PSTAT131_HW5

Tao Wang

2022-05-11

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(tidymodels)
```

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

## v broom        0.7.12     v recipes      0.2.0
## v dials        0.1.0      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()
## * Dig deeper into tidy modeling with R at https://www.tmwr.org
```

```
library(ISLR)
library(ISLR2)
```

```
##
## Attaching package: 'ISLR2'

## The following objects are masked from 'package:ISLR':
##
##     Auto, Credit
```

```
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()
```

```r
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
```

```r
tidymodels_prefer()
```

```r
# 1. Install and load the `janitor` package.
# install.packages('janitor')
library(janitor)
set.seed(1234) # can be any number
```

```r
pokemon <- read.csv(file = "data/Pokemon.csv")
head(pokemon)
```

```
##   X.                  Name Type.1 Type.2 Total HP Attack Defense Sp..Atk
## 1  1             Bulbasaur  Grass Poison   318 45     49      49      65
## 2  2               Ivysaur  Grass Poison   405 60     62      63      80
## 3  3              Venusaur  Grass Poison   525 80     82      83     100
## 4  3 VenusaurMega Venusaur  Grass Poison   625 80    100     123     122
## 5  4            Charmander   Fire          309 39     52      43      60
## 6  5            Charmeleon   Fire          405 58     64      58      80
##   Sp..Def Speed Generation Legendary
## 1      65    45          1     False
## 2      80    60          1     False
## 3     100    80          1     False
## 4     120    80          1     False
## 5      50    65          1     False
## 6      65    80          1     False
```

```r
#  Use its `clean_names()` function on the Pokémon data
pokemon<-clean_names(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
```

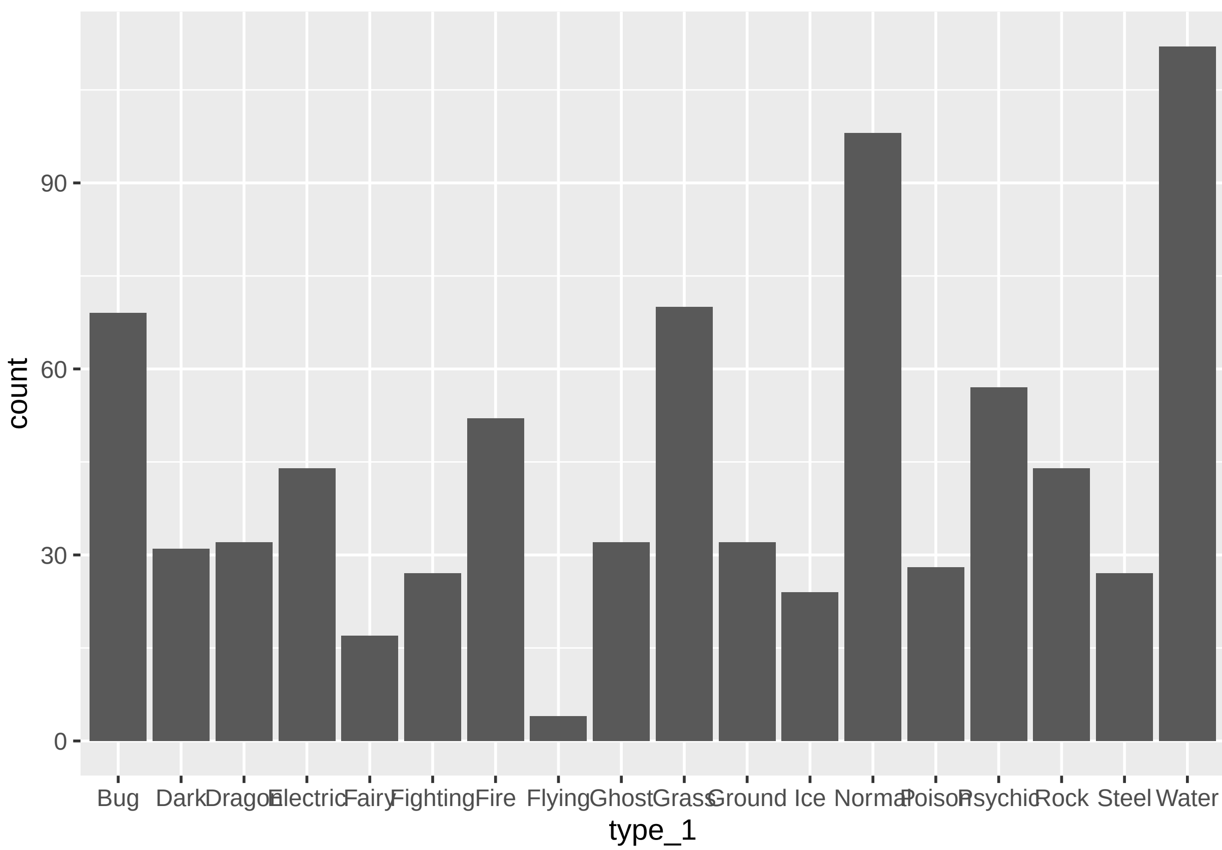
```
## 6    80          1      False
```

