Elastic Net Tuning

For 231 Students

Homework 5

Code

PSTAT 131/231

Load Package

Hide

```
library(tinytex)
library(tidyverse)
library(tidymodels)
library(ISLR)
library(ggplot2)
library(corrplot)
library(ggthemes)
library(yardstick)
library(klaR)
library(glmnet)
library(dplyr)
library(magrittr)
library(corrr)
library(discrim)
library(poissonreg)
tidymodels_prefer()
set.seed(5)
```

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 (https://www.kaggle.com/abcsds/pokemon).

The Pokémon (https://www.pokemon.com/us/) 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 (https://bulbapedia.bulbagarden.net/wiki/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.

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?

Hide

```
library(janitor)
pokemon <- read.csv("/Users/Yuer_Hao/Desktop/Pstat231-HW5-main/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
```

Hide

```
#view(pokemon)
pkm <- pokemon %>% clean_names()
head(pkm)
```

```
##   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
```

Hide

```
#view(pkm)
```

The data's variable names shift to a clear format. The only characters in the names are the space character, letters, and digits. By default, all of the names are lowercase. It helps the user access and comprehend the variable names, which makes it useful.

Exercise 2

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

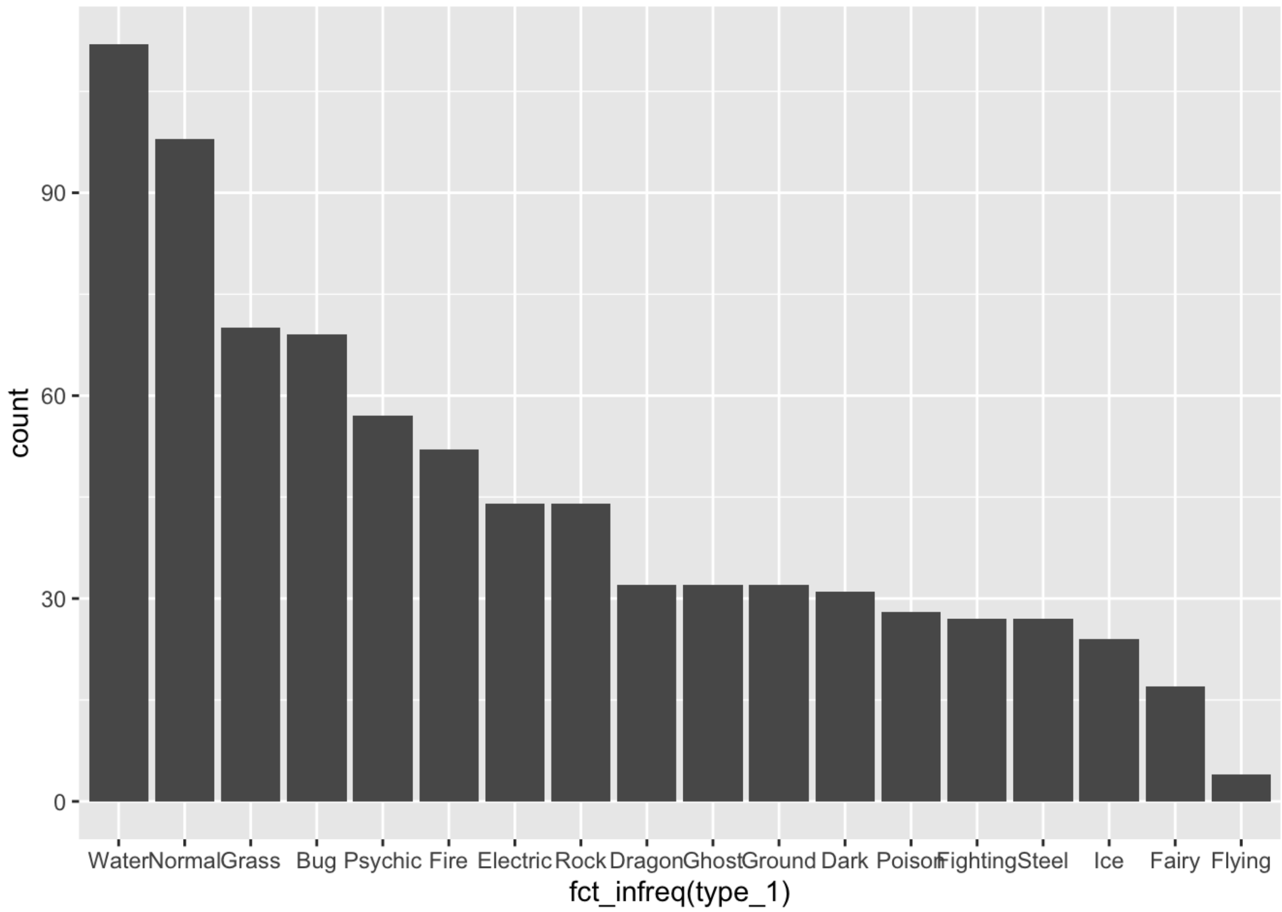
How many classes of the outcome are there? Are there any Pokémon types with very few Pokémon? If so, which ones?

