

PSTAT231 HW5 Cheng Ye

Cheng Ye

2022-11-20

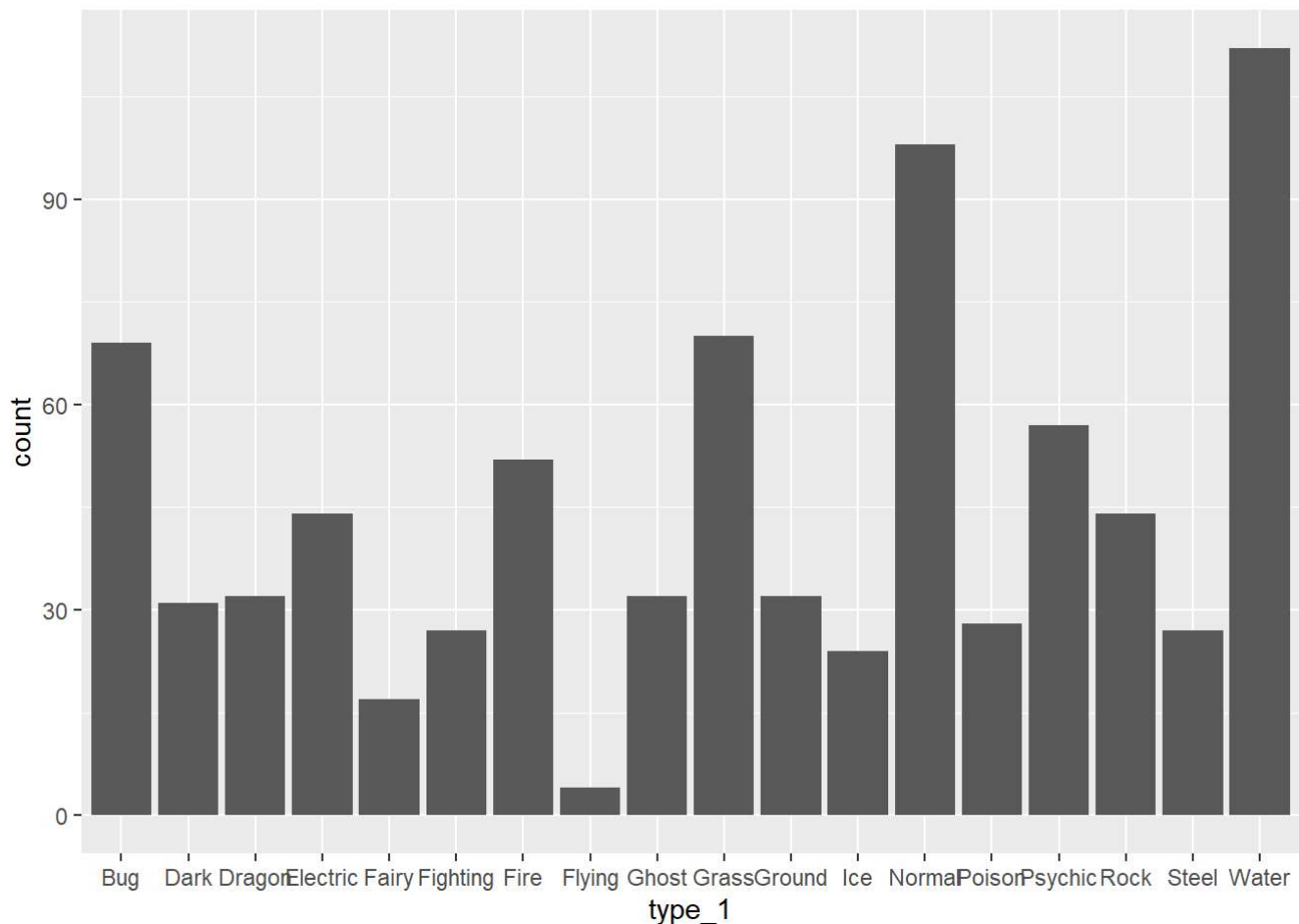
#Question 1

```
Pokemon_origin <- read.csv("C:/Cheng Ye/UCSB/PSTAT 231/HW/homework-5/homework-5/data/pokemon.csv")
#Pokemon_origin
Pokemon <- clean_names(Pokemon_origin)
#Pokemon

## From the description. By using clean_names(), the resulting column names are change to a form
at that only consist of the underscore, numbers, and letters. It is useful as in it makes callin
g variables easier and gets rid of unreadable characters.
```