- What happened to the data?
- Answer:
- By comparing the output with `clean_names()` and the output without `clean_names()`, we can see that the column names have changed by `clean_names()`function.
- To be more specific, without using `clean_names()`, the output of the column names are mixed uppercase, lowercase and special characters. This is very complicated. For example, we can see some names like `Sp..Def` and `Type.2`. However, by using `clean_names()`, all the columns become more formatted (consist of lowercase and underscore). For example, `Sp..Def` changed to `sp_def` by using `clean_names()`.
- Why do you think `clean_names()` is useful?
- Answer:
- The `clean_names()` function makes the column names more formatted, which can help us make it easier to write code. We don't have to type both upper and lower case at same time, and we don't have to type too many special characters. It can improve our work efficiency.

Exercise 2

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

```
# create a bar chart of the outcome variable, `type_1`.
pokemon %>%
  ggplot(aes(x=type_1))+
  geom_bar()
```



- How many classes of the outcome are there? Are there any Pokémon types with very few Pokémon? If so, which ones?
- Answer:
- According to the graph, there are 18 classes of the outcome here.
- Flying Pokémon types with very few Pokémon.

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

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

```
# convert `type_1` and `legendary` to factors
pokemon$type_1 <- factor(pokemon$type_1)
pokemon$legendary <- factor(pokemon$legendary)

# double check whether `type_1` and `legendary` are factors
is.factor(pokemon$type_1)
```

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

```
is.factor(pokemon$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.

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?

```
# Perform an initial split of the data.
pokemon_split <- initial_split(pokemon,prop = 0.80, strata = type_1)
pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)

# Verify that your training and test sets have the desired number of observations.
dim(pokemon)
```

```
## [1] 458  13
```

```
dim(pokemon_train)
```

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

```
dim(pokemon_test)
```

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

- Verify that your training and test sets have the desired number of observations.
- Each dataset has approximately the right number of observations;
- For the training dataset, 364 is almost exactly 80% of the full data set, which contains 458 observations.
- For the testing dataset, 94 is almost exactly 20% of the full data set, which contains 458 observations.

