Homework 5

PSTAT 131/231

Contents

Elastic Net Tuning

For this assignment, we will be working with the file "pokemon.csv",

The Pokémonfranchise 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.

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.

```
Pokemon <- read.csv("Pokemon.csv", header=TRUE)
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
```

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? ### Answer1 The tidyverse style guide recommends snake case (words separated by underscores like_this) for object and column names. Let's look back at our column names for a minute. There are all sorts of capital letters and dots (e.g. "Sp.." "Type.1"). The clean_names() function will convert all of these to snake case for us.

```
library(janitor)
Pokemon <-
  Pokemon %>%
  clean_names()

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
```

Exercise 2

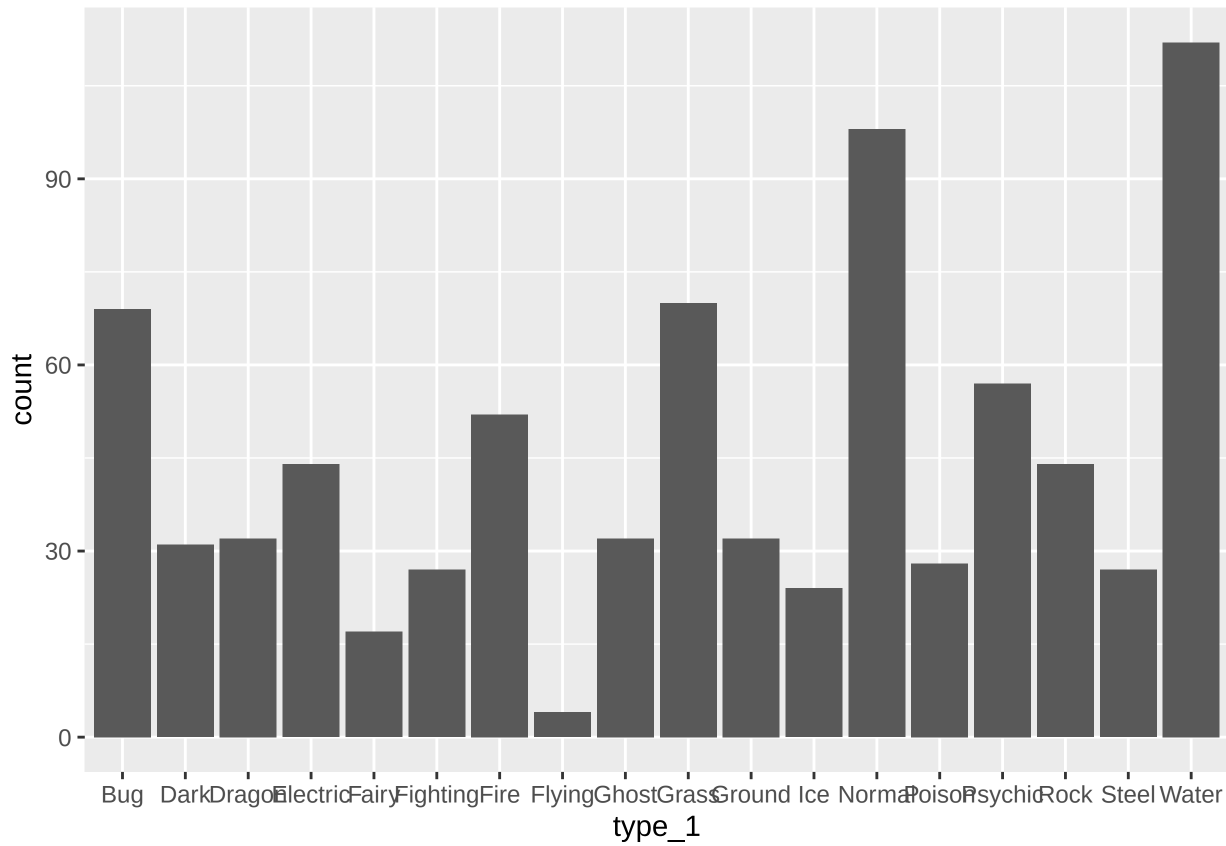
Using the entire data set, create a bar chart of the outcome variable, `type_1`.

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.

```
library(ggplot2)
library(tidymodels)
library(ROCR)
Pokemon %>%
  ggplot(aes(x =type_1)) +
  geom_bar()
```



Total number of outcome = 18, Pokemon with least number are Flying Pokemon.

```
#str(Pokemon$type_1)
length(unique(Pokemon$type_1))
```

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

```
filterdf <- Pokemon %>% filter(type_1%in% c("Bug", "Fire", "Grass", "Normal","Water","Psychic" )) %>%
            mutate(type_1 = factor(type_1),
                  legendary = factor(legendary))

head(filterdf)
```

```
##   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
```

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?