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.

Hide

```
# Plot the bar chart in descending order
pkm %>% ggplot(aes(x=fct_infreq(type_1))) +
  geom_bar()
```



Hide

```
# Count the number of classes in the outcomes
nlevels(factor(pkm$type_1))
```

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

Hide

```
# Count observations in each level (descending order)
table(fct_infreq(pkm$type_1))
```

```
## 
##    Water   Normal    Grass      Bug  Psychic     Fire Electric     Rock 
##      112       98       70       69       57       52       44       44 
##   Dragon    Ghost   Ground     Dark   Poison Fighting    Steel      Ice 
##       32       32       32       31       28       27       27       24 
##    Fairy   Flying 
##       17        4
```

Based on the data set, we can see there are 18 classes of pokemons, and the “flying” type has only 4 pokemons, which is the one type that with very few pokemon. Besides, the pokemons that in type Poison, Fighting, Steel, Ice, and Fairy are less than 30.

Hide

```
pkm2 <- pkm %>%
  filter(type_1 %in%
           c("Bug","Fire","Grass","Normal","Water","Psychic"))
pkm2$type_1 <- factor(pkm2$type_1)
pkm2$legendary <- factor(pkm2$legendary)
pkm2$generation <- factor(pkm2$generation)
```

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?

Hide

```
#initial split
pkm_split <- initial_split(pkm2, prop = 0.80,  strata = "type_1")
pkm_train <- training(pkm_split)
pkm_test <- testing(pkm_split)
#verify the number of observations
dim(pkm_train)
```

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

Hide

```
dim(pkm_test)
```

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

Hide

```
364/(94+364)
```

```
## [1] 0.7947598
```

The number of observations is correct.

Hide

```
pkm_folds <- vfold_cv(pkm_train,
                      v=5,
                      strata = "type_1")
```

Because stratification guarantees that each class contains a representative number, it is beneficial. In other words, it guarantees that the training set and the entire dataset contain the same percentage of each class. The model will then perform equally well during training and testing.

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.

