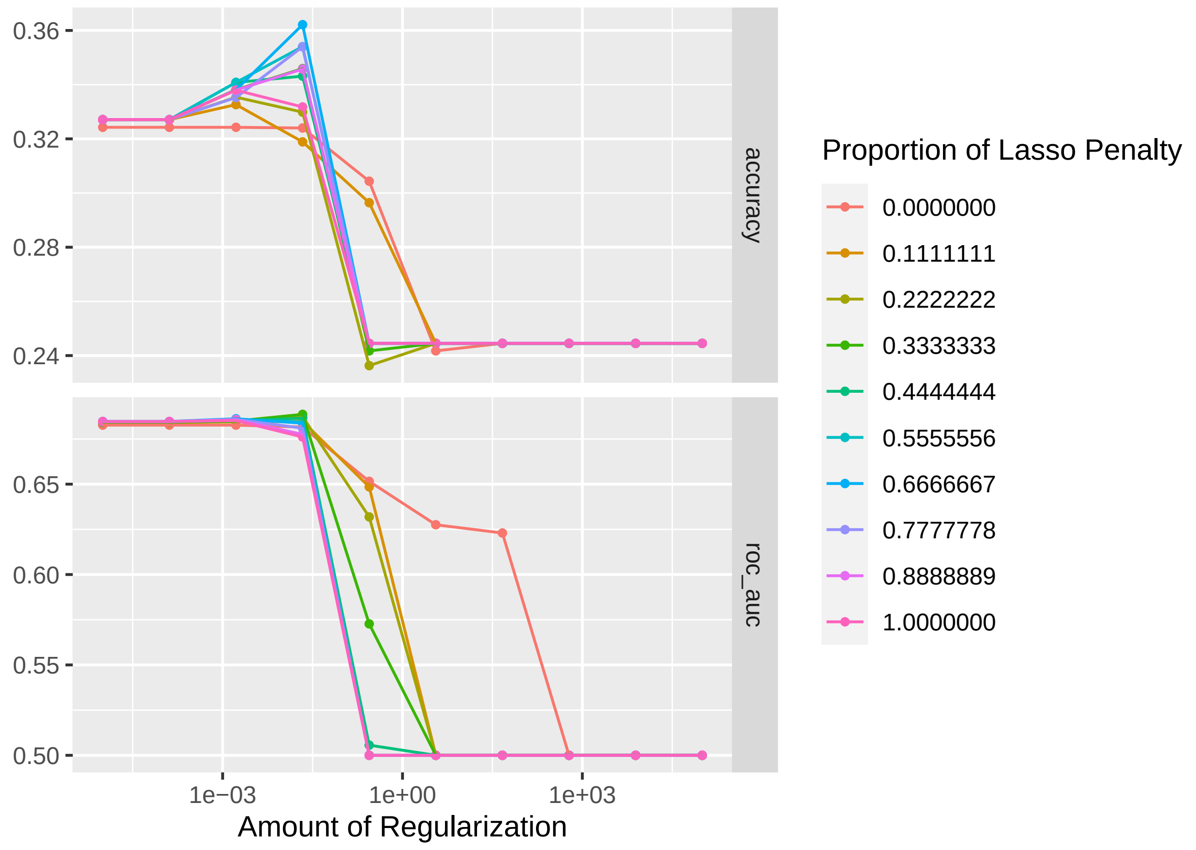
Question 6

Fit the models to the folded data using `tune_grid()`.

```
tune_pokemon <- tune_grid(pokemon_wkflow,
                          resamples = pokemon_folds,
                          grid = pokemon_grid)
```

We use `autoplot()` to check the results.

```
autoplot(tune_pokemon)
```



From the plot above, we notice that smaller values of penalty and mixture lead to a better accuracy and ROC AUC.

Question 7

Using `select_best()` to choose the model that has the optimal `roc_auc`. Then we use `finalize_workflow()`, `fit()`, and `augment()` to fit the model to the training set and evaluate its performance on the testing set.

```
pokemon_best <- select_best(tune_pokemon, metric = "roc_auc")
pokemon_best
```

```
## # A tibble: 1 x 3
##   penalty mixture .config
##     <dbl>   <dbl> <chr>
## 1  0.0215   0.333 Preprocessor1_Model034
```

```
final_wkflow <- pokemon_wkflow %>%
  finalize_workflow(pokemon_best) %>%
  fit(pokemon_train) %>%
  augment(pokemon_test)

accuracy(final_wkflow, truth = type_1, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.404
```

