Homework 5

PSTAT 131/231

Contents

```
library(tidymodels)
library(tidyverse)
library(glmnet)
```

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('Pokemon.csv')
pokemon <- janitor:: clean_names(dat = pokemon)
head(pokemon)
```

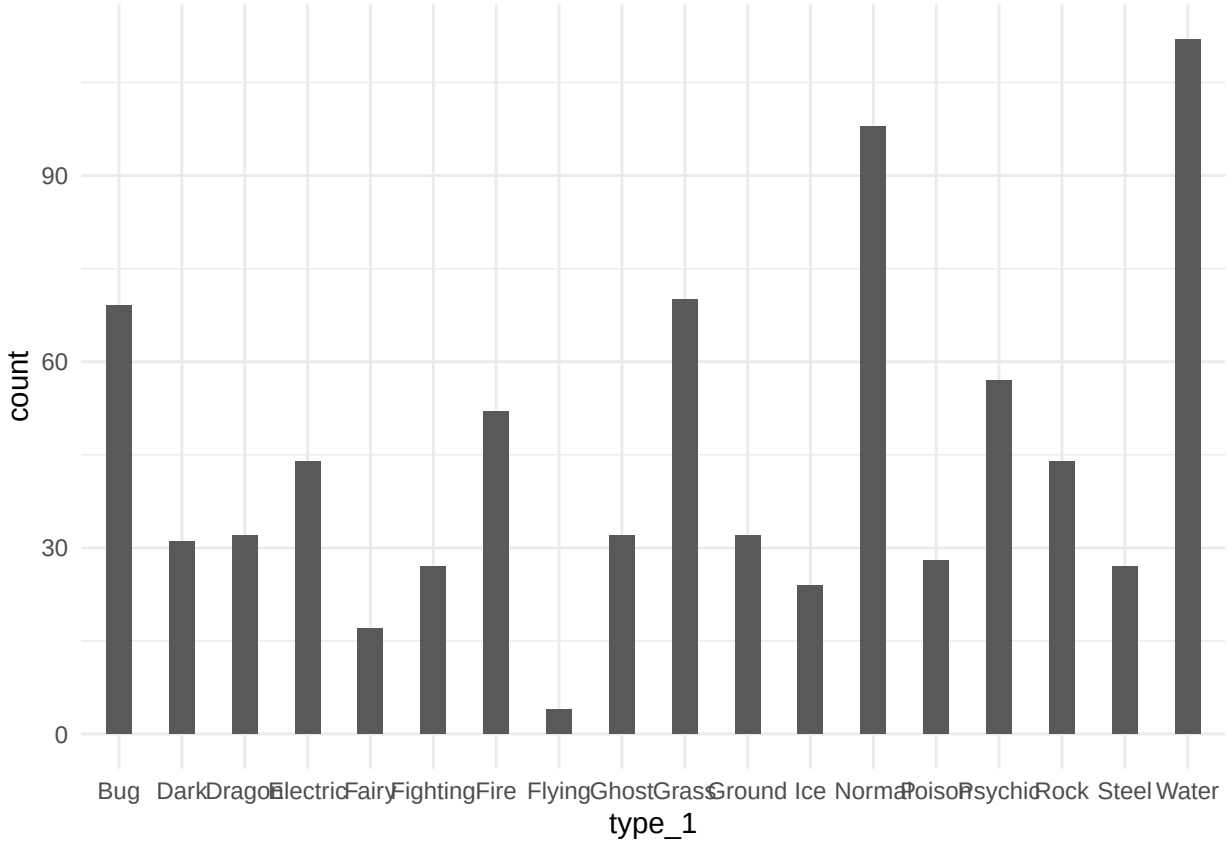
```
##   x                  name type_1 type_2 total hp attack defense sp_atk sp_def
## 1 1             Bulbasaur  Grass Poison   318 45     49      49     65     65
## 2 2               Ivysaur  Grass Poison   405 60     62      63     80     80
## 3 3              Venusaur  Grass Poison   525 80     82      83    100    100
## 4 3 VenusaurMega Venusaur  Grass Poison   625 80    100     123    122    120
## 5 4            Charmander   Fire          309 39     52      43     60     50
## 6 5            Charmeleon   Fire          405 58     64      58     80     65
##   speed generation legendary
## 1    45          1     False
## 2    60          1     False
## 3    80          1     False
## 4    80          1     False
## 5    65          1     False
## 6    80          1     False
```

Each space is replaced by an underscore character, and the first letter of each word written in lowercase. clean_names() function can help us to change the variable name to the snake case, and it is convenient for us to use in future process.

Exercise 2

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

```
ggplot(data = pokemon, aes(x = type_1)) +
  geom_bar(width = 0.4) +
  theme_minimal()
```



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

- There are 18 classes of the outcome
- There are pokemon types with very few pokemon, and three lowest types are flying,fairy,and ice

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.