Hide

```
pkm_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed +
defense + hp + sp_def,
                    data = pkm_train) %>%
  step_dummy(c("legendary","generation")) %>%  # Dummy-code `legendary` and `gen
eration`
  step_center(all_predictors()) %>%  # Center and scale 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?

Hide

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

pkm_workflow <- workflow() %>%
  add_recipe(pkm_recipe) %>%
  add_model(pkm_spec)

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

There will be 500 models in total.
(10 penalty levels x 10 mixture levels x 5 folds)

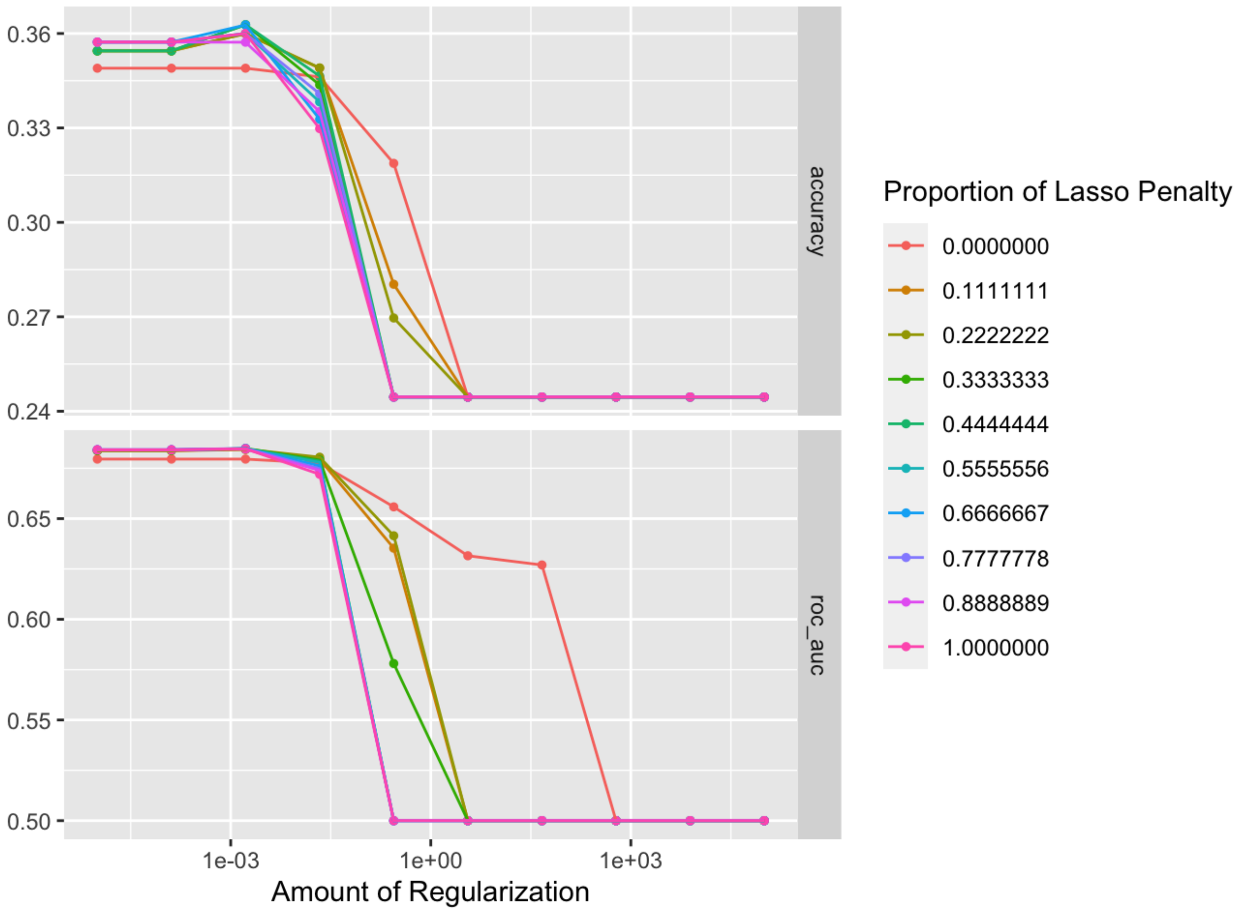
Exercise 6

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?

Hide

```
tune_res <- tune_grid(pkm_workflow,resamples = pkm_folds,
                      grid = penalty_grid)
autoplot(tune_res)
```



The graphs demonstrate that smaller values of the penalty and mixture result in higher 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.

Hide

```
pkm_best<-select_best(tune_res,metrix="roc_auc")
pkm_final<-finalize_workflow(pkm_workflow,pkm_best)
pkm_fit <- fit(pkm_final, data = pkm_train)
predict(pkm_fit,new_data=pkm_test,type="class")
```

```
## # A tibble: 94 × 1
##    .pred_class
##    <fct>
##  1 Water
##  2 Fire
##  3 Water
##  4 Water
##  5 Normal
##  6 Normal
##  7 Water
##  8 Water
##  9 Normal
## 10 Normal
## # … with 84 more rows
```

Hide

```
test_acc<-augment(pkm_fit,new_data=pkm_test) %>%
  accuracy(truth=type_1,estimate=.pred_class)
test_acc
```

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.372
```

The accuracy of the model on the testing set is just 0.3723, hence the output indicates that it does not perform well.

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?

