Elastic Net Tuning

For 231 Students

Homework 5

Code

PSTAT 131/231

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)
library(ggplot2)
library(dbplyr)
library(tidyverse)
library(tidymodels)
library(corrr)

library(ISLR) # For the Smarket data set
library(ISLR2) # For the Bikeshare data set
library(discrim)
library(poissonreg)
library(glmnet)
library(klaR) # for naive bayes
tidymodels_prefer()
library(pROC)
library(boot)
library(rsample)
```

Hide

```
pokemon_data <- read.csv("pokemon.csv")
pokemon <- clean_names(pokemon_data)
```

Name of columns are changed, resulting in unique names consist only of underscore, numbers and lower-case letters.

Exercise 2

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

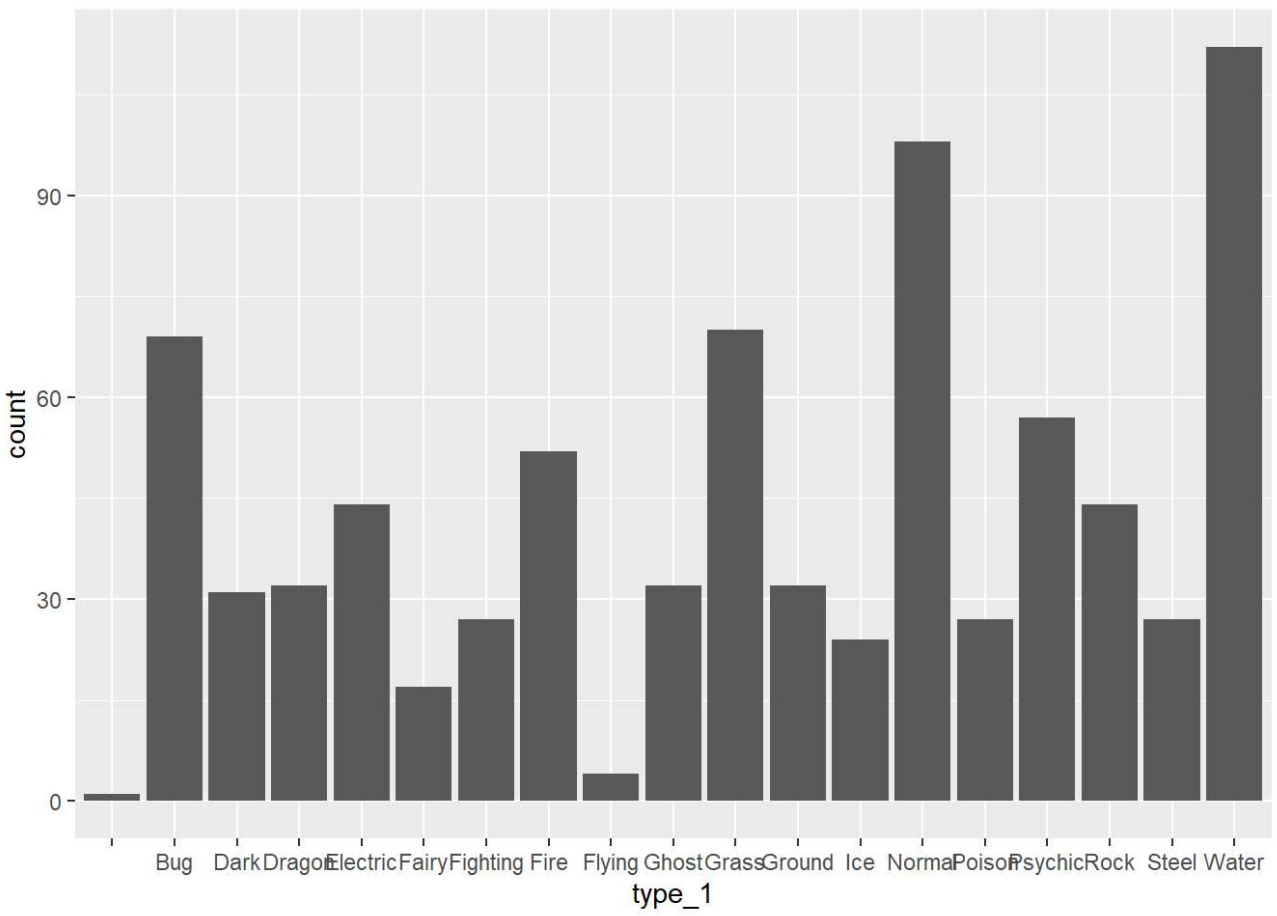
How many classes of the outcome are there? Are there any Pokémon types with very few Pokémon? If so, which ones?

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

