Homework 5

PSTAT 131/231

Contents

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 <- clean_names(pokemon)
pokemon
```

```
## # A tibble: 800 x 13
##    number name      type_1 type_2 total    hp attack defense sp_atk sp_def speed
##     <dbl> <chr>     <chr>  <chr>  <dbl> <dbl>  <dbl>   <dbl>  <dbl>  <dbl> <dbl>
##  1      1 Bulbasaur Grass  Poison   318    45     49      49     65     65    45
##  2      2 Ivysaur   Grass  Poison   405    60     62      63     80     80    60
##  3      3 Venusaur  Grass  Poison   525    80     82      83    100    100    80
##  4      3 Venusaur~ Grass  Poison   625    80    100     123    122    120    80
##  5      4 Charmand~ Fire   <NA>     309    39     52      43     60     50    65
##  6      5 Charmele~ Fire   <NA>     405    58     64      58     80     65    80
##  7      6 Charizard Fire   Flying   534    78     84      78    109     85   100
##  8      6 Charizar~ Fire   Dragon   634    78    130     111    130     85   100
##  9      6 Charizar~ Fire   Flying   634    78    104      78    159    115   100
## 10      7 Squirtle  Water  <NA>     314    44     48      65     50     64    43
## # ... with 790 more rows, and 2 more variables: generation <dbl>,
## #   legendary <lgl>
```

The data was "cleaned" or formatted so that everything is consistent. For example, all the variable names became lower case and the periods/spaces were replaced with underscores. This is likely useful to catch discontinuities that might make your coding more difficult. Additionally, referencing a variable name with an underscore is easier than one with a space.

Exercise 2

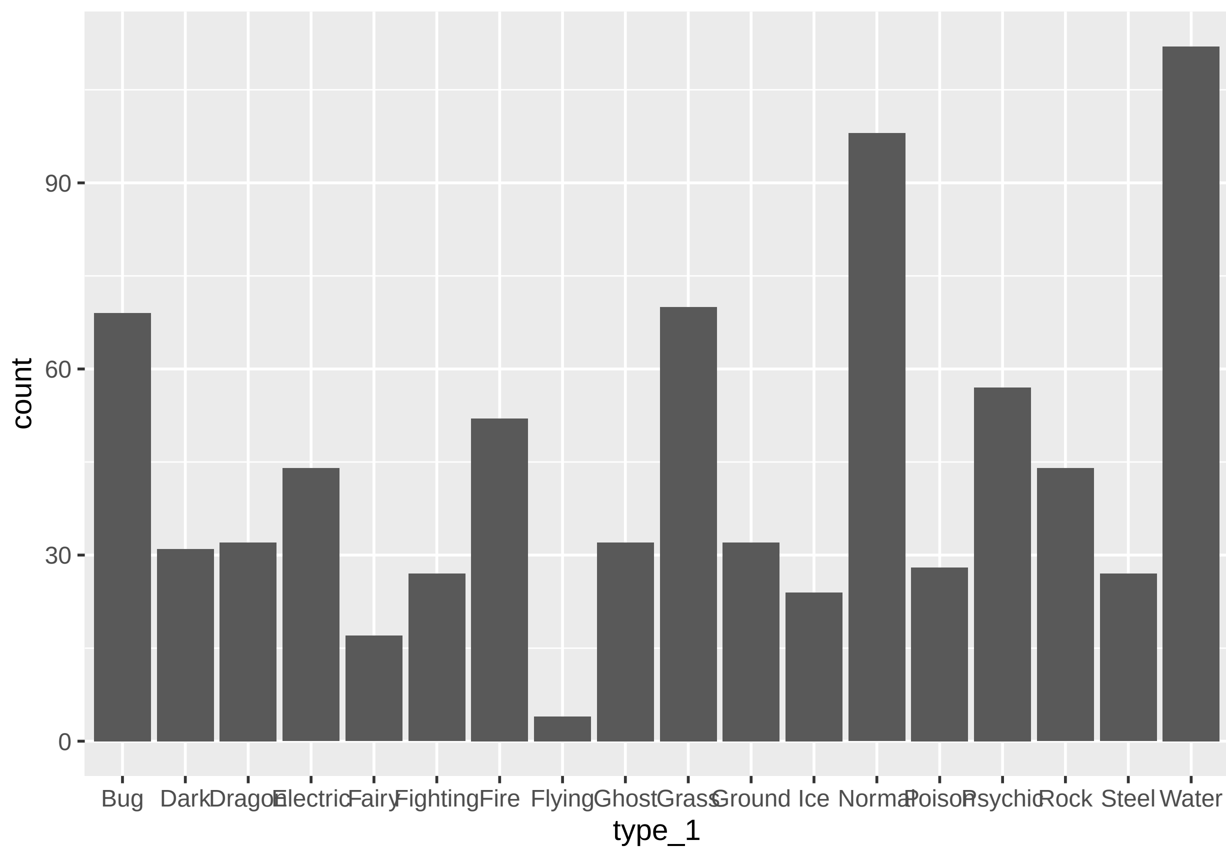
Using the entire data set, create a bar chart of the outcome variable, `type_1`.

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

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.

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

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