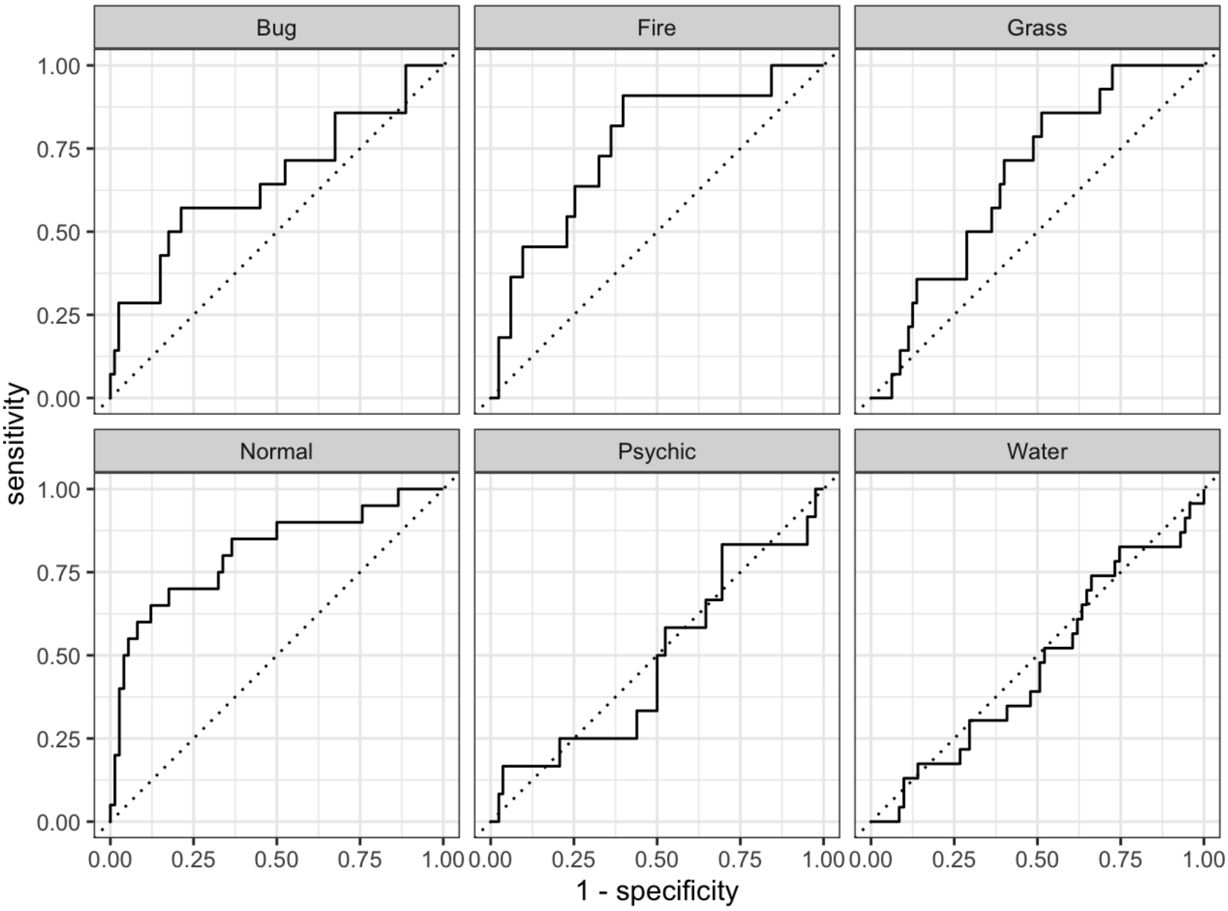
Hide

```
roc_auc(augment(pkm_fit,new_data=pkm_test),
        type_1,.pred_Bug,.pred_Fire,.pred_Grass,.pred_Normal,
        .pred_Water,.pred_Psychic)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.635
```