```r
# Use *v*-fold cross-validation on the training set. Use 5 folds.
# Stratify the folds by `type_1` as well.
pokemon_folds <- vfold_cv(pokemon_train, v = 5, strata = type_1)
```

- Why might stratifying the folds be useful?
- In previous question, we have already noticed that our responses classes are imbalanced. At this time, it will be useful for us to stratify the folds.
- Here are the reasons:
- According to the lecture and online resource, we know that "in stratified k-fold cross-validation, the partitions are selected so that the mean response value is approximately equal in all the partitions." (https://en.wikipedia.org/wiki/Cross-validation_(statistics)) It means that stratifying the folds can "ensure that each fold is an appropriate representative of the original data. (class distribution, mean, variance, etc)" (https://stats.stackexchange.com/questions/49540/understanding-stratified-cross-validation)

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_def, pokemon_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).

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
# fitting and tuning an elastic net, tuning `penalty` and `mixture`
elastic_net_spec <- multinom_reg(penalty = tune(), mixture = tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

# Set up this model and workflow.
elastic_net_workflow <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(elastic_net_spec)

# Create a regular grid for `penalty` and `mixture` with 10 levels each; `mixture`
elastic_net_grid <- grid_regular(penalty(range = c(-5, 5)), mixture(range = c(0,1)), levels = 10)
```

- How many total models will you be fitting when you fit these models to your folded data?
- Answer: 500 models will be fit when we fit these models to your folded data.

Exercise 6

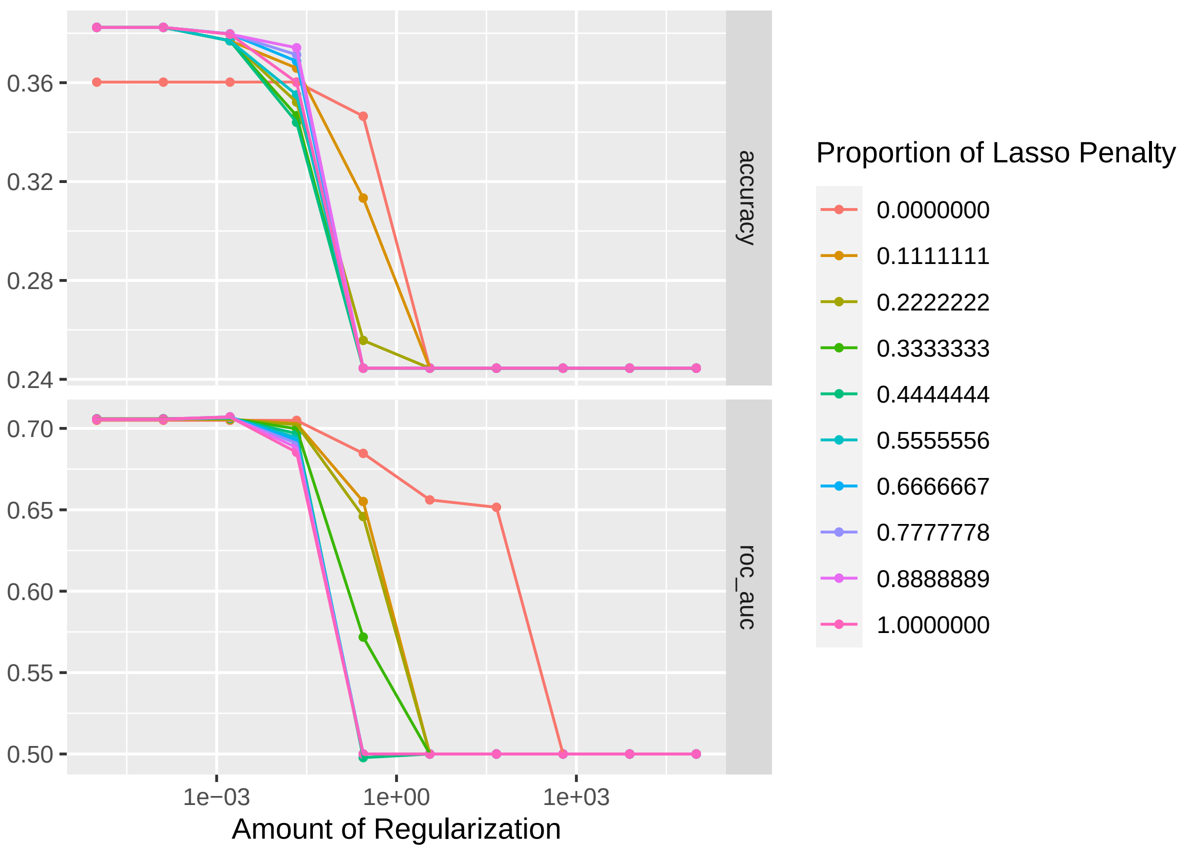
Fit the models to your folded data using `tune_grid()`.