```
ggplot(data=pokemon, aes(type_1)) +
  geom_bar()
```



There are 18 classes of the outcomes. Flying class has very few pokemon. Fairy, ice, poison and steel are also with few pokemon.

Hide

```
pokemon1 <- pokemon %>% filter(type_1 %in% c("Bug","Fire", "Grass", "Normal", "Water","Ps
ychic"))

pokemon1$type_1 <- factor(pokemon1$type_1)
pokemon1$legendary <- factor(pokemon1$legendary)
```

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

```
pokemon1_split <- initial_split(pokemon1,prop=0.80,strata=type_1)
pokemon1_training <- training(pokemon1_split)
pokemon1_testing <- testing(pokemon1_split)
dim(pokemon1_testing)
```

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

Hide

```
dim(pokemon1_training)
```

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

Hide

```
pokemon_folds <- vfold_cv(pokemon1_training,v=5, strata=type_1)
pokemon_folds
```

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 x 2
##   splits           id
##   <list>           <chr>
## 1 <split [289/75]> Fold1
## 2 <split [291/73]> Fold2
## 3 <split [291/73]> Fold3
## 4 <split [292/72]> Fold4
## 5 <split [293/71]> Fold5
```

Stratified folds helps balance the percentage of each class.

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

```
pokemon_recipe <- recipe(type_1~legendary+generation+sp_atk+attack+speed+defense+hp+sp_de
f,
                         data=pokemon1_training) %>% step_dummy(legendary,generation) %>%
step_normalize(all_predictors())

pokemon_recipe %>% prep() %>% bake(pokemon1_training)
```

```
## # A tibble: 364 x 13
##    sp_atk attack   speed defense      hp sp_def type_1 legendary_True
##     <dbl>  <dbl>   <dbl>   <dbl>   <dbl>  <dbl> <fct>           <dbl>
##  1 -1.60  -1.41  -0.777   -1.27  -0.955  -1.88  Bug            -0.259
##  2 -1.45  -1.72  -1.27    -0.510 -0.766  -1.69  Bug            -0.259
##  3  0.521 -0.938  0.0438  -0.701 -0.388   0.395 Bug            -0.259
##  4 -1.60  -1.25  -0.613   -1.47  -1.14   -1.88  Bug            -0.259
##  5 -0.841  0.470  0.208   -1.08  -0.200   0.395 Bug            -0.259
##  6 -1.75   2.35   2.51    -1.08  -0.200   0.395 Bug            -0.259
##  7 -0.841 -0.156 -1.43    -0.510 -1.33   -0.554 Bug            -0.259
##  8 -0.387  0.626 -1.27     0.445 -0.388   0.395 Bug            -0.259
##  9 -0.992 -0.625 -0.777   -0.701 -0.388  -0.554 Bug            -0.259
## 10  0.521 -0.312  0.701   -0.319 -0.0108  0.205 Bug            -0.259
## # ... with 354 more rows, and 5 more variables: generation_X2 <dbl>,
## #   generation_X3 <dbl>, generation_X4 <dbl>, generation_X5 <dbl>,
## #   generation_X6 <dbl>
```

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

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

pokemon_workflow <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(tune_engine)

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

10x10x5 = 500 models.