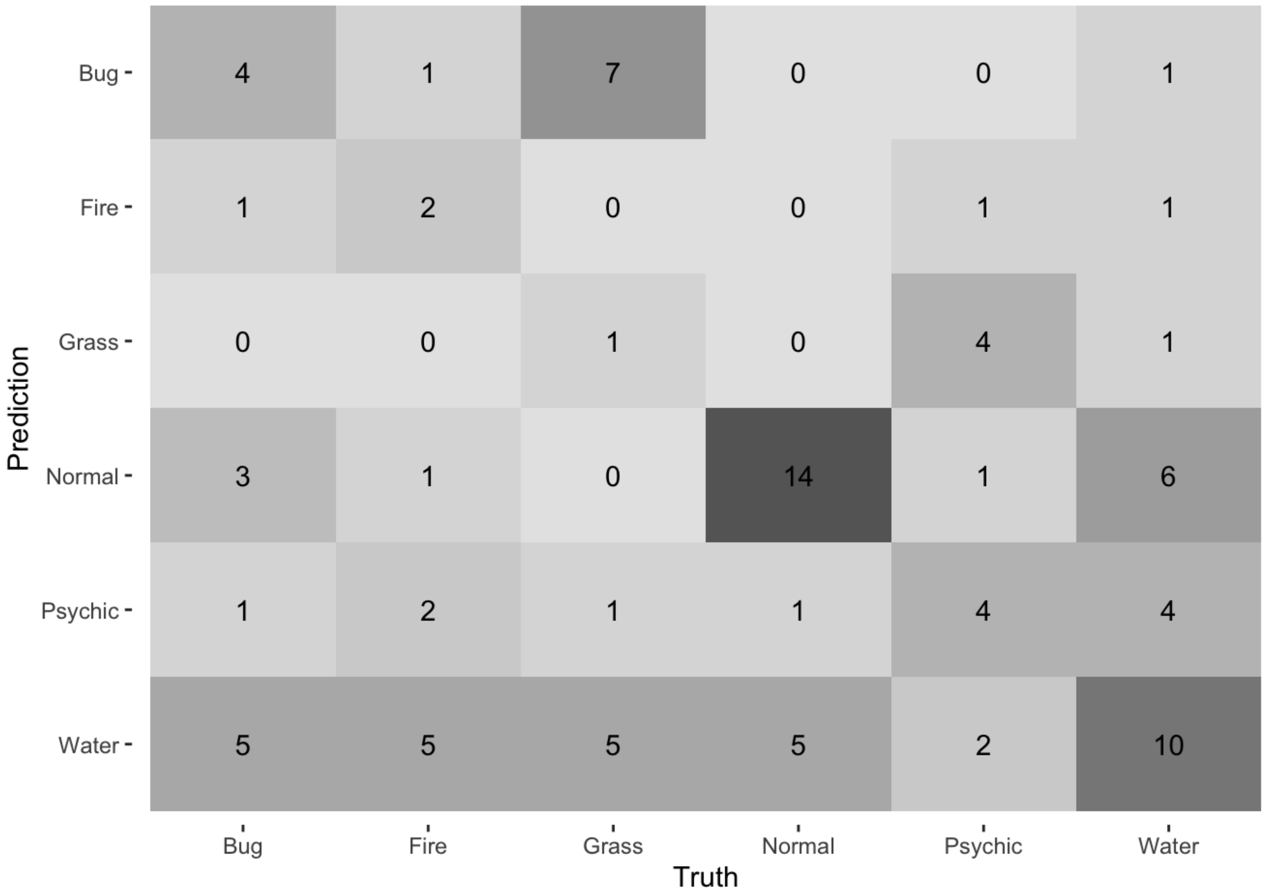
Hide

```
roc_curve(augment(pkm_fit,new_data=pkm_test),
        type_1,.pred_Bug,.pred_Fire,.pred_Grass,.pred_Normal,
        .pred_Water,.pred_Psychic) %>%
  autoplot()
```



Hide

```
conf_mat(augment(pkm_fit,new_data=pkm_test),
         truth=type_1,.pred_class) %>%
  autoplot(type="heatmap")
```



The accuracy of the model is 0.3723, and the total rocauc is only 0.6351, which is not very high. The model performs best on the Normal type with the largest area under the ROC curve, and worst on the Psychic type, which has the smallest area under the roc curve. The reason might be we don't have enough characteristics to employ for prediction because the sample size of psychics is too small.