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

```
# Fit the models to your folded data using `tune_grid()`.
tune_res <- tune_grid(
  elastic_net_workflow,
  resamples = pokemon_folds,
  grid = elastic_net_grid
)
```

```
## ! Fold1: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold2: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold3: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold4: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold5: preprocessor 1/1: The following variables are not factor vectors and wil...
```

```
# Use `autoplot()` on the results.
autoplot(tune_res)
```



- What do you notice? Do larger or smaller values of `penalty` and `mixture` produce better accuracy and ROC AUC?
- 1. We notice that as the amount of regularization increases, both accuracy and ROC_AUC will decrease.

- 2. According to this graph, for ROC_AUC, we can conclude that smaller values of `penalty` and `mixture` produce better ROC_AUC. However, for accuracy, we cannot make a conclusion just based on this graph. From the top of the graph, we can see that before the midpoint of 1*e-03 and 1e+00, larger values of `penalty` and `mixture` produce better accuracy, but after that point, smaller values of `penalty` and `mixture` produce better accuracy.

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.

```
# Use `select_best()` to choose the model that has the optimal `roc_auc`
best_model <- select_best(tune_res, metric = "roc_auc" )
best_model
```

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

```
# Then use `finalize_workflow()`, `fit()`, and `augment()` to fit the model
# to the training set and evaluate its performance on the testing set.
elastic_net_final <- finalize_workflow(elastic_net_workflow, best_model)
elastic_net_final_fit <- fit(elastic_net_final, data = pokemon_train)
```

```
## Warning: The following variables are not factor vectors and will be ignored:
## `generation`
```

```
# first method from the lab
multi_metric <- metric_set(accuracy, sensitivity, specificity)
augment(elastic_net_final_fit, new_data = pokemon_test) %>% multi_metric(truth = type_1, estimate = .pr
```

```
## # A tibble: 3 x 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 accuracy    multiclass     0.383
## 2 sensitivity macro          0.344
## 3 specificity macro          0.873
```

```
# the second method from office hours
augment(elastic_net_final_fit, new_data = pokemon_test) %>%
                  select(type_1,starts_with(".pred"))
```

```
## # A tibble: 94 x 8
##    type_1 .pred_class .pred_Bug .pred_Fire .pred_Grass .pred_Normal
##    <fct>  <fct>           <dbl>      <dbl>       <dbl>        <dbl>
##  1 Grass  Water         0.176       0.129       0.166        0.0674
##  2 Fire   Psychic       0.0124      0.259       0.137        0.00412
##  3 Water  Water         0.278       0.0843      0.153        0.0944
##  4 Bug    Normal        0.163       0.0399      0.0548       0.515
##  5 Normal Normal        0.290       0.0621      0.0552       0.434
##  6 Normal Normal        0.00859     0.0254      0.0244       0.741
##  7 Bug    Water         0.0638      0.148       0.128        0.144
##  8 Water  Water         0.141       0.135       0.143        0.128
##  9 Water  Water         0.0643      0.156       0.146        0.123
## 10 Water  Normal        0.170       0.0657      0.0570       0.491
## # ... with 84 more rows, and 2 more variables: .pred_Psychic <dbl>,
```

```
## #   .pred_Water <dbl>
```