Answer 3:

```
filterdf_split <- filterdf %>%
  initial_split(strata = type_1, prop = 0.7)
filterdf_train <- training(filterdf_split)
filterdf_test <- testing(filterdf_split)
dim(filterdf_train)
```

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

```
dim(filterdf_test)
```

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

```
Auto_folds <- vfold_cv(filterdf_train, strata = type_1, v = 5)
Auto_folds
```

```
## #  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
```

Each resample is created within the stratification variable ### 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.

```
filterdf_recipe <- recipe(type_1 ~ legendary+ generation + sp_atk +
                           attack + speed+ hp+sp_def, filterdf_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_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?

```
filterdf_spec <-
  multinom_reg(penalty = tune(), mixture = 0) %>%
  set_mode("classification") %>%
  set_engine("glmnet")
```

```
filterdf_workflow <- workflow() %>%
  add_recipe(filterdf_recipe) %>%
  add_model(filterdf_spec)
```

```
penalty_grid <- grid_regular(penalty(range = c(-5, 5)), levels = 10)
penalty_grid
```

```
## # A tibble: 10 x 1
##          penalty
##            <dbl>
##  1      0.00001
##  2      0.000129
##  3      0.00167
##  4      0.0215
##  5      0.278
##  6      3.59
##  7     46.4
##  8    599.
##  9   7743.
## 10 100000
```

Exercise 6

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

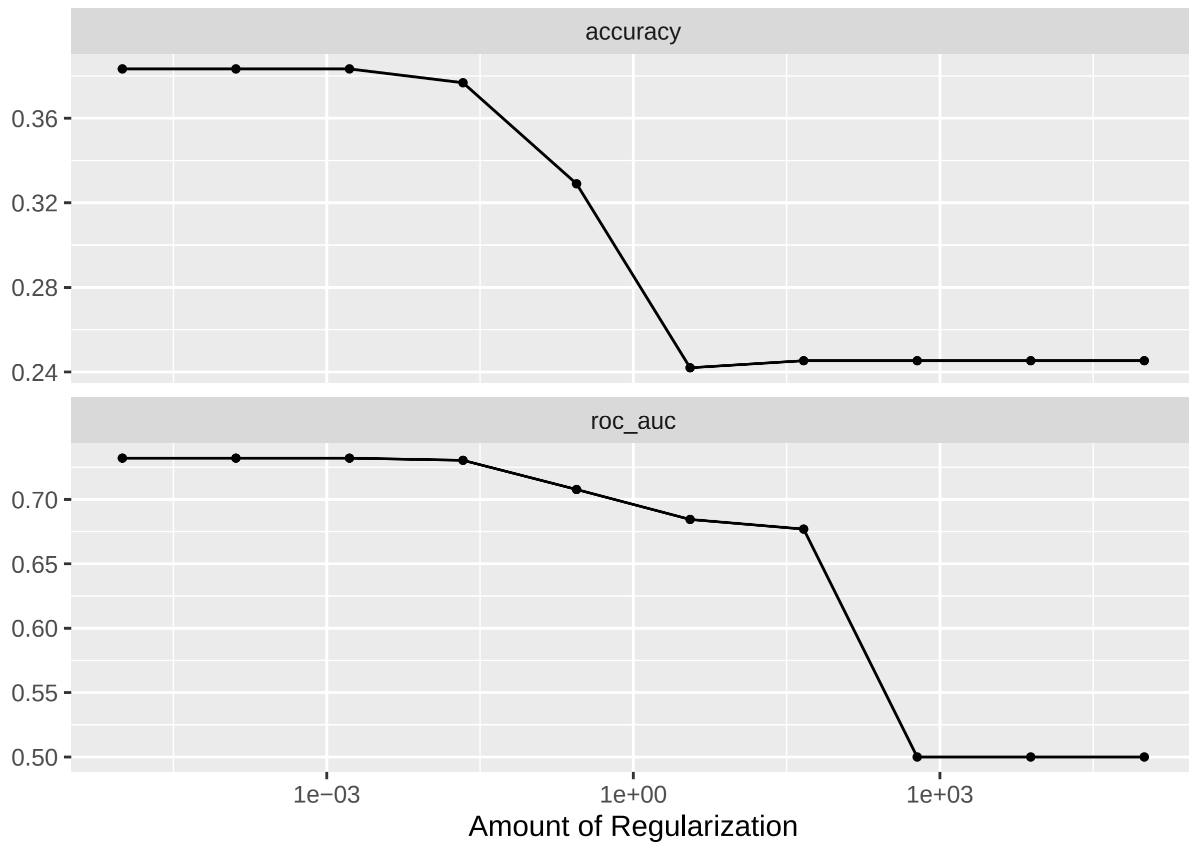
Use `autoplot()` on the results. What do you notice? Do larger or smaller values of `penalty` and `mixture` produce better accuracy and ROC AUC?

Answer 6: Smaller values of Penalty produces higher Accuracy and ROC AUC.