```
length(unique(pokemon$type_1))
```

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

```
type_variables <- c("Bug",'Fire', 'Grass',"Normal","Water","Psychic")
pokemon_edit <- filter(pokemon, pokemon$type_1 %in% type_variables)
pokemon_edit$type_1<-as.factor(pokemon_edit$type_1)
pokemon_edit$legendary<-as.factor(pokemon_edit$legendary)
pokemon_edit$generation<-as.factor(pokemon_edit$generation)
```

There are 18 unique classes of the outcome and there seems to be few flying and fairy types.

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.

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?

```r
pokemon_split <- initial_split(pokemon_edit, strata = type_1, prop = 0.8)
pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)
pokemon_folds <- vfold_cv(pokemon_train, v = 5,strata=type_1)
```

Stratified sampling for a v fold cross-validation is good for ensuring that the samples have an equal representation of of the outcome variable in the dataset.

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.

```r
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_
  step_dummy(generation) %>%
  step_dummy(legendary) %>%
  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?

```r
penalty_mixture_grid <- grid_regular(penalty(range=c(-5,5)), mixture(range=c(0,1)),levels = 10)
spec <- multinom_reg(penalty = tune(), mixture = tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")
workflow <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(spec)
```

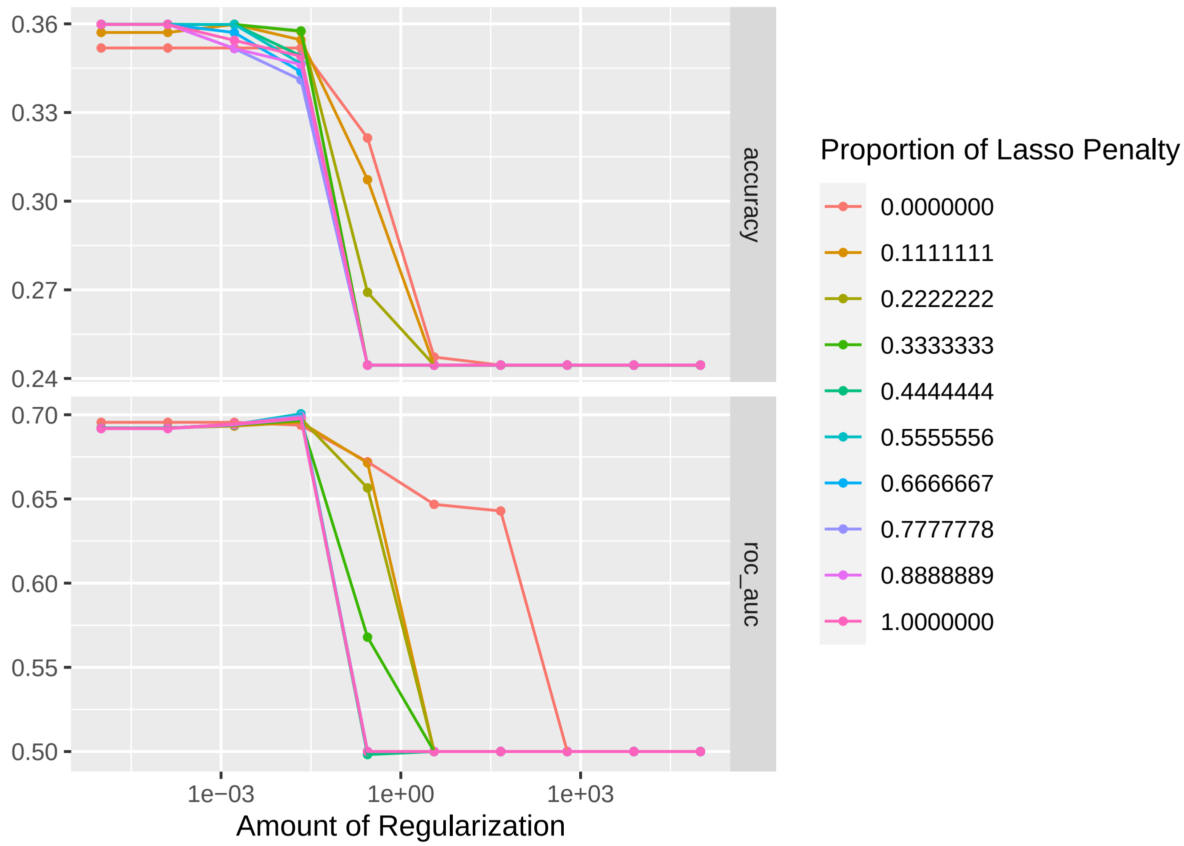
Because there are 10 levels for both the penalty and mixture, and 5 folds, there are a total of 500 different models folded to the model.

