Homework 5

PSTAT 131/231

Contents

Elastic Net Tuning

For this assignment, we will be working with the file `"pokemon.csv"`, found in `/data`. The file is from Kaggle: https://www.kaggle.com/abcsds/pokemon.

The Pokémon franchise encompasses video games, TV shows, movies, books, and a card game. This data set was drawn from the video game series and contains statistics about 721 Pokémon, or "pocket monsters." In Pokémon games, the user plays as a trainer who collects, trades, and battles Pokémon to (a) collect all the Pokémon and (b) become the champion Pokémon trainer.

Each Pokémon has a primary type (some even have secondary types). Based on their type, a Pokémon is strong against some types, and vulnerable to others. (Think rock, paper, scissors.) A Fire-type Pokémon, for example, is vulnerable to Water-type Pokémon, but strong against Grass-type.

Figure 1: Fig 1. Vulpix, a Fire-type fox Pokémon from Generation 1.

The goal of this assignment is to build a statistical learning model that can predict the **primary type** of a Pokémon based on its generation, legendary status, and six battle statistics.

Read in the file and familiarize yourself with the variables using `pokemon_codebook.txt`.

Exercise 1

Install and load the `janitor` package. Use its `clean_names()` function on the Pokémon data, and save the results to work with for the rest of the assignment. What happened to the data? Why do you think `clean_names()` is useful?

```
library(janitor)

Pokemon <- read_csv("D:/UCSB/Spring_2022/PSTAT 131/PSTAT_131_HW/HW2/PSTAT-131/homework-5/data/Pokemon.cs
Pokemon = clean_names(Pokemon)
```

Solution: Resulting names are unique and consist only of the _ character, numbers, and letters. Capitalization preferences can be specified using the case parameter. It standardizes the naming of column names so it can reduce confusion in the later analysis.

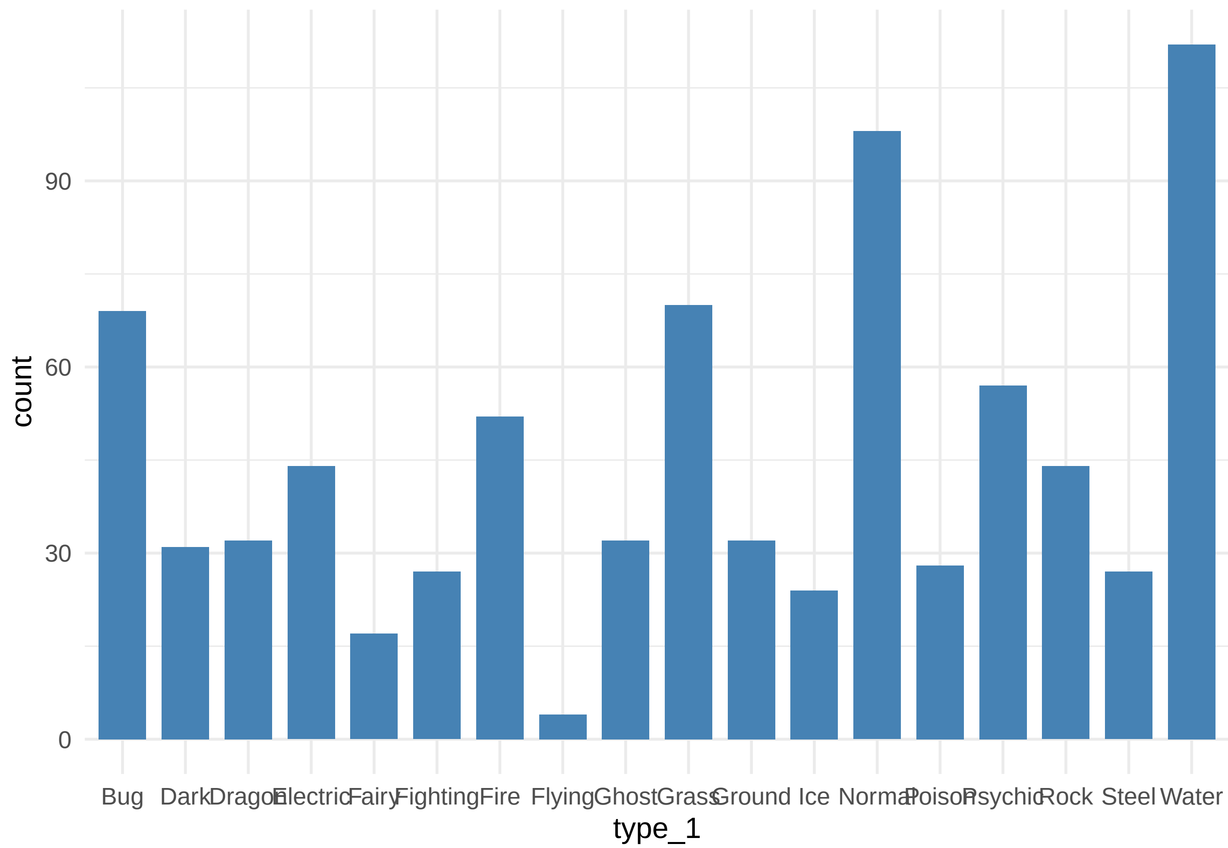
```
colnames(Pokemon)
```

```
##  [1] "number"     "name"       "type_1"     "type_2"     "total"
##  [6] "hp"         "attack"     "defense"    "sp_atk"     "sp_def"
## [11] "speed"      "generation" "legendary"
```