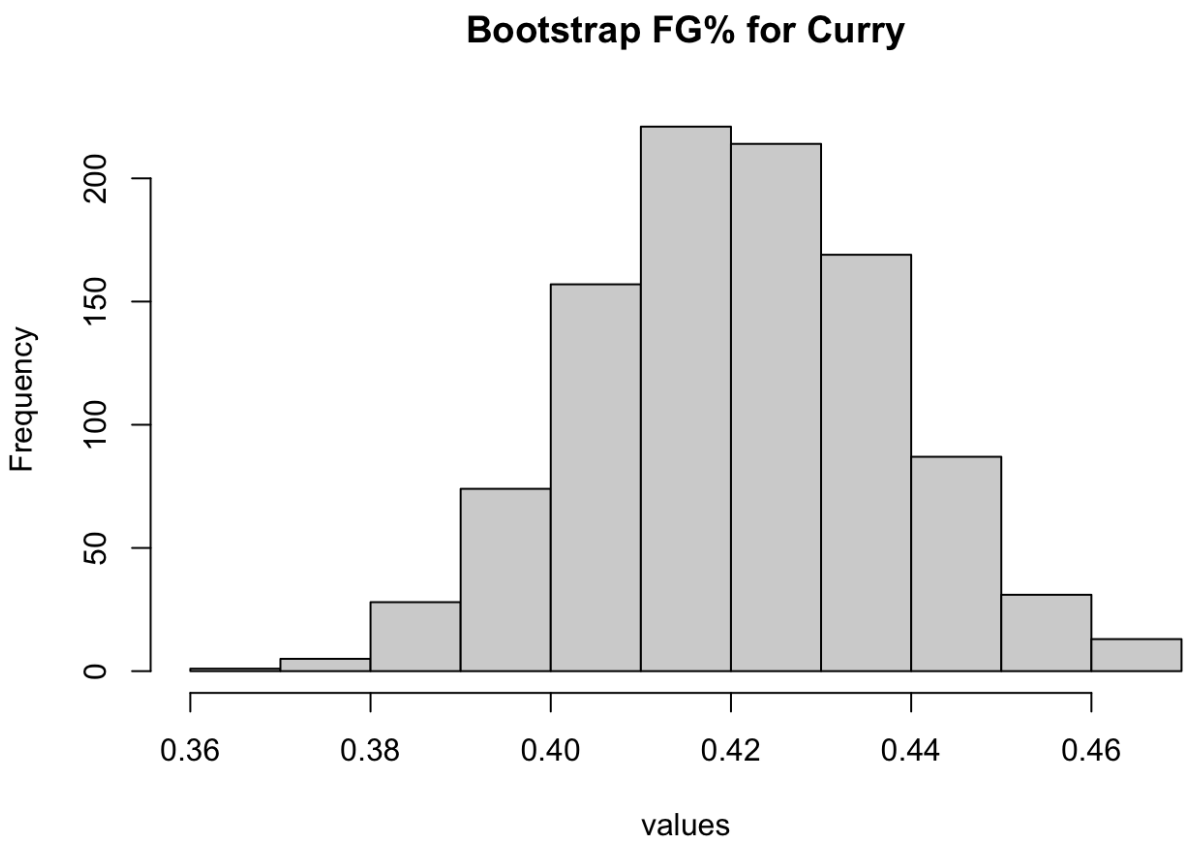
For 231 Students

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.

Hide

```r
StephenCurry_shots = rep(1:0, c(337, 464))
PNB <- function(n){
  x = list()
  for (i in 1:n){
    bootstrap = sample(StephenCurry_shots,
                       length(StephenCurry_shots),
                       replace = T)
    x = append(x,
               sum(bootstrap)/length(bootstrap))
  }
  return (unlist(x))
}
# Use 1000 bootstrap samples to plot a histogram of those values.
values = PNB(1000)
hist(values, main = "Bootstrap FG% for Curry")
```



Hide

```r
quantile(values, probs = seq(0.005, 0.995, 0.99))
```

```
##      0.5%     99.5%
## 0.3795256 0.4631835
```

The 99% bootstrap confidence interval is [0.3795, 0.4631] with the endpoints rounded from the result above.