```
pokemon1 <- pokemon  %>%
  filter(type_1  %in%  c('Bug','Fire', 'Grass', 'Normal', 'Water', 'Psychic'))

pokemon1 %>%
  count(type_1, sort = T)
```

```
##    type_1   n
## 1   Water 112
## 2  Normal  98
## 3   Grass  70
## 4     Bug  69
## 5 Psychic  57
## 6    Fire  52
```

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

```
pokemon2 <- pokemon1 %>%
  mutate(type_1 = factor(type_1),
         legendary = factor(legendary))
is.factor(pokemon2$type_1)
```

```
## [1] TRUE
```

```
is.factor(pokemon2$legendary)
```

```
## [1] TRUE
```

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.

```
set.seed(1979)
pokemon2_split <- initial_split(pokemon2, prop = 0.8,
                                strata = type_1)
pokemon2_train <- training(pokemon2_split)
pokemon2_test  <- testing(pokemon2_split)
c(dim(pokemon2_train), dim(pokemon2_test))
```

```
## [1] 364  13  94  13
```

The training sets have 364 observations and test sets have 94 observations. They both have the desired number of observations.

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?

```
pokemon2_folds <- vfold_cv(pokemon2_train, v = 5, strata = type_1)
```

Stratifying the folds ensure each fold is representative of all strata of the data. In other words, the distribution of types in each folds are approximately the same with the entire data set.

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.

```
simple_pokemon_recipe <-
  recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_def,
         data = pokemon2_train) %>%
  step_dummy(legendary, generation) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())
```

Exercise 5

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

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

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).

```
pokemon_workflow <- workflow() %>%
  add_recipe(simple_pokemon_recipe) %>%
  add_model(pokemon_spec)
```

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

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

500 models will be fitting when I fit these models to your folded data.

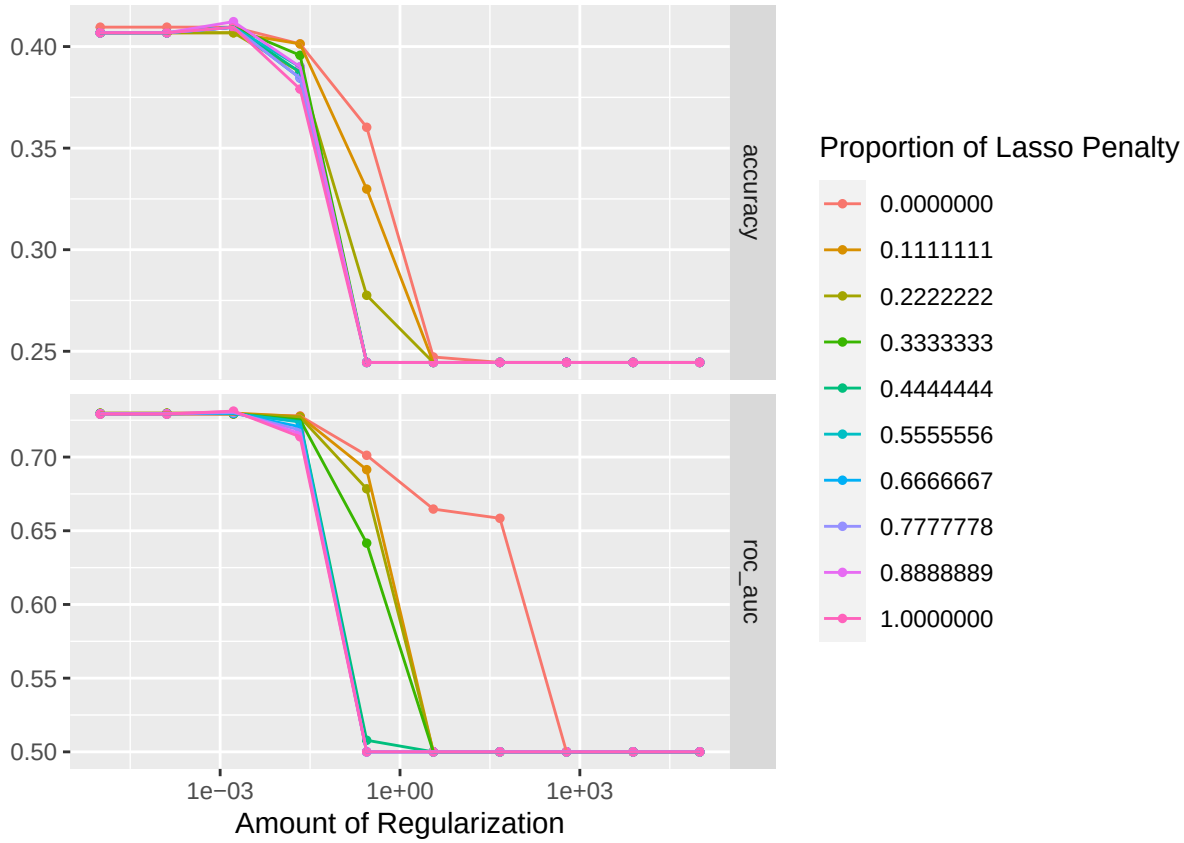
Exercise 6

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