Exercise 2

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

```
ggplot(Pokemon, aes(type_1))+
  geom_bar( width=0.75, fill="steelblue") +
  theme_minimal()
```



How many classes of the outcome are there? Are there any Pokémon types with very few Pokémon? If so, which ones?

Solution: There are 18 classes of outcomes. The flying type has very few Pokemons.

For this assignment, we'll handle the rarer classes by simply filtering them out. Filter the entire data set to contain only Pokémon whose `type_1` is Bug, Fire, Grass, Normal, Water, or Psychic.

```
Pokemon_ra = Pokemon %>% filter(type_1 == "Bug" | type_1 == "Fire" |
                                type_1 == "Grass" | type_1 == "Grass" |
                                type_1 == "Normal" | type_1 == "Water" |
                                type_1 == "Psychic")
```

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

```
Pokemon_ra = Pokemon_ra %>%
  mutate(legendary = as.factor(legendary)) %>%
  mutate(type_1 = as.factor(type_1))

class(Pokemon_ra$legendary)
```

```
## [1] "factor"
```

```
class(Pokemon_ra$type_1)
```

```
## [1] "factor"
```

Exercise 3

Perform an initial split of the data. Stratify by the outcome variable. You can choose a proportion to use. Verify that your training and test sets have the desired number of observations.

```
Pokemon_ra_split = initial_split(Pokemon_ra, prop=0.7, strata = type_1)
typeI_train = training(Pokemon_ra_split)
typeI_test = testing(Pokemon_ra_split)

dim(typeI_train)
```

```
## [1] 318  13
```

```
dim(typeI_test)
```

```
## [1] 140  13
```

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?

Solution: With a strata argument, the random sampling is conducted within the stratification variable. This can help ensure that the resamples have equivalent proportions as the original data set. For a categorical variable, sampling is conducted separately within each class.

```
typeI_fold = vfold_cv(typeI_train, v=5, strata = type_1)
typeI_fold
```

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 x 2
##   splits           id
##   <list>           <chr>
## 1 <split [252/66]> Fold1
## 2 <split [253/65]> Fold2
## 3 <split [253/65]> Fold3
## 4 <split [256/62]> Fold4
## 5 <split [258/60]> Fold5
```

Exercise 4

Set up a recipe to predict `type_1` with `legendary`, `generation`, `sp_atk`, `attack`, `speed`, `defense`, `hp`, and `sp_def`.

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

```
typeI_recipe = recipe(type_1 ~ legendary + generation + sp_atk +
                      attack +  speed + defense + hp + sp_def,
                      data = typeI_train) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_predictors())
```

Exercise 5

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

Set up this model and workflow. Create a regular grid for `penalty` and `mixture` with 10 levels each; `mixture` should range from 0 to 1. For this assignment, we'll let `penalty` range from -5 to 5 (it's log-scaled).

How many total models will you be fitting when you fit these models to your folded data?

Solution: $10 \times 10 \times 5 = 500$ since we have 100 choices for regularization and 5 folds.