Exercise 6

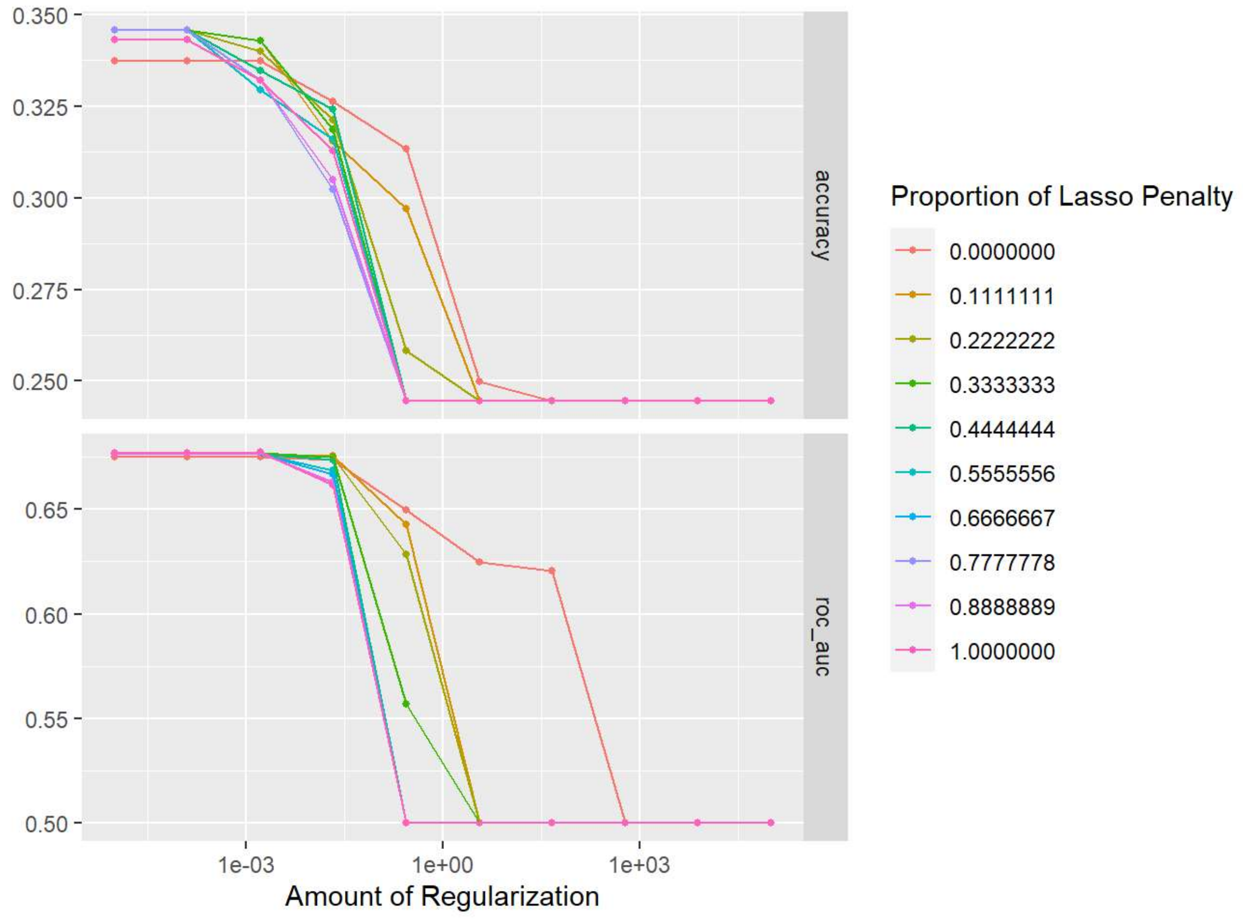
Fit the models to your folded data using `tune_grid()` .

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(
  pokemon_workflow,
  resamples = pokemon_folds,
  grid = penalty_grid
)

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.

Hide

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

final_fit <- fit(final, data = pokemon1_training)

predict(final_fit, new_data = pokemon1_testing, type = "class")
```

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

Hide

