Classification on CS:GO matches

Code

Andrew Gerungan

First we load the dataset (a set of CS:GO matches that includes map bans and their corresponding best_of)

Hide

```
df <- read.csv("picks.csv")
str(df)
```

```
'data.frame':	16035 obs. of  17 variables:
 $ date          : chr  "2020-03-18" "2020-03-18" "2020-03-18" "2020-03-17" ...
 $ team_1        : chr  "TeamOne" "Rugratz" "New England Whalers" "Complexity" ...
 $ team_2        : chr  "Recon 5" "Bad News Bears" "Station7" "forZe" ...
 $ inverted_teams: int  1 0 0 1 0 1 0 0 1 1 ...
 $ match_id      : int  2340454 2340453 2340461 2340279 2340456 2340397 2340130 2340131 2340443
2340444 ...
 $ event_id      : int  5151 5151 5243 5226 5247 5226 5243 5243 5236 5236 ...
 $ best_of       : chr  "3" "3" "1" "3" ...
 $ system        : int  123412 123412 121212 123412 123412 123412 121212 121212 123412 123412
...
 $ t1_removed_1  : chr  "Vertigo" "Dust2" "Mirage" "Inferno" ...
 $ t1_removed_2  : chr  "Train" "Nuke" "Dust2" "Nuke" ...
 $ t1_removed_3  : chr  "0.0" "0.0" "Vertigo" "0.0" ...
 $ t2_removed_1  : chr  "Nuke" "Mirage" "Nuke" "Overpass" ...
 $ t2_removed_2  : chr  "Overpass" "Train" "Train" "Vertigo" ...
 $ t2_removed_3  : chr  "0.0" "0.0" "Overpass" "0.0" ...
 $ t1_picked_1   : chr  "Dust2" "Vertigo" "0.0" "Dust2" ...
 $ t2_picked_1   : chr  "Inferno" "Inferno" "0.0" "Train" ...
 $ left_over     : chr  "Mirage" "Overpass" "Inferno" "Mirage" ...
```

We then isolate the columns that we are concerned about, those being best_of, 2 maps banned out by team 1 and 2 maps banned out by team 2. We also clean up best_of so that it’s easier to classify

Hide

```
df <- df[,c(7,9,10,12,13)] #grab best_of, t1_removed_1, t1_removed_2, t2_removed_1, t2_removed_
2, left_over


df$best_of <- factor(df$best_of)
df$t1_removed_1 <- factor(df$t1_removed_1)
df$t1_removed_2 <- factor(df$t1_removed_2)
df$t2_removed_1 <- factor(df$t2_removed_1)
df$t2_removed_2 <- factor(df$t2_removed_2)

levels(df$best_of)[levels(df$best_of)=="1(Online)"] <- "1"
levels(df$best_of)[levels(df$best_of)=="2(Online)"] <- "2"
levels(df$best_of)[levels(df$best_of)=="3(LAN)"] <- "3"
levels(df$best_of)[levels(df$best_of)=="3(Online)"] <- "3"
levels(df$best_of)[levels(df$best_of)=="3."] <- "3"
levels(df$best_of)[levels(df$best_of)=="of"] <- "3"

summary(df$t2_removed_2)
```

```
        0.0       Cache Cobblestone       Dust2     Inferno 
          6        1751        1057        1557        1995 
     Mirage        Nuke    Overpass       Train     Vertigo 
       2170        2098        2448        2330         623 
```

Hide

```
#df$left_over <- factor(df$left_over)

sapply(df, function(x) sum(is.na(x)==TRUE))
```

```
     best_of t1_removed_1 t1_removed_2 t2_removed_1 t2_removed_2 
           0            0            0            0            0 
```

Now we split our data into train/test (80/20)

Hide

```
set.seed(1234)
i <- sample(1:nrow(df), .8*nrow(df), replace=FALSE)
train <- df[i,]
test <- df[-i,]
```

Data Exploration

We run a summary on the dataset and each of its columns before checking the head/tail of best_of; I already have an idea of a relationship I want to check - some maps don’t see a lot of play outside of bo5s since certain maps are banned a lot. I want to see if I can determine what type of series the match is based on the maps banned - this should work for some of the more fringe maps (hopefully). I also checked the dim of the dataset and the head, and tail of the two columns