#Question 2

```
Pokemon_plot <- ggplot(data=Pokemon, aes(x=type_1)) +
  geom_bar(stat = "count")
Pokemon_plot
```



```
Pokemon <- Pokemon[Pokemon$type_1 %in% c('Bug', 'Fire', 'Grass', 'Normal', 'Water', 'Psychic'),
]
Pokemon$type_1 = factor(Pokemon$type_1)
Pokemon$legendary = factor(Pokemon$legendary)
Pokemon$generation = factor(Pokemon$generation)

##There are 18 classes of outcome. There are very few pokemons belonging to the flying type.
```

#Question 3

```
set.seed(231)
Pokemon_split<-initial_split(Pokemon,strata = type_1,prop = 0.8)

Pokemon_train<-training(Pokemon_split)
Pokemon_test<-testing(Pokemon_split)
dim(Pokemon_train)
```

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

```
dim(Pokemon_test)
```

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

#From the results we could observe that the training and testing data sets have desired number of observations

#K-fold Cross Validation

```
Pokemon_folds<-vfold_cv(Pokemon_train, v = 5, strata = type_1)
#Stratifying the folds could be useful because it keeps the distribution, aka the proportion of variable types in each fold to be the same so that it is easier for us to analyze and avoid overfitting.
```

#Question 4

```
Pokemon_recipe <-
  recipe(formula = type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp
_def, data = Pokemon_train) %>%
  step_dummy(c('legendary', 'generation')) %>%
  step_normalize(all_predictors())
Pokemon_recipe %>%
  prep() %>%
  juice()
```

```
## # A tibble: 364 x 13
##   sp_atk attack    speed defense      hp sp_def type_1 legenda~1 gener~2 gener~3
##   <dbl> <dbl>    <dbl>    <dbl>    <dbl> <dbl> <fct>      <dbl>    <dbl>    <dbl>
## 1 -1.63 -1.36   -0.821   -1.15   -0.866 -1.75 Bug        -0.247   -0.417   -0.513
## 2 -1.48 -1.67   -1.34    -0.444 -0.690 -1.58 Bug        -0.247   -0.417   -0.513
## 3  0.565 -0.889  0.0367  -0.622 -0.338  0.353 Bug        -0.247   -0.417   -0.513
## 4 -1.63 -1.20   -0.650   -1.33   -1.04   -1.75 Bug        -0.247   -0.417   -0.513
## 5 -1.48 -1.52   -1.16    -0.622 -0.866 -1.58 Bug        -0.247   -0.417   -0.513
## 6 -0.848  0.524  0.208    -0.976 -0.162  0.353 Bug        -0.247   -0.417   -0.513
## 7 -1.79  2.41    2.61    -0.976 -0.162  0.353 Bug        -0.247   -0.417   -0.513
## 8 -0.848 -0.104 -1.51    -0.444 -1.22   -0.524 Bug        -0.247   -0.417   -0.513
## 9  0.565 -0.261  0.723    -0.267  0.0140  0.177 Bug        -0.247   -0.417   -0.513
## 10 -0.534  1.15    1.24      0.443  0.0140  0.353 Bug        -0.247   -0.417   -0.513
## # ... with 354 more rows, 3 more variables: generation_X4 <dbl>,
## #   generation_X5 <dbl>, generation_X6 <dbl>, and abbreviated variable names
## #   1: legendary_True, 2: generation_X2, 3: generation_X3
```