```
test_acc <- augment(final_fit, new_data = pokemon1_testing) %>%
  accuracy(truth = type_1, estimate = .pred_class)

test_acc
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.415
```

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?

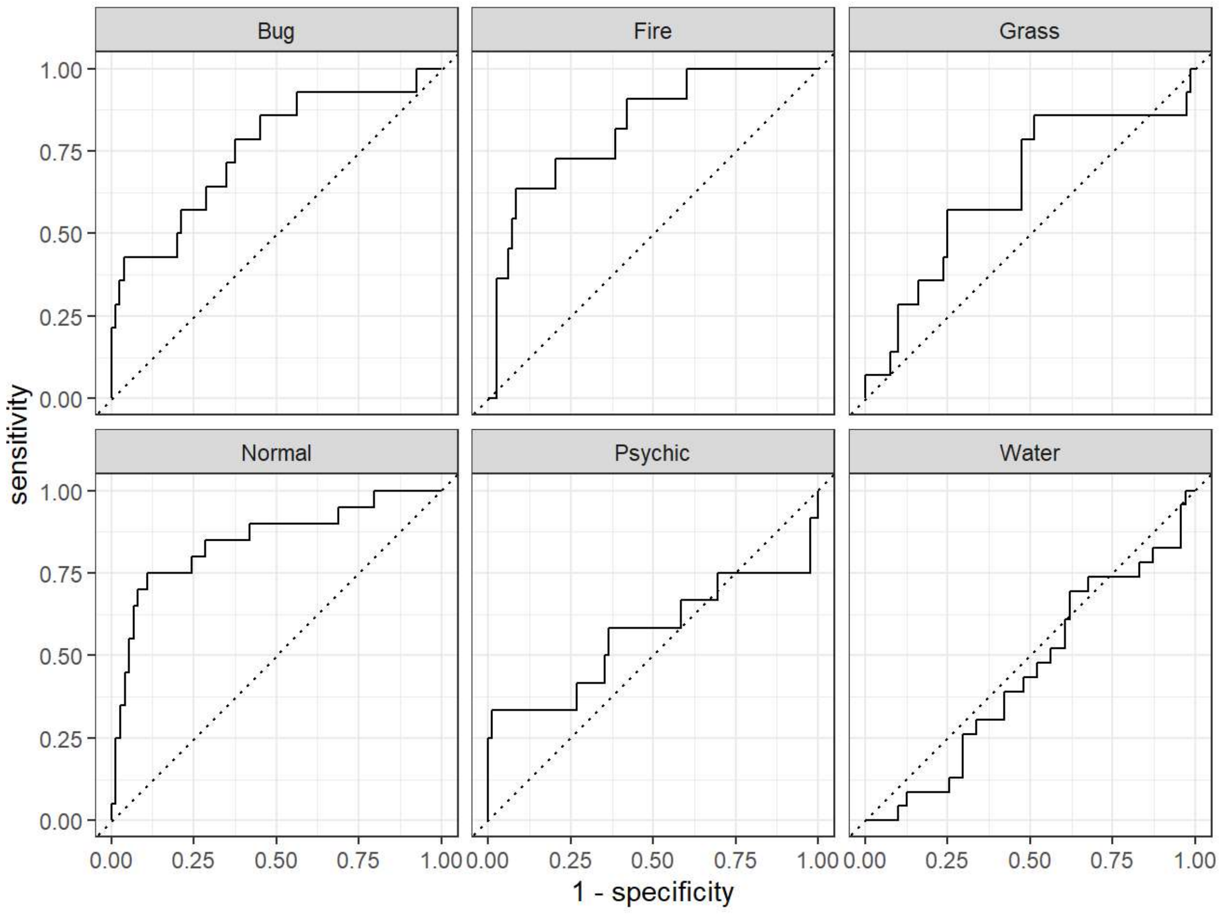
Hide