Hide

```
summary(train)
```

```
 best_of    t1_removed_1    t1_removed_2    t2_removed_1
 1:4539   Nuke    :2609   Overpass:2030   Nuke    :2569
 2: 171   Overpass:1984   Train   :1926   Overpass:1962
 3:8118   Train   :1699   Nuke    :1745   Train   :1625
          Cache   :1424   Mirage  :1575   Cache   :1564
          Mirage  :1252   Inferno :1523   Mirage  :1263
          Inferno :1203   Cache   :1428   Inferno :1173
          (Other) :2657   (Other) :2601   (Other) :2672
   t2_removed_2
 Overpass:1952
 Train   :1868
 Mirage  :1693
 Nuke    :1670
 Inferno :1611
 Cache   :1414
 (Other) :2620
```

Hide

```
dim(train)
```

```
[1] 12828     5
```

Hide

```
summary(train$best_of)
```

```
   1    2    3 
4539  171 8118 
```

Hide

```
summary(train$t1_removed_1)
```

```
      Cache Cobblestone       Dust2     Inferno      Mirage 
       1424         868         953        1203        1252 
       Nuke    Overpass       Train     Vertigo 
       2609        1984        1699         836 
```

Hide

```
summary(train$t1_removed_2)
```

```
        0.0       Cache Cobblestone       Dust2     Inferno 
          9        1428         831        1229        1523 
     Mirage        Nuke    Overpass       Train     Vertigo 
       1575        1745        2030        1926         532 
```

Hide

```
summary(train$t2_removed_1)
```

```
        0.0       Cache Cobblestone       Dust2     Inferno 
          1        1564         899         976        1173 
     Mirage        Nuke    Overpass       Train     Vertigo 
       1263        2569        1962        1625         796 
```

Hide

```
summary(train$t2_removed_2)
```

```
        0.0       Cache Cobblestone       Dust2     Inferno 
          6        1414         862        1266        1611 
     Mirage        Nuke    Overpass       Train     Vertigo 
       1693        1670        1952        1868         486 
```

Hide

```
#summary(train$left_over)
```

Hide

```
head(train$best_of)
```

```
[1] 3 3 3 3 3 3
Levels: 1 2 3
```

Hide

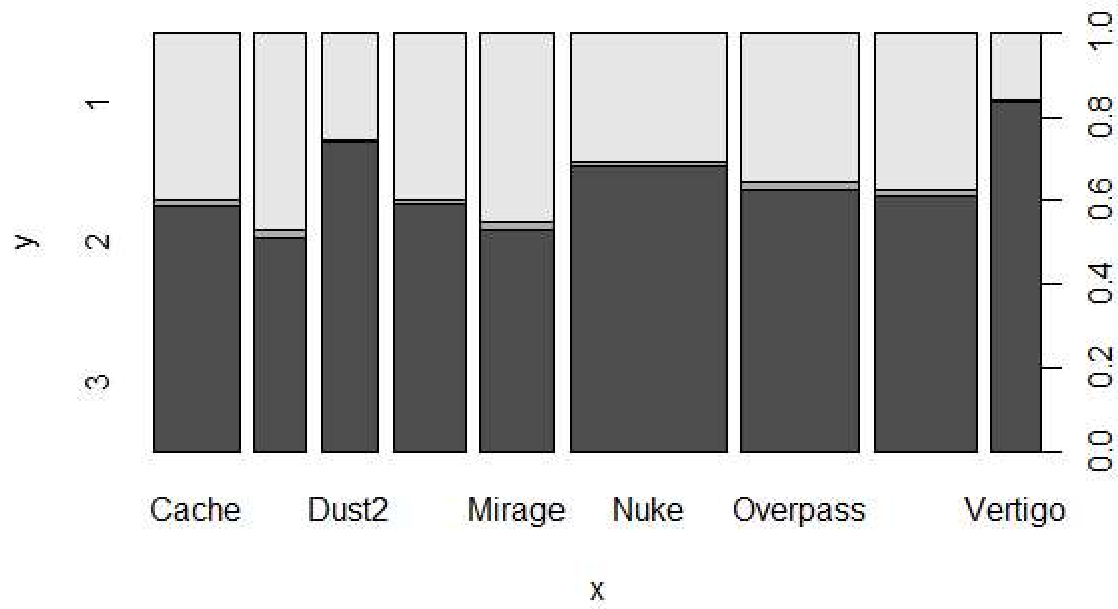
```
tail(train$best_of)
```

```
[1] 3 1 3 3 3 3
Levels: 1 2 3
```