```
tune_res <- tune_grid(
  pokemon_workflow,
  resamples = pokemon2_folds,
  grid = pokemon_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?

```
autoplot(tune_res)
```



As proportion of Lasso penalty increase, accracy and ROC AUC decrease Larger or smaller values of penalty and mixture do not produce 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_penalty <- select_best(tune_res, metric = "roc_auc")
best_penalty
```

```
## # A tibble: 1 x 3
##   penalty mixture .config
##     <dbl>   <dbl> <chr>
## 1 0.00167       1 Preprocessor1_Model093
```

```
pokemon_final <- finalize_workflow(pokemon_workflow, best_penalty)
pokemon_final_fit <- fit(pokemon_final, data = pokemon2_train)
augment(pokemon_final_fit, new_data = pokemon2_test) %>%
  accuracy(truth = type_1, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.245
```

Exercise 8

Calculate the overall ROC AUC on the testing set.

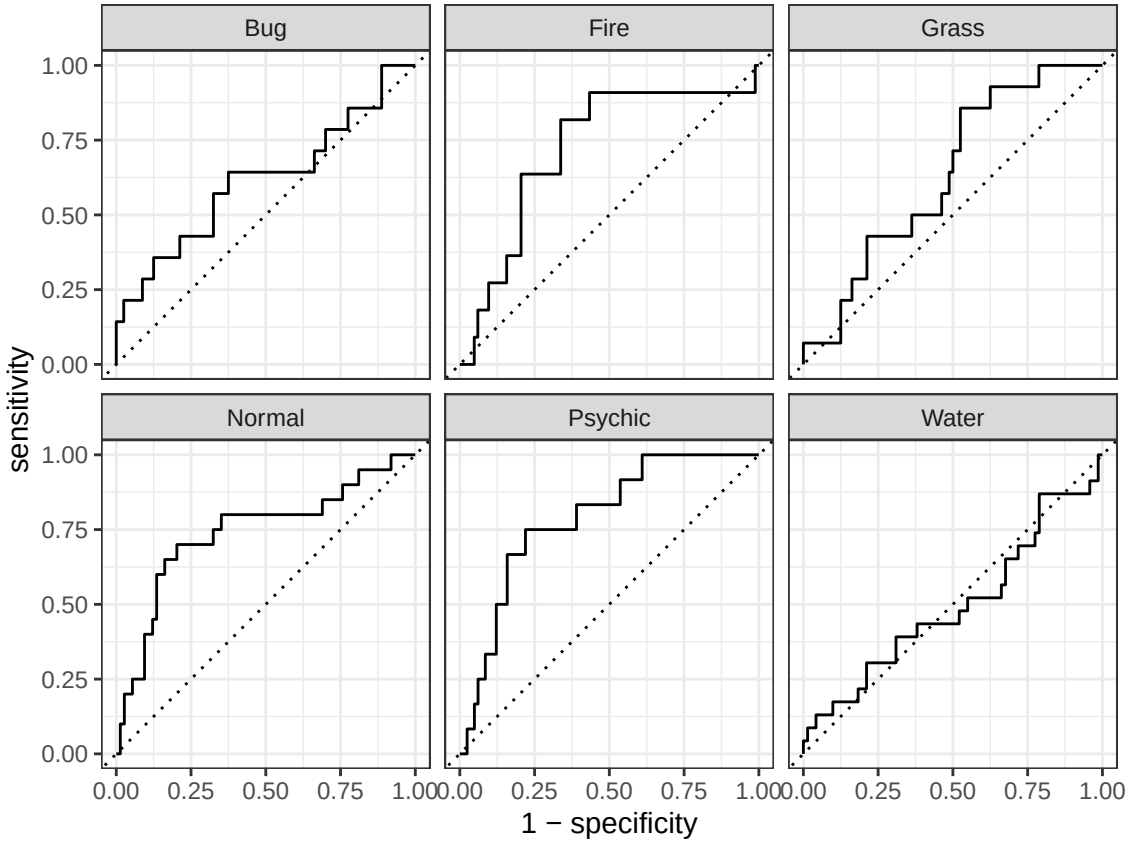
```
pred_result <- augment(pokemon_final_fit, new_data = pokemon2_test) %>%
  select(type_1, .pred_class,.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic,.pred_Wate
```