Exercise 8

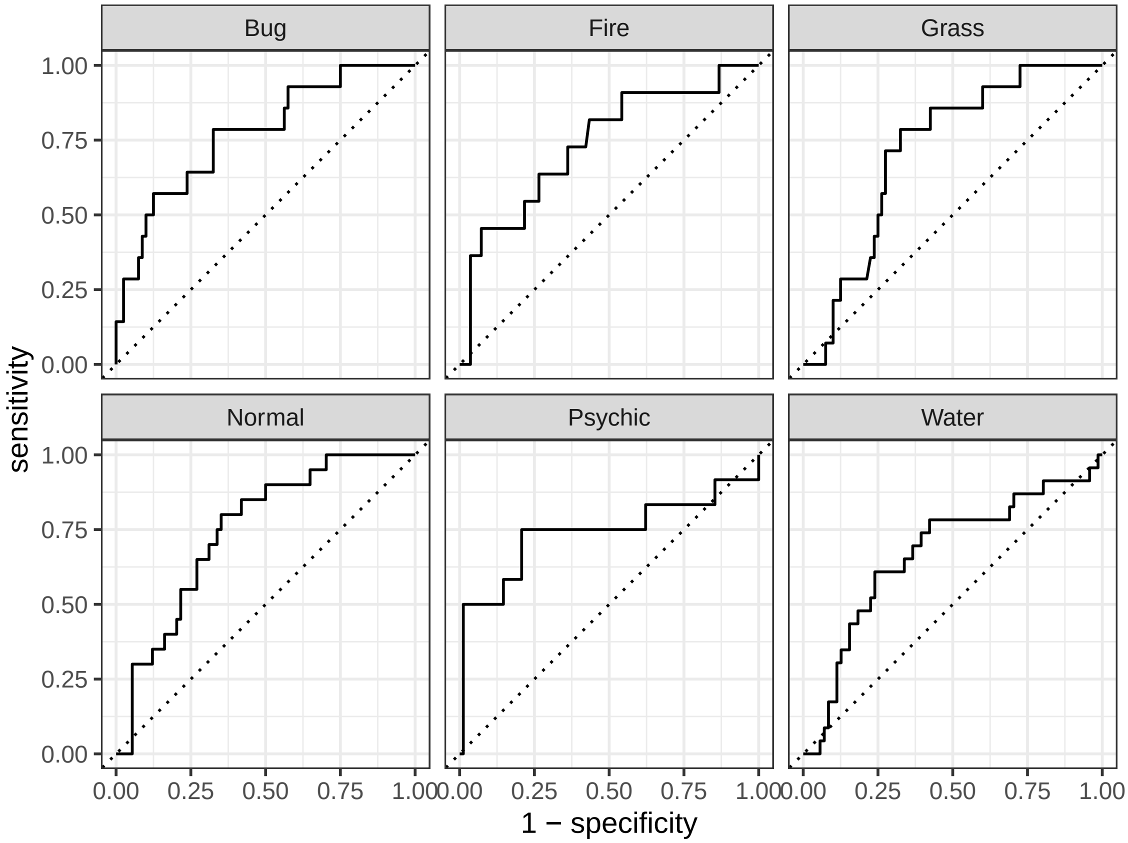
Calculate the overall ROC AUC on the testing set.

```
augment(elastic_net_final_fit, new_data = pokemon_test) %>% roc_auc(type_1, .pred_Bug:.pred_Water)
```