We then plot t1's first removed map vs best_of to see if you can immediately guess the best_of series based off of the first removed map

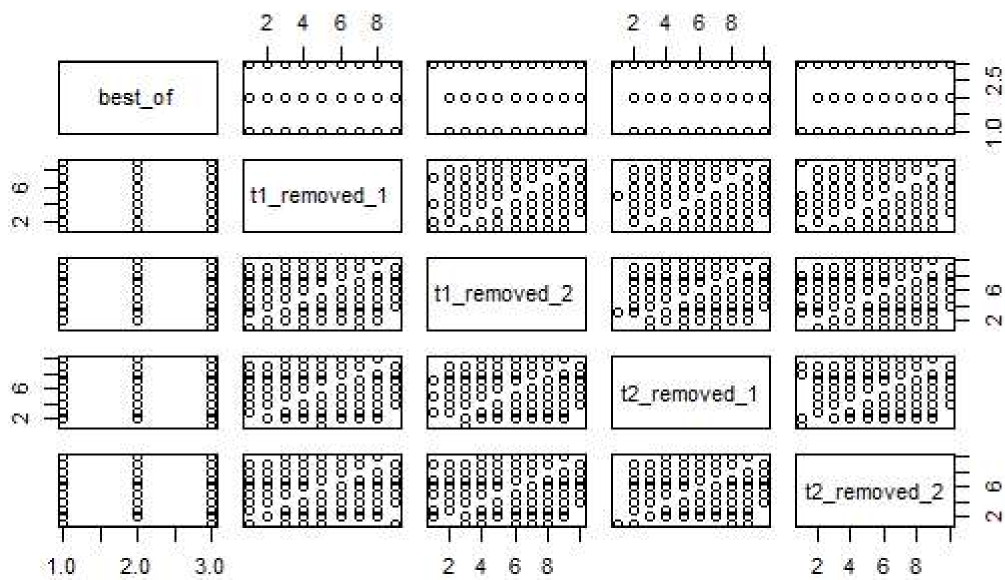
Hide

```
plot(x=train$t1_removed_1, y=train$best_of)
```



Hide

```
pairs(train)
```



Logistic Regression

We’re going to go ahead and run logistic regression on the type of series

Hide

```
glm1 <- glm(best_of~., data=train, family="binomial")
summary(glm1)
```