```
library(glmnet)
tune_res <- tune_grid(
  filterdf_workflow,
  resamples = Auto_folds,
  grid = penalty_grid
)

tune_res
```

```
## # Tuning results
## # 5-fold cross-validation using stratification
## # A tibble: 5 x 4
##   splits           id    .metrics          .notes
##   <list>           <chr> <list>            <list>
## 1 <split [252/66]> Fold1 <tibble [20 x 5]> <tibble [0 x 3]>
## 2 <split [253/65]> Fold2 <tibble [20 x 5]> <tibble [0 x 3]>
## 3 <split [253/65]> Fold3 <tibble [20 x 5]> <tibble [0 x 3]>
## 4 <split [256/62]> Fold4 <tibble [20 x 5]> <tibble [0 x 3]>
## 5 <split [258/60]> Fold5 <tibble [20 x 5]> <tibble [0 x 3]>
```

```r
autoplot(tune_res)
```



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.

```r
collect_metrics(tune_res)
```

```
## # A tibble: 20 x 7
##          penalty .metric  .estimator  mean     n std_err .config
##            <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1      0.00001  accuracy multiclass 0.383     5 0.00951 Preprocessor1_Model01
```

```
## 2      0.00001  roc_auc  hand_till  0.732     5 0.0143  Preprocessor1_Model01
## 3      0.000129 accuracy multiclass 0.383     5 0.00951 Preprocessor1_Model02
## 4      0.000129 roc_auc  hand_till  0.732     5 0.0143  Preprocessor1_Model02
## 5      0.00167  accuracy multiclass 0.383     5 0.00951 Preprocessor1_Model03
## 6      0.00167  roc_auc  hand_till  0.732     5 0.0143  Preprocessor1_Model03
## 7      0.0215   accuracy multiclass 0.377     5 0.00978 Preprocessor1_Model04
## 8      0.0215   roc_auc  hand_till  0.730     5 0.0149  Preprocessor1_Model04
## 9      0.278    accuracy multiclass 0.329     5 0.0206  Preprocessor1_Model05
## 10     0.278    roc_auc  hand_till  0.708     5 0.0194  Preprocessor1_Model05
## 11     3.59     accuracy multiclass 0.242     5 0.00234 Preprocessor1_Model06
## 12     3.59     roc_auc  hand_till  0.684     5 0.0211  Preprocessor1_Model06
## 13    46.4      accuracy multiclass 0.245     5 0.00147 Preprocessor1_Model07
## 14    46.4      roc_auc  hand_till  0.677     5 0.0206  Preprocessor1_Model07
## 15   599.       accuracy multiclass 0.245     5 0.00147 Preprocessor1_Model08
## 16   599.       roc_auc  hand_till  0.5       5 0       Preprocessor1_Model08
## 17  7743.       accuracy multiclass 0.245     5 0.00147 Preprocessor1_Model09
## 18  7743.       roc_auc  hand_till  0.5       5 0       Preprocessor1_Model09
## 19 100000       accuracy multiclass 0.245     5 0.00147 Preprocessor1_Model10
## 20 100000       roc_auc  hand_till  0.5       5 0       Preprocessor1_Model10
```

```
best_penalty <- select_best(tune_res, metric = "roc_auc")
best_penalty
```

```
## # A tibble: 1 x 2
##   penalty .config
##     <dbl> <chr>
## 1 0.00001 Preprocessor1_Model01
```

```
filterdf_final <- finalize_workflow(filterdf_workflow, best_penalty)
filterdf_final_fit <- fit(filterdf_final, data = filterdf_train)
```