```
roc_auc(pred_result,type_1,.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic,.pred_Water)
```

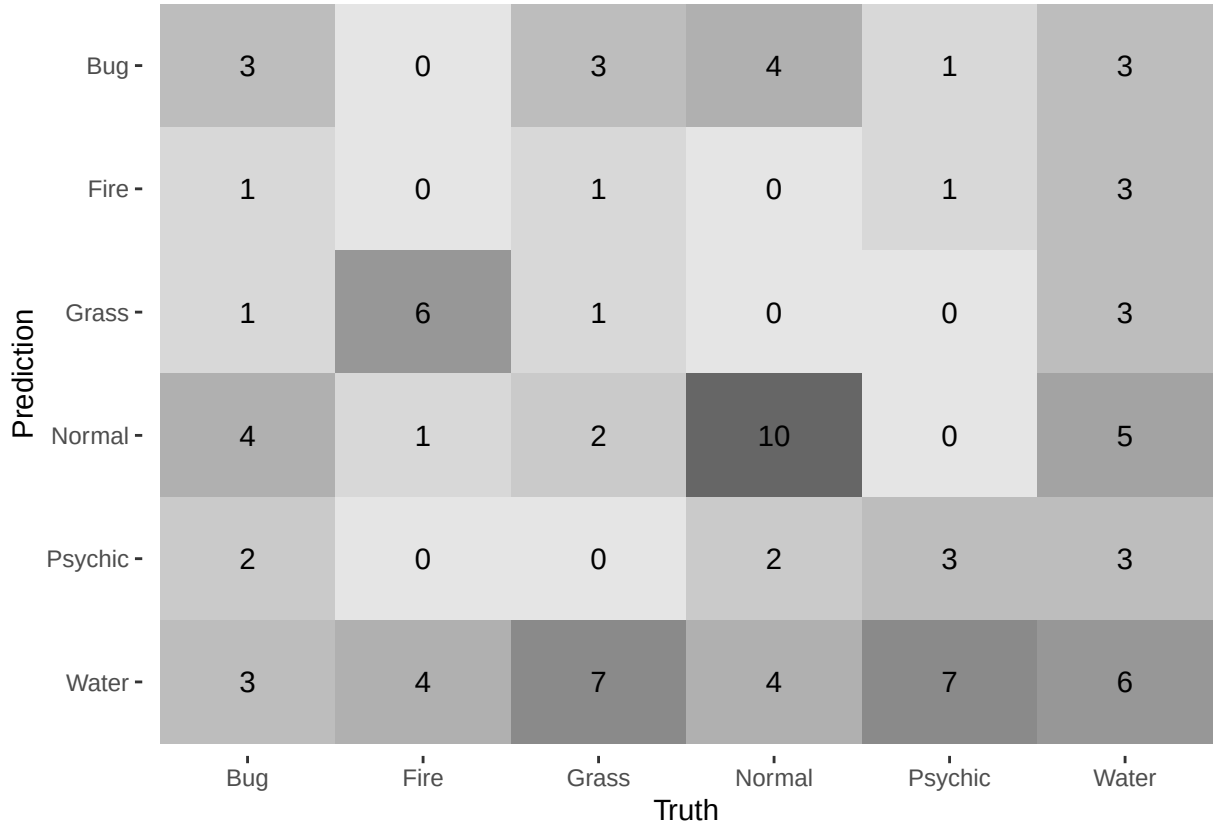
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.661
```

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

```
pred_result %>%
  roc_curve(type_1,.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic,.pred_Water) %>% aut
```




```
pred_result %>%
  conf_mat(type_1, .pred_class) %>%
  autoplot(type = "heatmap")
```



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?

The model dose not have a great performance on Pokemon type prediction with 0.66 roc_auc. Also, the overall accuracy is 0.24468. The psychic is the model best at predicting. The water is the model worst at predicting on. The reason might be size of each type is vary, for example water has 112 observations and fire has 52 observations.