```
augment(final_fit, new_data = pokemon1_testing) %>%
   roc_auc(type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal,.pred_Water,.pred_Psy
chic)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.677
```

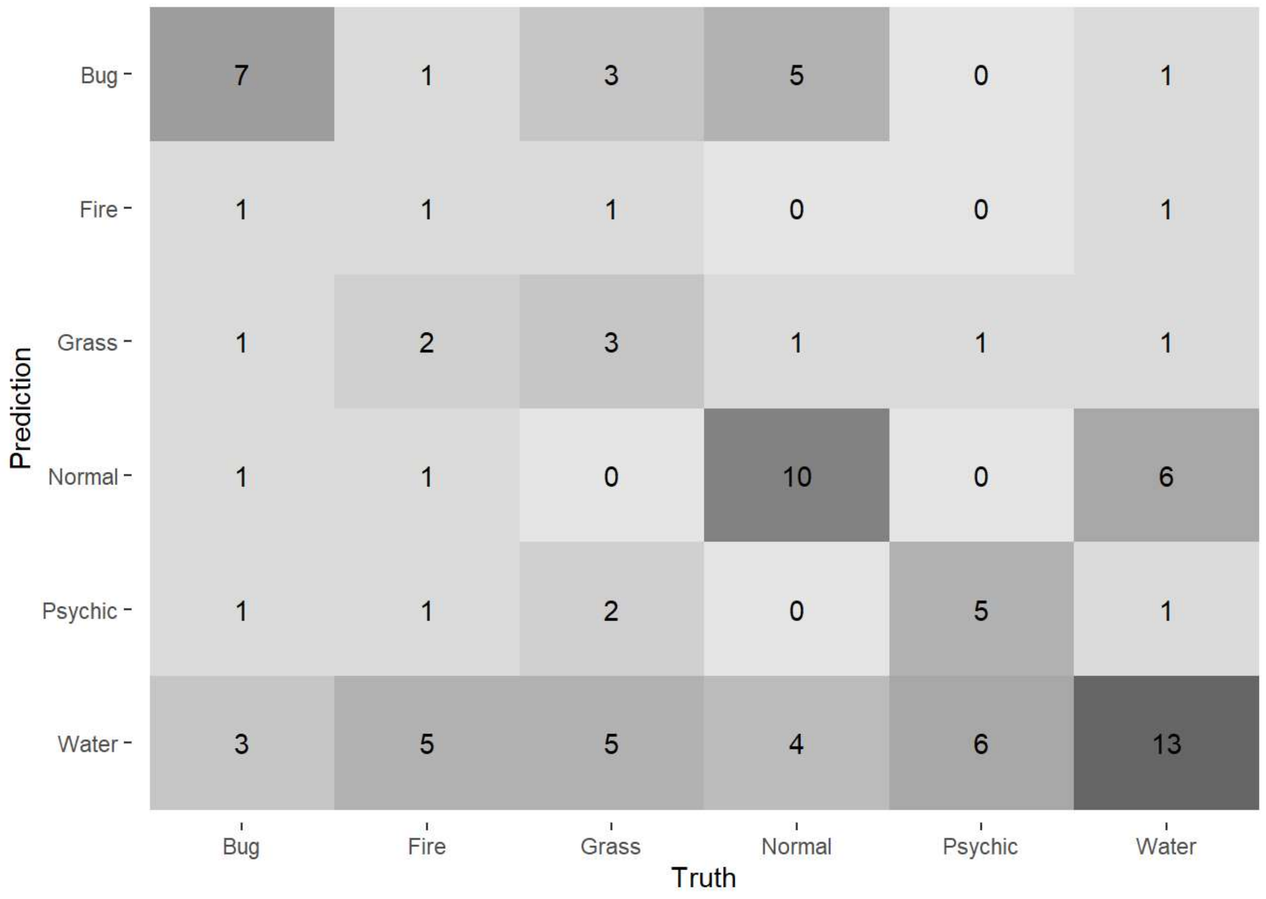
Hide

```
augment(final_fit, new_data = pokemon1_testing) %>% roc_curve(type_1, .pred_Bug, .pred_Fi
re, .pred_Grass, .pred_Normal,.pred_Water,.pred_Psychic) %>%
  autoplot()
```



Hide