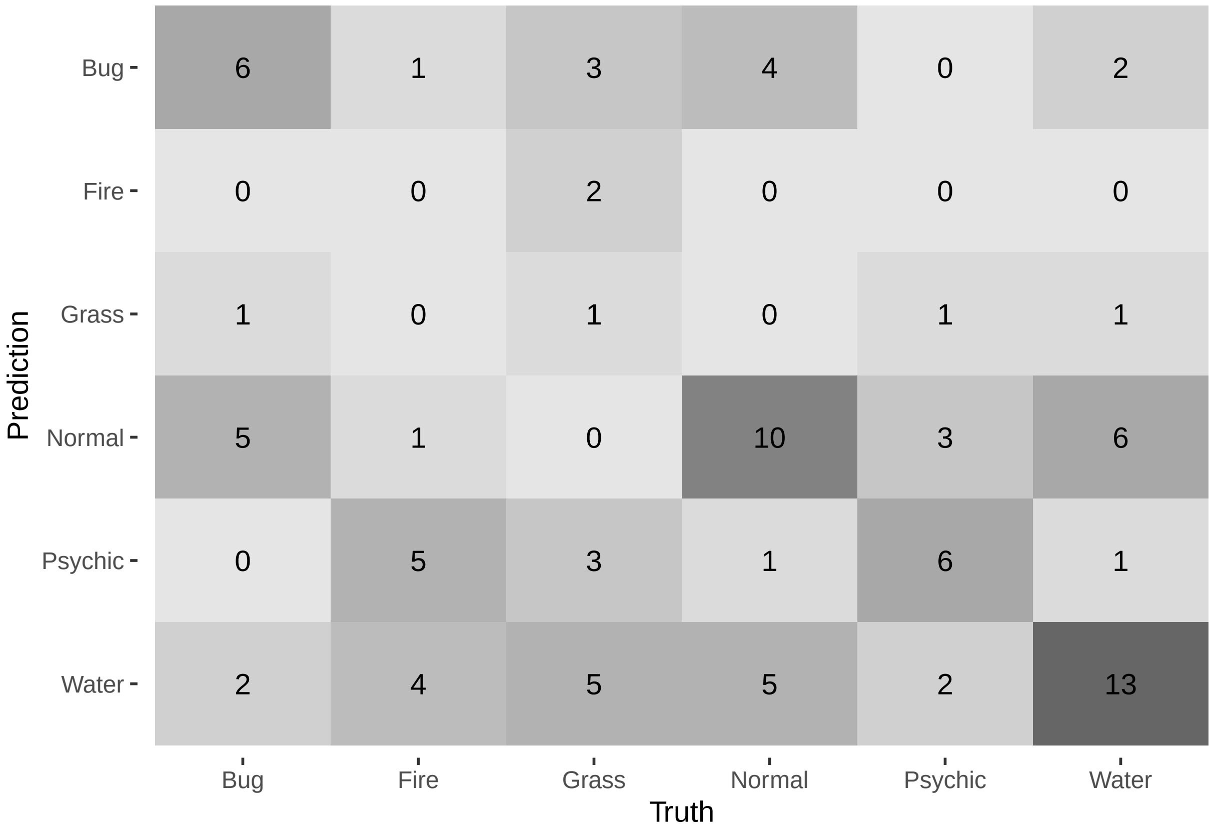
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.730
```

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