```
augment(filterdf_final_fit, new_data = filterdf_test) %>%
  accuracy(truth =  type_1, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.343
```

Exercise 8

Calculate the overall ROC AUC on the testing set.

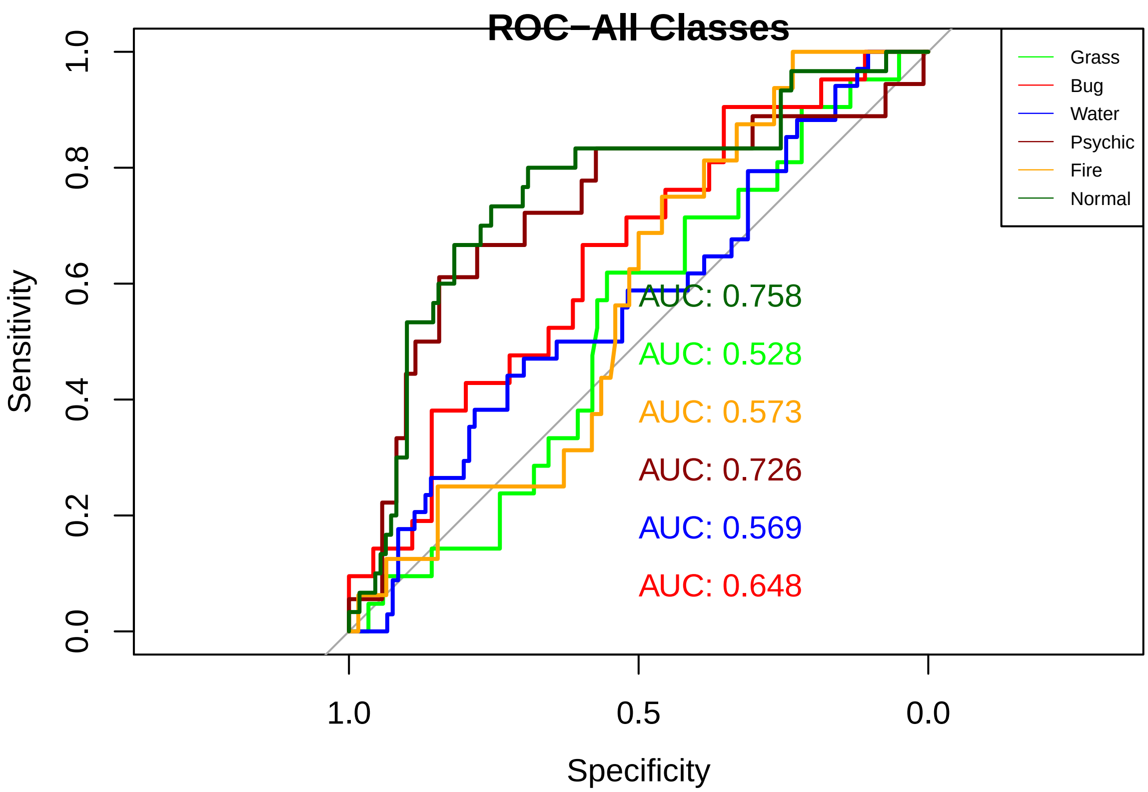
Then create plots of the different ROC curves, one per level of the outcome. Also make a heat map of the confusion matrix.

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?