```
ridge_spec = multinom_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet")
```

```
# actually mean ridge to lasso
ridge_wkflw = workflow() %>%
  add_recipe(typeI_recipe) %>%
  add_model(ridge_spec)
```

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

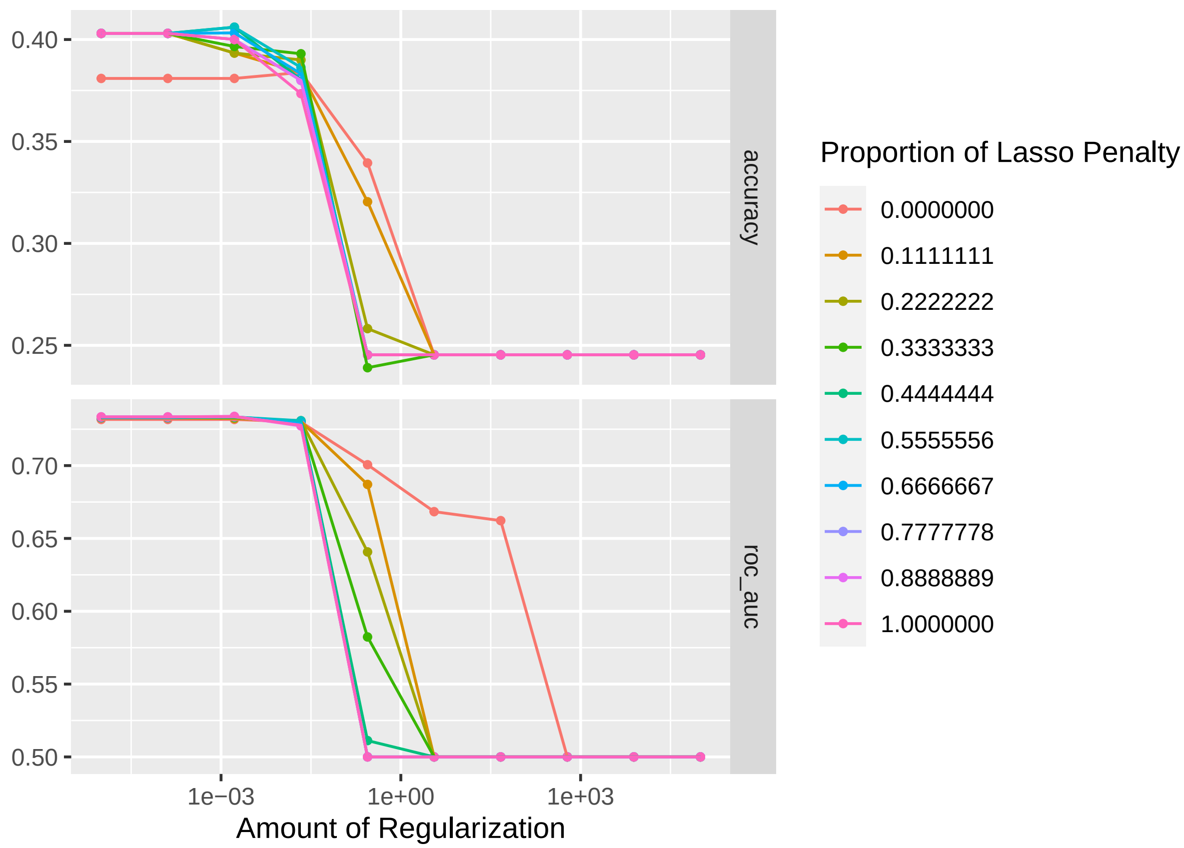
Exercise 6

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

Use `autoplot()` on the results. What do you notice? Do larger or smaller values of `penalty` and `mixture` produce better accuracy and ROC AUC?

```
tune_res <- tune_grid(
  ridge_wkflw,
  resamples = typeI_fold,
  grid = penalty_grid
)

autoplot(tune_res)
```



Solution: At the beginning, the accuracy remains high as the penalty increases, but it suddenly decreases when the penalty reaches the fifth level. Different values of mixture means different rates of decreasing. From the graph, smaller penalty and middle mixture produces 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(tune_res, metric = "roc_auc")
best_mod # showing the best penalty and mixture
```

```
## # A tibble: 1 x 3
##   penalty mixture .config
##     <dbl>   <dbl> <chr>
## 1 0.00167   0.889 Preprocessor1_Model083
```

```
final = finalize_workflow(ridge_wkflw, best_mod)
final_fit = fit(final, data = typeI_train)


final_test = augment(final_fit, new_data = typeI_test)
```