#Question 5

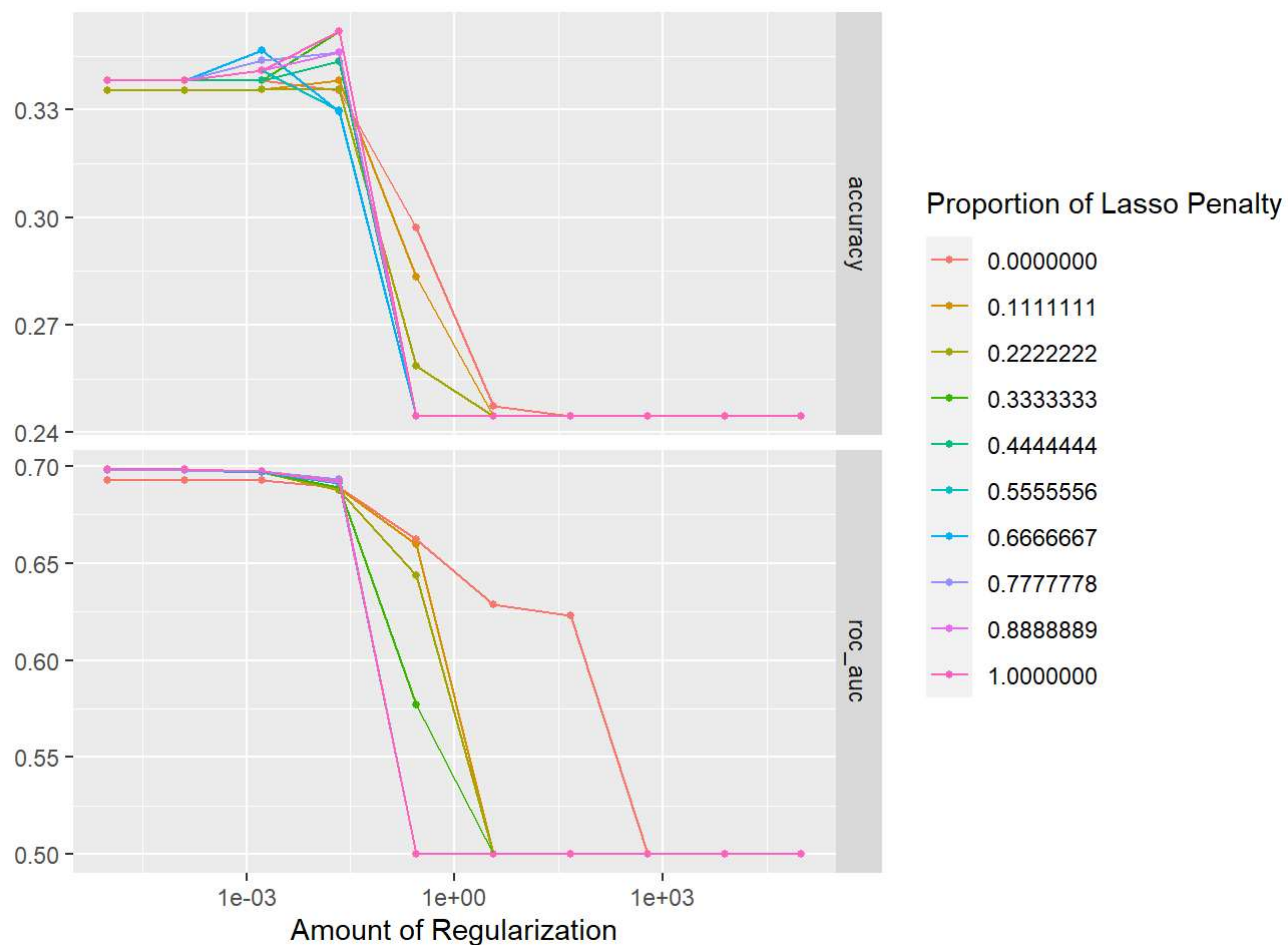
```
Mul_reg <- multinom_reg(penalty = tune(),mixture = tune()) %>%
  set_engine("glmnet")

Pokemon_wkflow <- workflow() %>%
  add_recipe(Pokemon_recipe) %>%
  add_model(Mul_reg)

Pokemon_grid <- grid_regular(penalty(range = c(-5, 5)),mixture(range=c(0,1)), levels = 10)
#Because the grid has 10 level penalty and 10 level mixture, and we set 5 folds, so there are 50
0 models to be fitted in total
```

#Question 6

```
Pokemon_tune <- tune_grid(object = Pokemon_wkflow,
  resamples = Pokemon_folds,
  grid = Pokemon_grid
)
autoplot(Pokemon_tune)
```



#From the results we could observe that the higher the penalty, the lower the accuracy. Larger/S maller value of regularization determines ROC_AUC and accuracy

#Question 7

```
best_penalty <- select_best(Pokemon_tune, metric = "roc_auc")
final_flow <- finalize_workflow(Pokemon_wkflow, best_penalty)
final_fit <- fit(final_flow, data = Pokemon_train)
aug_fit <- augment(final_fit, new_data = Pokemon_test)
aug_fit
```