```
augment(final_fit, new_data = pokemon1_testing) %>% conf_mat(truth=type_1, .pred_class) %
>%
  autoplot(type="heatmap")
```



The overall roc auc is not high with estimate value of 0.5769. The model does not do well since the overall accuracy is only about 0.30. The model has the best prediction on “normal” type, which is illustrated by the relatively large area under ROC curve. The model predicts worst on “psychic,” in which the ROC curve below the diagonal line. It might because that the data size of “normal” is relatively large, but data size of “psychic” type is relatively small, so that the model cannot capture the relation of features and response of “psychic”very well.

For 231 Students

Exercise 9

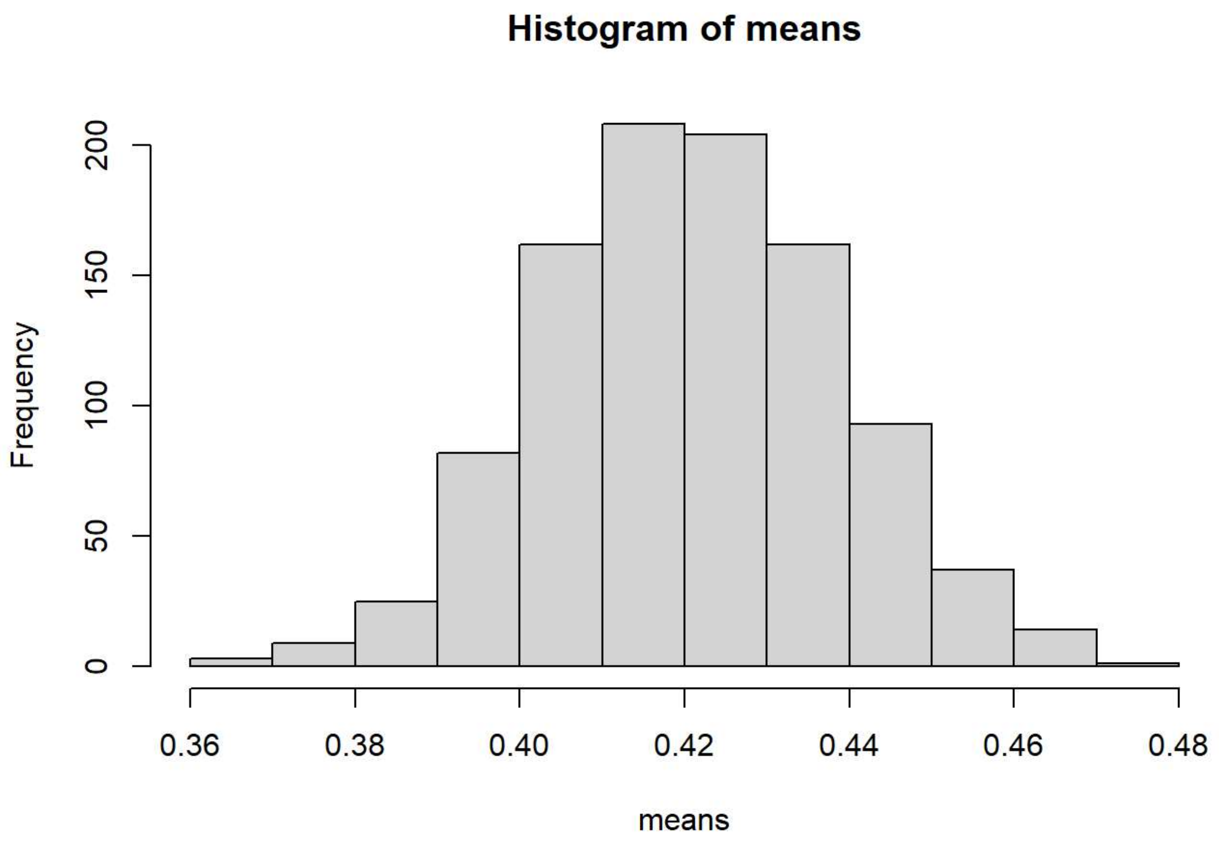
In the 2020-2021 season, Stephen Curry, an NBA basketball player, made 337 out of 801 three point shot attempts (42.1%). Use bootstrap resampling on a sequence of 337 1’s (makes) and 464 0’s (misses). For each bootstrap sample, compute and save the sample mean (e.g. bootstrap FG% for the player). Use 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

```
sq <- c(rep(1,times=337),rep(0,times=464))

n = 1000
means = numeric(n)
for (i in 1:1000){
  sample_i <- sample(sq,replace=TRUE)
  means[i] = mean(sample_i)
}

hist(means)
```



Hide

```
quantile(means,probs=c(0.005,0.995))
```

```
##      0.5%     99.5%
## 0.3770162 0.4656679
```