```
# create plots of the different ROC curves, one per level of the outcome.
augment(elastic_net_final_fit, new_data = pokemon_test) %>% roc_curve(type_1, .pred_Bug:.pred_Water)%>%a
```



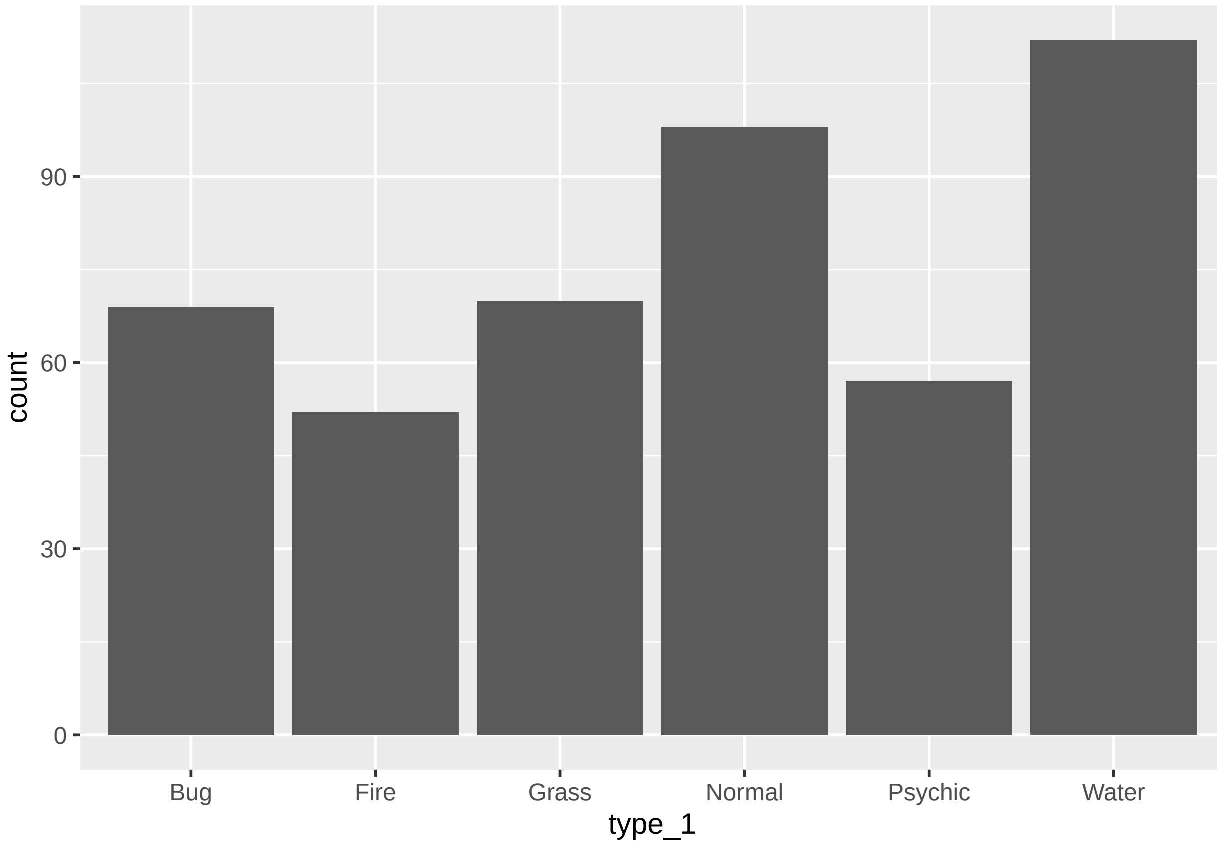

```
# make a heat map of the confusion matrix
augment(elastic_net_final_fit, new_data = pokemon_test) %>%
  conf_mat(truth = type_1, estimate =.pred_class)%>%
  autoplot("heatmap")
```



- 1. What do you notice?

- We notice that the overall ROC AUC on the testing set is 0.73, it is not good enough. Also, from the plots of the different ROC curves, we can see that normal type of Pokemon may have best ROC curves.
- 2. How did your model do?
- Since the overall ROC AUC on the testing set is 0.73 is less than 0.8, we can conclude that our model didn't do pretty well. This model is not good enough.
- 3. Which Pokemon types is the model best at predicting, and which is it worst at?
- In order to answer this question, let calculate the number of truth / the number of total
- for bug: $6/(6+1+3+4+0+2) = 0.375$
- for fire: $0/(2+0+0+0+0+0) = 0$
- for grass: $1/(1+1+1+1+0+0) = 0.25$
- for normal: $10/(5+1+0+10+3+6) = 0.4$
- for psychic: $6/(0+5+3+1+6+1) = 0.375$
- for water: $13/(2+4+5+5+2+13) = 0.41935$
- From our calculations, we can see that the water type of Pokemon is the model best at predicting, and the fire type of Pokemon is the model worst at predicting.
- 3. Do you have any ideas why this might be?

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



- From this graph, we can know that the fire type of Pokemon has the fewest observations, and the water type of Pokemon has the most observations. It may be the reason why the water type of Pokemon is the model best at predicting, and the fire type of Pokemon is the model worst at predicting.