Exercise 8

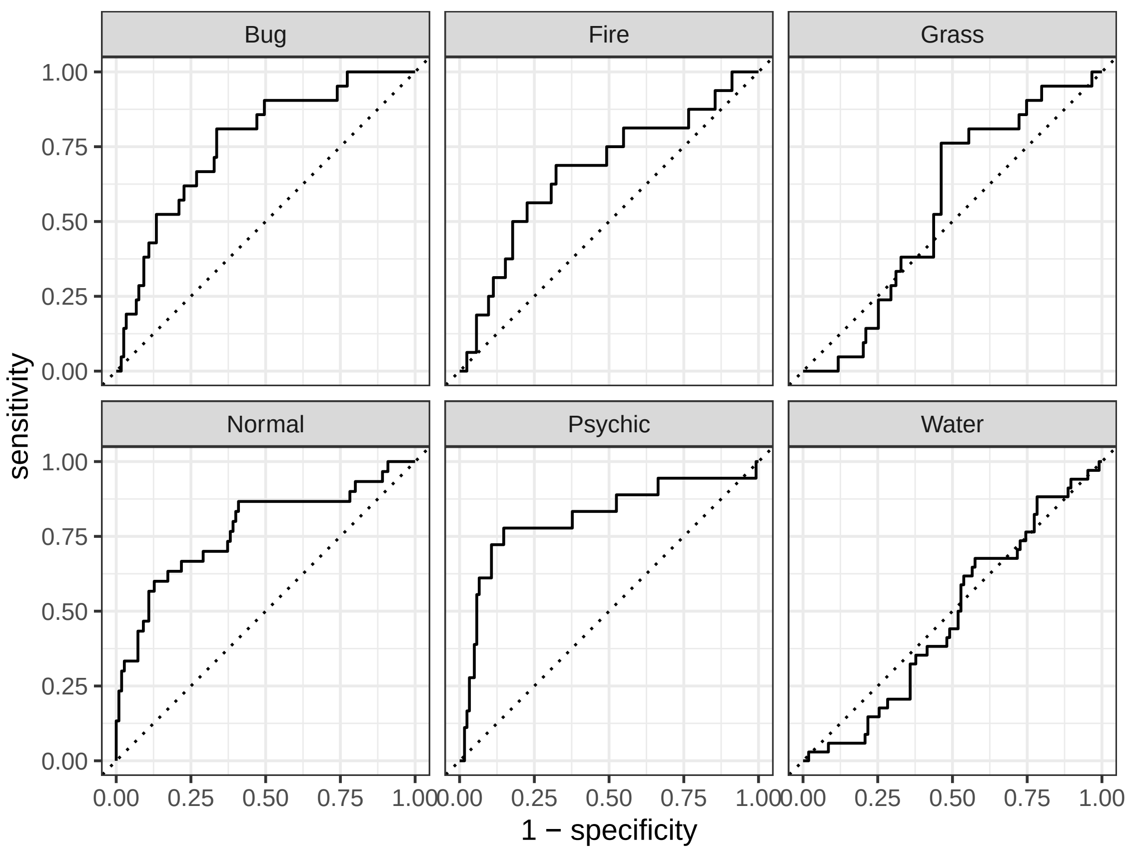
Calculate the overall ROC AUC on the testing set.

```
final_test %>% accuracy(truth = type_1, estimate = .pred_class)
```