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(
  workflow,
  resamples = pokemon_folds,
  grid = penalty_mixture_grid)
autoplot(tune_res)
```



Smaller penalty and mixture values result in higher accuracy. When referencing the graph, you can see this through the smaller proportions maintaining a higher accuracy for a larger amount of regularization.

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")
ridge_final <- finalize_workflow(workflow, best_penalty)
ridge_final_fit <- fit(ridge_final, data = pokemon_train)
prediction_pokemon <- augment(ridge_final_fit, new_data = pokemon_test)
```

Exercise 8

Calculate the overall ROC AUC on the testing set.

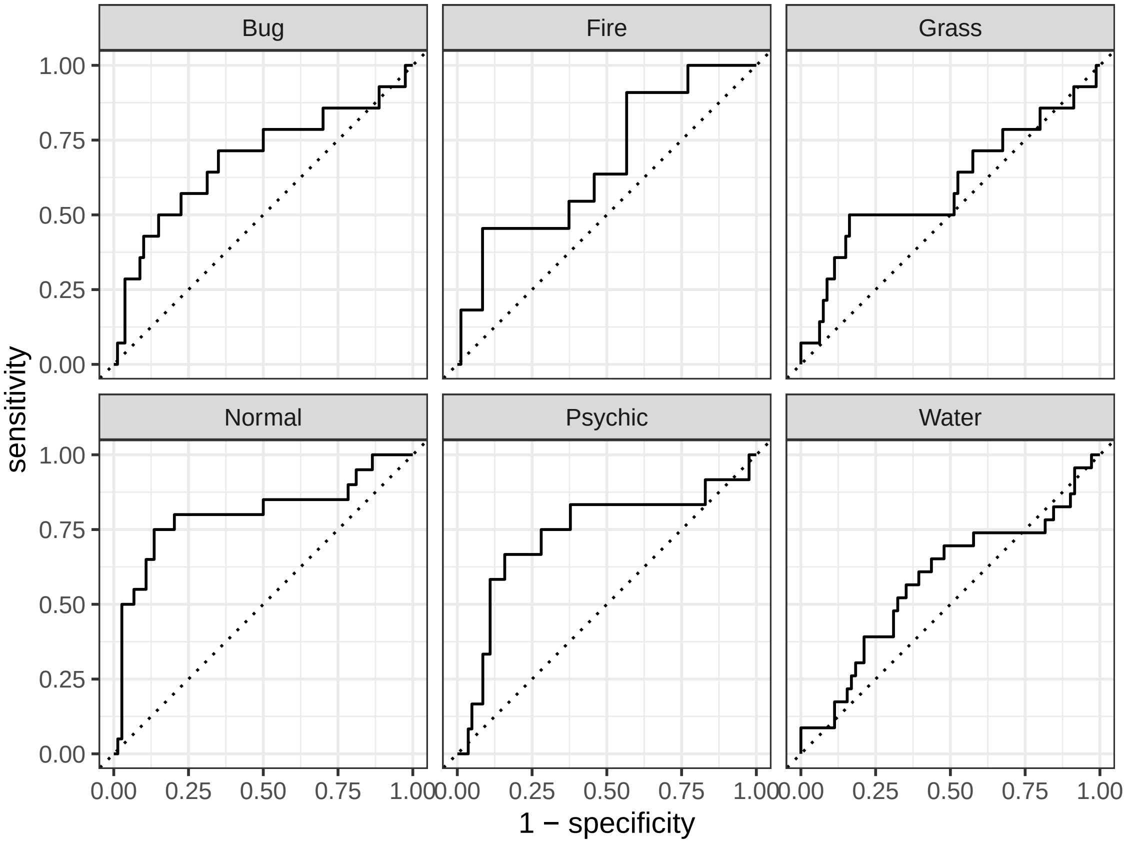
Then create plots of the different ROC curves, one per level of the outcome. Also make a heat map of the confusion matrix.