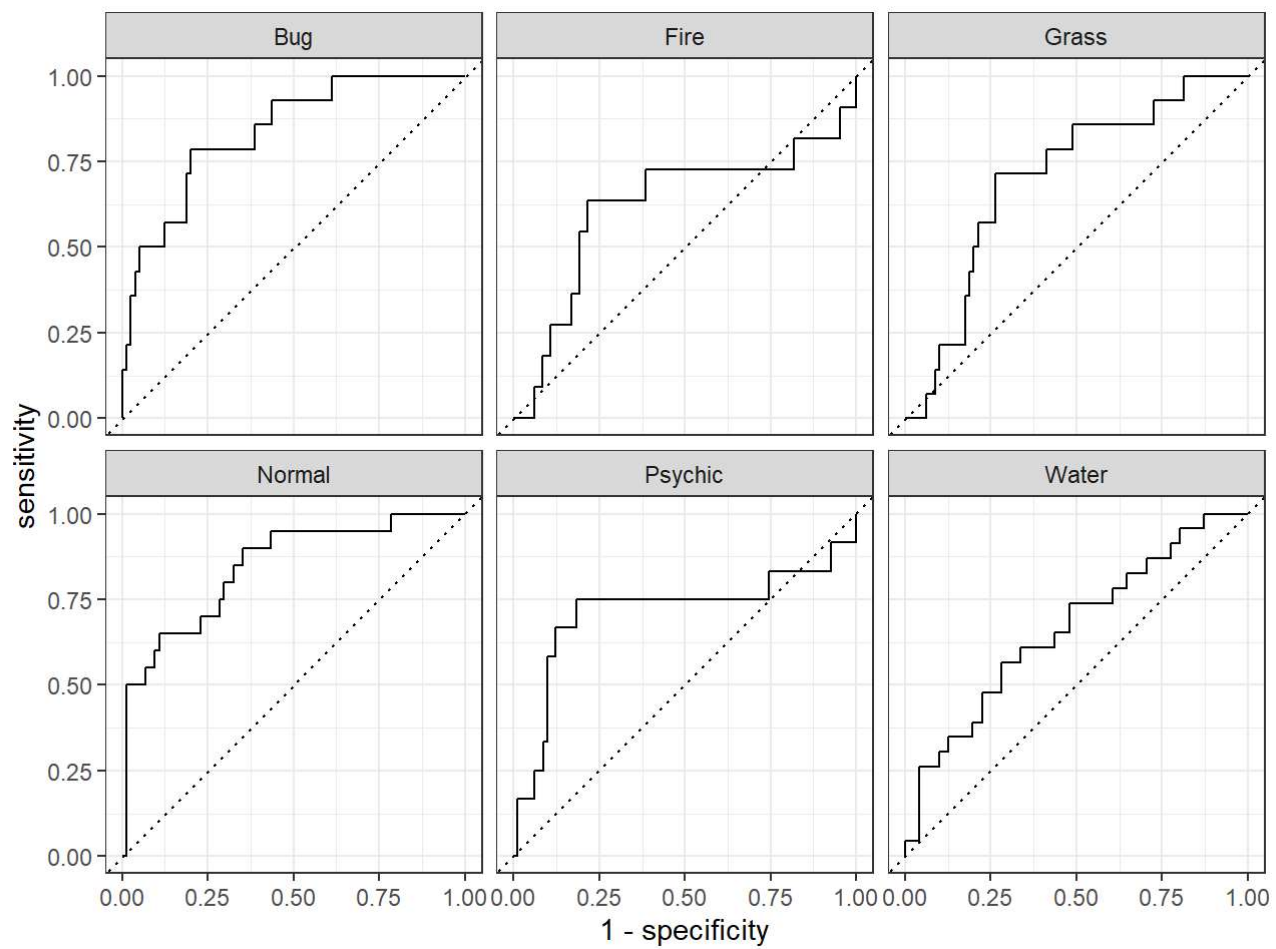
```
## # A tibble: 94 x 20
##       x name      type_1 type_2 total    hp attack defense sp_atk sp_def speed
##   <int> <chr>      <fct> <chr> <int> <int> <int> <int> <int> <int> <int>
## 1     1 1 Bulbasaur  Grass "Pois~ 318    45    49    49    65    65    45
## 2     3 3 VenusaurM~ Grass "Pois~ 625    80   100   123   122   120    80
## 3     5 5 Charmeleon Fire  ""    405    58    64    58    80    65    80
## 4     6 6 Charizard~ Fire  "Flyi~ 634    78   104    78   159   115   100
## 5    22 22 Fearow    Normal "Flyi~ 442    65    90    65    61    61   100
## 6    38 38 Ninetales Fire  ""    505    73    76    75    81   100   100
## 7    45 45 Vileplume Grass "Pois~ 490    75    80    85   110    90    50
## 8    47 47 Parasect Bug   "Gras~ 405    60    95    80    60    80    30
## 9    48 48 Venonat  Bug   "Pois~ 305    60    55    50    40    55    45
## 10   69 69 Bellsprout Grass "Pois~ 300    50    75    35    70    30    40
## # ... with 84 more rows, and 9 more variables: generation <fct>,
## #   legendary <fct>, .pred_class <fct>, .pred_Bug <dbl>, .pred_Fire <dbl>,
## #   .pred_Grass <dbl>, .pred_Normal <dbl>, .pred_Psychic <dbl>,
## #   .pred_Water <dbl>
```