```
augment(filterdf_final_fit, new_data = filterdf_test) %>%
  accuracy(truth =  type_1, estimate = .pred_class)
```

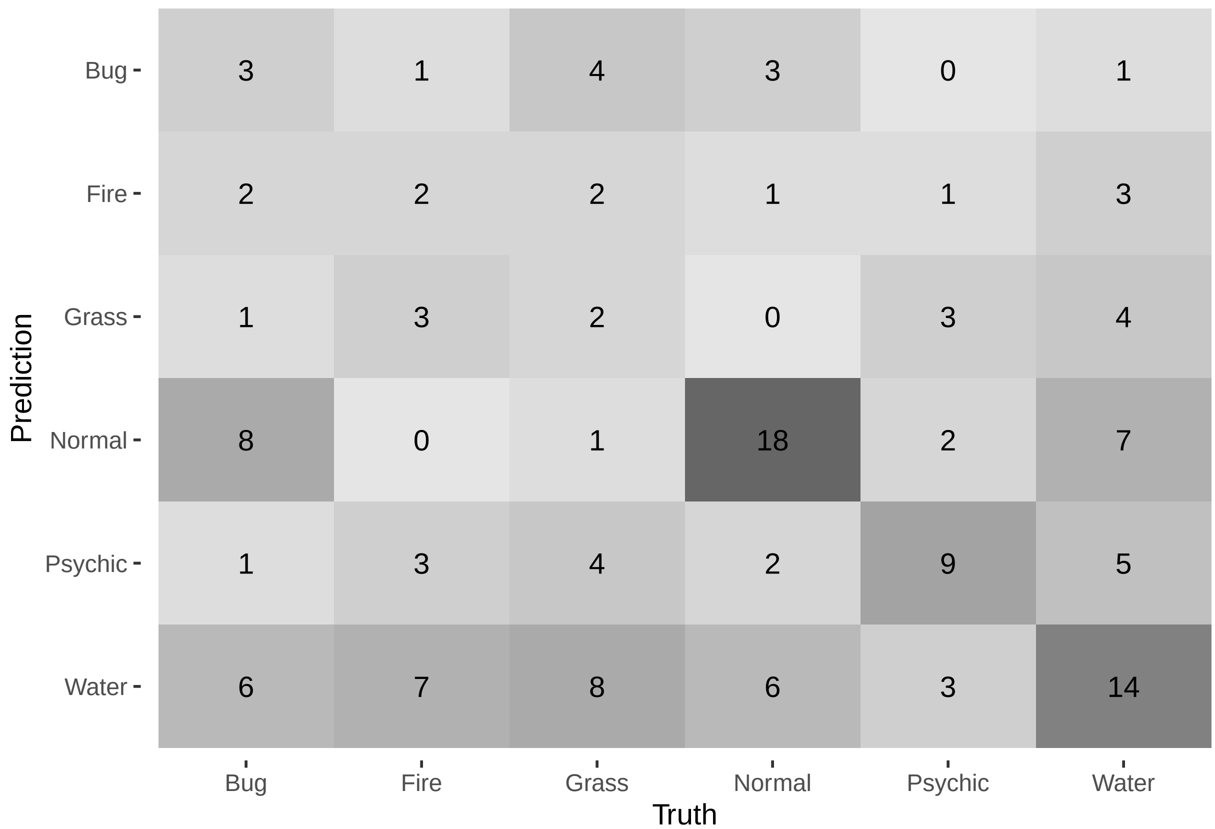
```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.343
```

```
library(pROC)
new<-augment(filterdf_final_fit, new_data = filterdf_test) %>%

    mutate(
        Grasss_true = ifelse(filterdf_test$type_1== "Grass", 1, 0),
        Fire_true = ifelse(filterdf_test$type_1== "Fire", 1, 0),
        Normal_true = ifelse(filterdf_test$type_1== "Normal", 1, 0),
        Psychic_true = ifelse(filterdf_test$type_1== "Psychic", 1, 0),
        Water_true = ifelse(filterdf_test$type_1== "Water", 1, 0),
        Bug_true = ifelse(filterdf_test$type_1== "Bug", 1, 0)

    )

roc_plot <- plot(roc(new$Grasss_true,new$.pred_Grass), print.auc=TRUE, col = "green")
roc_plot <- plot(roc(new$Bug_true,new$.pred_Bug), print.auc = TRUE,
                 col = "red", print.auc.y = .1, add = TRUE)
roc_plot <- plot(roc(new$Water_true,new$.pred_Water), print.auc = TRUE,
                 col = "blue", print.auc.y = .2, add = TRUE)
roc_plot <- plot(roc(new$Psychic_true,new$.pred_Psychic), print.auc = TRUE,
                 col = "darkred", print.auc.y = .3, add = TRUE)
roc_plot <- plot(roc(new$Fire_true,new$.pred_Fire), print.auc = TRUE,
                 col = "orange", print.auc.y = .4, add = TRUE)
roc_plot <- plot(roc(new$Normal_true,new$.pred_Normal), print.auc = TRUE,
                 col = "darkgreen", print.auc.y = .6, add = TRUE)

    plot_colors <- c("green","red","blue","darkred", "orange", "darkgreen" )
    legend(x = "topright",inset = 0,
       legend = c("Grass", "Bug", "Water", "Psychic", "Fire", "Normal"),
       col=plot_colors, lwd=.6, cex=.6, horiz = FALSE)
    title(main = "ROC-All Classes")
```



Noraml Pokemon performed best and worst is Water pokemon

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


Prediction
Bug
Fire
Grass
Normal
Psychic
Water
3
1
4
3
0
1
2
2
2
1
1
3
1
3
2
0
3
4
8
0
1
18
2
7
1
3
4
2
9
5
6
7
8
6
3
14
Bug
Fire
Grass
Normal
Psychic
Water
Truth