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?

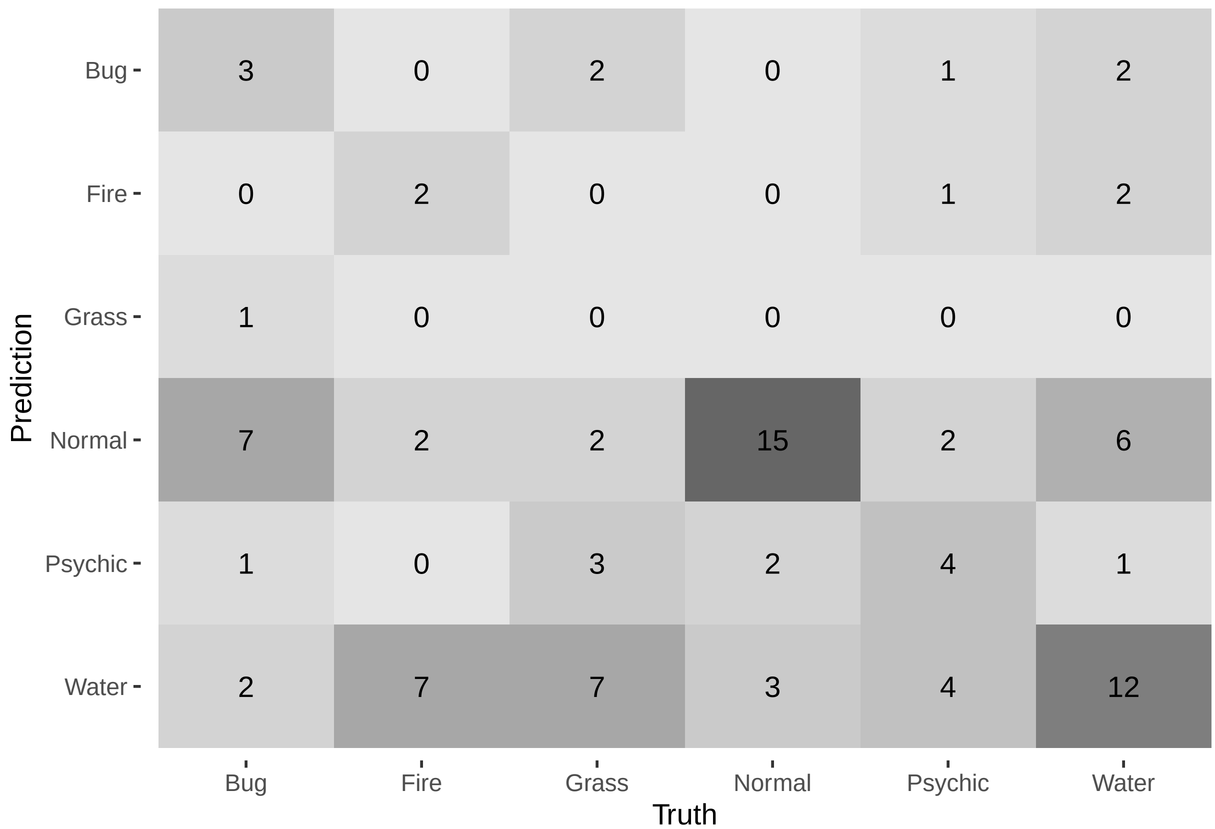
```
prediction_pokemon %>%
  roc_auc(type_1,.pred_Bug,.pred_Fire,.pred_Grass,.pred_Normal,.pred_Psychic,.pred_Water)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.676
```

```
prediction_pokemon %>%
  roc_curve(type_1,.pred_Bug,.pred_Fire,.pred_Grass,.pred_Normal,.pred_Psychic,.pred_Water) %>% autoplo
```




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



The model performed exceptionally well with around a 67.6% accuracy. According to the ROC curves, normal and psychic performed significantly better than the rest of the other types. This can be due to a variety of different factors; for example, perhaps psychic and normal type have specific stats that distinguish them more than the other types(like normal types having an unusually high amount of health). This may also be partially attributed to the sample size of certain types.