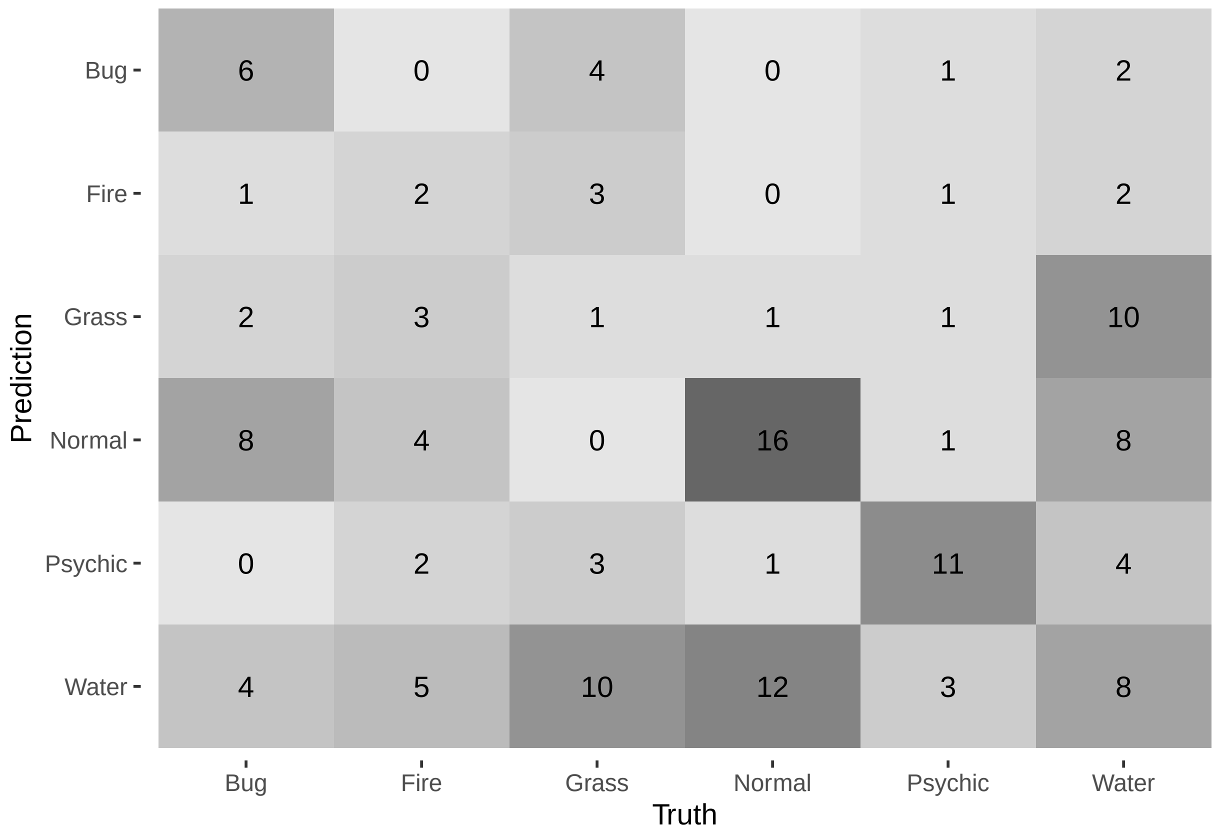
```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.314
```

Then create plots of the different ROC curves, one per level of the outcome. Also make a heat map of the confusion matrix.

```
# roc curve
final_test %>% roc_curve(type_1, .pred_Bug:.pred_Water) %>%
  autoplot()
```




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




```
final_test
```

```
## # A tibble: 140 x 20
##    number name      type_1 type_2 total    hp attack defense sp_atk sp_def speed
##     <dbl> <chr>     <fct>  <chr>  <dbl> <dbl>  <dbl>   <dbl>  <dbl>  <dbl> <dbl>
##  1      2 Ivysaur   Grass  Poison   405    60     62      63     80     80    60
##  2      3 Venusaur  Grass  Poison   525    80     82      83    100    100    80
##  3      6 Charizard Fire   Flying   534    78     84      78    109     85   100
##  4     10 Caterpie  Bug    <NA>     195    45     30      35     20     20    45
##  5     13 Weedle    Bug    Poison   195    40     35      30     20     20    50
##  6     15 Beedrill  Bug    Poison   395    65     90      40     45     80    75
##  7     18 Pidgeot   Normal Flying   479    83     80      75     70     70   101
##  8     18 PidgeotM~ Normal Flying   579    83     80      80    135     80   121
##  9     20 Raticate  Normal <NA>     413    55     81      60     50     70    97
## 10     22 Fearow    Normal Flying   442    65     90      65     61     61   100
## # ... with 130 more rows, and 9 more variables: generation <dbl>,
## #   legendary <fct>, .pred_class <fct>, .pred_Bug <dbl>, .pred_Fire <dbl>,
## #   .pred_Grass <dbl>, .pred_Normal <dbl>, .pred_Psychic <dbl>,
## #   .pred_Water <dbl>
```

What do you notice? How did your model do? Which Pokemon types is the model best at predicting, and which is it worst at? Do you have any ideas why this might be?

Solution: From the testing set, the model does not perform well since the accuracy is less than 0.5. Also, the prediction of each Pokemon type does not follow a similar pattern. The model is best at predicting *Normal*, and the worst at *Grass* after considering confusion matrix and roc curve. Possible explanation is that the model can not derive numerical features of a specific type due to lack of data and similarity between certain types. Also, we can increase the number of folds for cross-validation. On the other hand, we should try other models to reduce overfitting.

For 231 Students

Exercise 9

In the 2020-2021 season, Stephen Curry, an NBA basketball player, made 337 out of 801 three point shot attempts (42.1%). Use bootstrap resampling on a sequence of 337 1's (makes) and 464 0's (misses). For each bootstrap sample, compute and save the sample mean (e.g. bootstrap FG% for the player). Use 1000 bootstrap samples to plot a histogram of those values. Compute the 99% bootstrap confidence interval for Stephen Curry's "true" end-of-season FG% using the quantile function in R. Print the endpoints of this interval.