Question 8

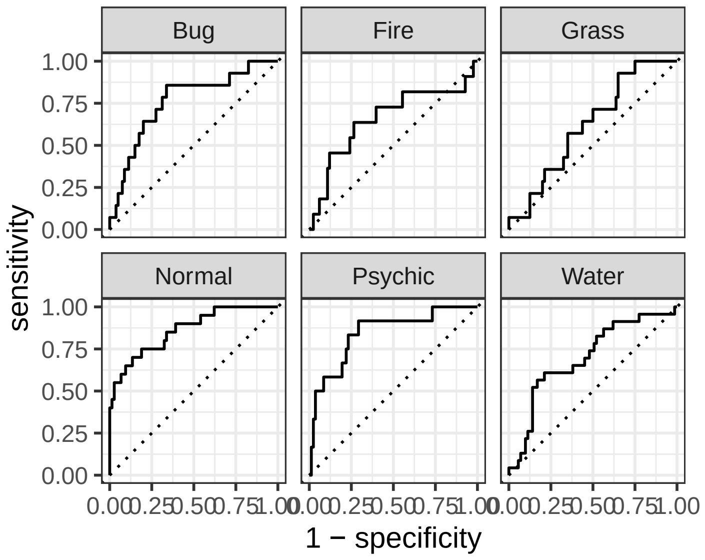
Calculating the overall ROC AUC on the testing set.

```
roc_auc(final_wkflow, truth = type_1, estimate = c(.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .p
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.740
```

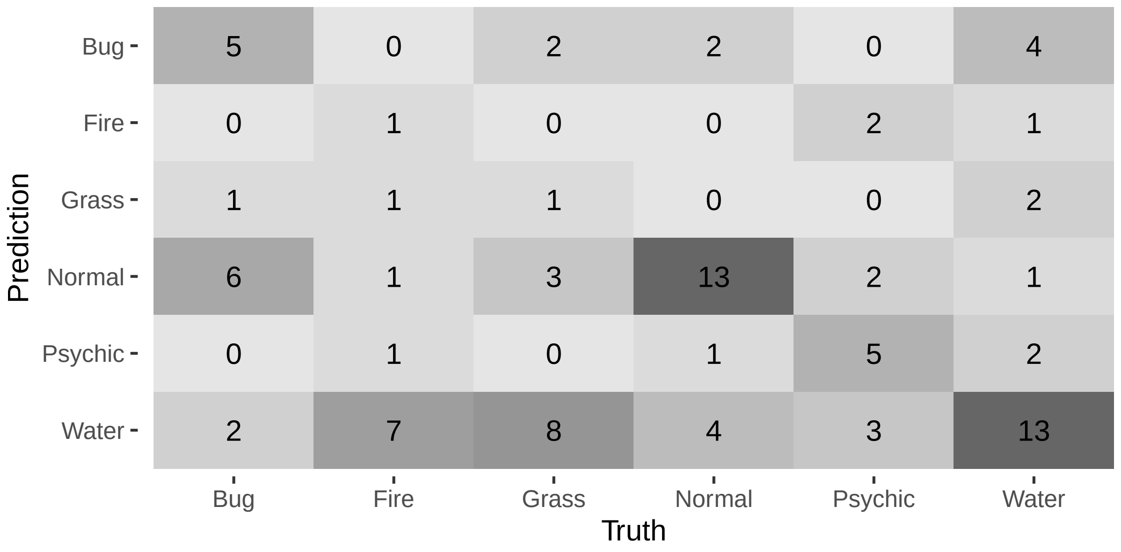
Plots of the different ROC curves, one per level of the outcome

```
autoplot(roc_curve(final_wkflow, truth = type_1, .pred_Bug, .pred_Fire,
                   .pred_Grass, .pred_Normal, .pred_Psychic, .pred_Water))
```



Heat map of the confusion matrix

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



Prediction \ Truth	Bug	Fire	Grass	Normal	Psychic	Water
Bug	5	0	2	2	0	4
Fire	0	1	0	0	2	1
Grass	1	1	1	0	0	2
Normal	6	1	3	13	2	1
Psychic	0	1	0	1	5	2
Water	2	7	8	4	3	13

Based on the results above, I notice that the overall ROC AUC yields an approximation of 0.7400513. When looking at the plots, there were some that were performed well, such as Psychic, Normal, and Bug. The model does not perform well for Grass and Water. When we look at the heat map of the confusion matrix, Normal and Water a had high positive counts of 13 for both. However, many performed poorly. Variables such as Fire and Grass only had a count of 1. Overall, I don't believe the model performed well.