#Question 8

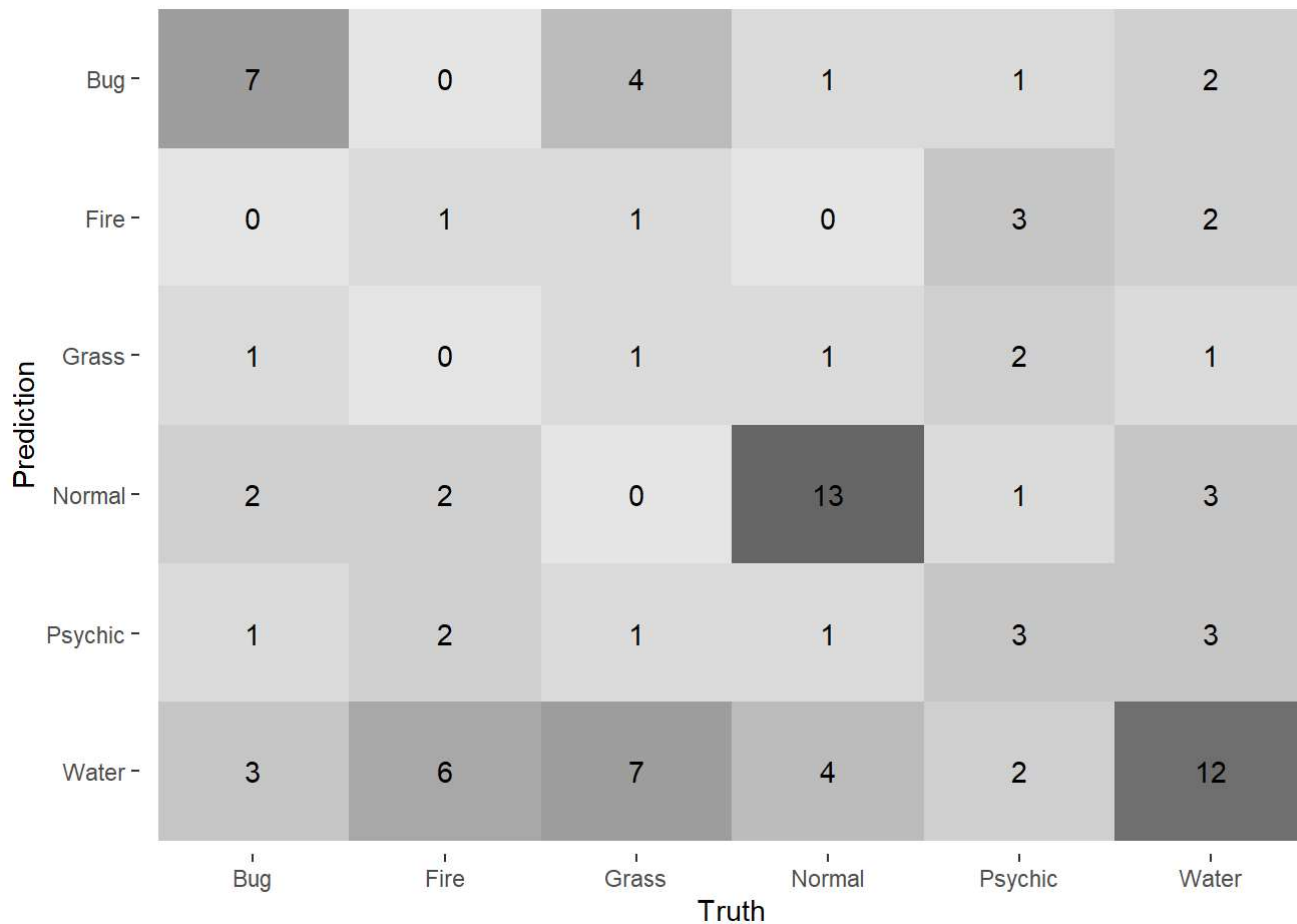
```
aug_fit <- augment(final_fit, new_data = Pokemon_test, type = "prob") %>%
  mutate(type_1 = as.factor(type_1))
Pokemon_roc <- roc_auc(aug_fit, truth = type_1 ,estimate = .pred_Bug:.pred_Water)
Pokemon_roc
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>      <dbl>
## 1 roc_auc hand_till    0.727
```

```
roc_curve(aug_fit, truth = type_1, estimate=.pred_Bug:.pred_Water) %>%
  autoplot()
```



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

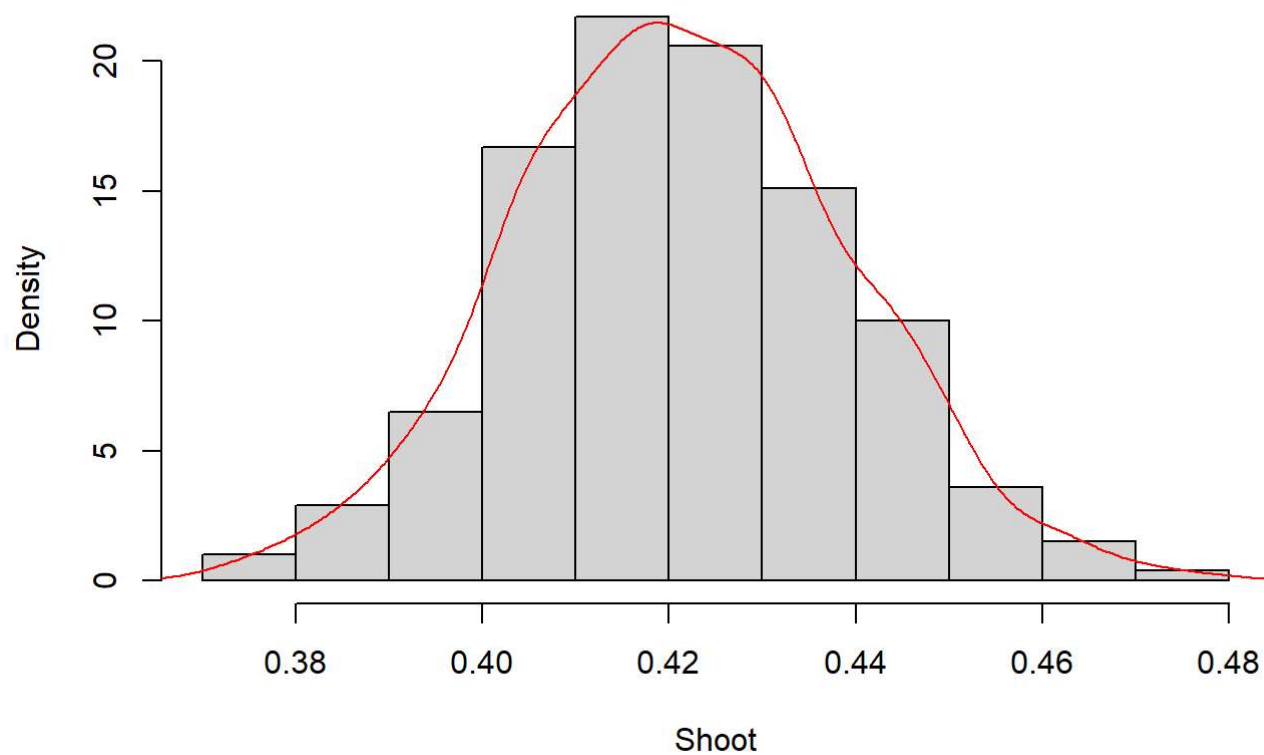


#From the results, we could observe that the model performs poorly in predicting grass and fire types. From the heat map, the model doesn't perform well besides predicting the normal type. I think the independent variables have little correlations to the response variable so that we cannot come up with a logical prediction between them.

#Question 9

```
set.seed(231)
library(boot)
Curry_shot <- c(rep(1,337),rep(0,464))
Curry_shot_mean <- function(original_vector, resample_vector) {
  mean(original_vector[resample_vector])
}
Curry_shots <- boot(Curry_shot, Curry_shot_mean, R=1000)
Shoot <- Curry_shots$t
Nine_CI <- boot.ci(Curry_shots, conf = 0.99)
hist(Shoot, freq = F)
lines(density(Shoot), col="red")
```

Histogram of Shoot



```
Nine_CI$normal
```

```
##      conf  
## [1,] 0.99 0.3742098 0.4661448
```

```
#From the results, we could observe that the 99%CI for bootstrap is (0.3742098 0.4661448)
```