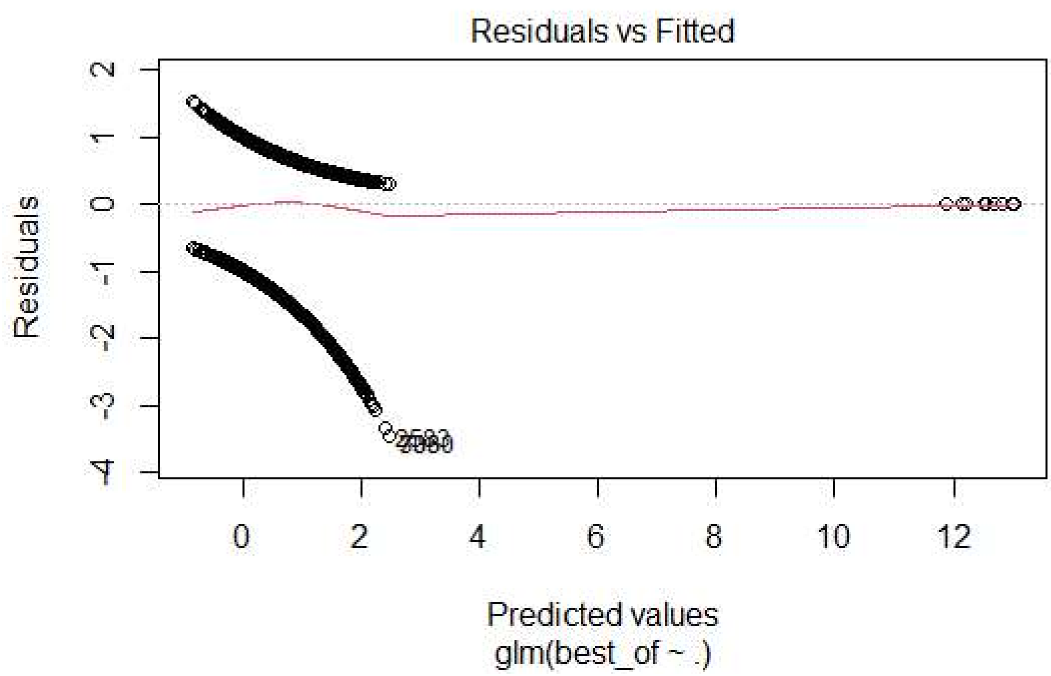
```
Call:
glm(formula = best_of ~ ., family = "binomial", data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.2601  -1.2323   0.7283   0.9596   1.5530

Coefficients:
                         Estimate Std. Error z value Pr(>|z|)
(Intercept)              25.59971  341.33566   0.075  0.94022
t1_removed_1Cobblestone  -0.43015    0.09346  -4.602 4.18e-06 ***
t1_removed_1Dust2         0.46461    0.09850   4.717 2.40e-06 ***
t1_removed_1Inferno      -0.27151    0.08662  -3.135  0.00172 **
t1_removed_1Mirage       -0.46711    0.08447  -5.530 3.20e-08 ***
t1_removed_1Nuke          0.42052    0.07754   5.423 5.86e-08 ***
t1_removed_1Overpass     -0.02270    0.07981  -0.284  0.77613
t1_removed_1Train        -0.16039    0.08089  -1.983  0.04737 *
t1_removed_1Vertigo       1.12594    0.11196  10.057  < 2e-16 ***
t1_removed_2Cache       -12.02304  105.12636  -0.114  0.90895
t1_removed_2Cobblestone -12.56653  105.12637  -0.120  0.90485
t1_removed_2Dust2       -11.67581  105.12637  -0.111  0.91157
t1_removed_2Inferno     -12.17319  105.12636  -0.116  0.90781
t1_removed_2Mirage      -12.28077  105.12636  -0.117  0.90700
t1_removed_2Nuke        -11.90474  105.12636  -0.113  0.90984
t1_removed_2Overpass    -12.13646  105.12636  -0.115  0.90809
t1_removed_2Train       -12.15725  105.12636  -0.116  0.90793
t1_removed_2Vertigo     -11.47997  105.12640  -0.109  0.91304
t2_removed_1Cache       -12.29523  324.74581  -0.038  0.96980
t2_removed_1Cobblestone -12.71707  324.74581  -0.039  0.96876
t2_removed_1Dust2       -12.06176  324.74581  -0.037  0.97037
t2_removed_1Inferno     -12.74815  324.74581  -0.039  0.96869
t2_removed_1Mirage      -12.75434  324.74581  -0.039  0.96867
t2_removed_1Nuke        -11.96697  324.74581  -0.037  0.97060
t2_removed_1Overpass    -12.38238  324.74580  -0.038  0.96958
t2_removed_1Train       -12.58697  324.74581  -0.039  0.96908
t2_removed_1Vertigo     -11.49529  324.74582  -0.035  0.97176
t2_removed_2Cache        -0.61203    1.16841  -0.524  0.60041
t2_removed_2Cobblestone  -0.94927    1.16911  -0.812  0.41681
t2_removed_2Dust2        -0.38740    1.16870  -0.331  0.74028
t2_removed_2Inferno      -0.79011    1.16806  -0.676  0.49877
t2_removed_2Mirage       -0.60972    1.16801  -0.522  0.60166
t2_removed_2Nuke         -0.46541    1.16790  -0.398  0.69026
t2_removed_2Overpass     -0.66613    1.16812  -0.570  0.56851
t2_removed_2Train        -0.66799    1.16789  -0.572  0.56734
t2_removed_2Vertigo      -0.14955    1.17190  -0.128  0.89845
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 16671  on 12827  degrees of freedom
```

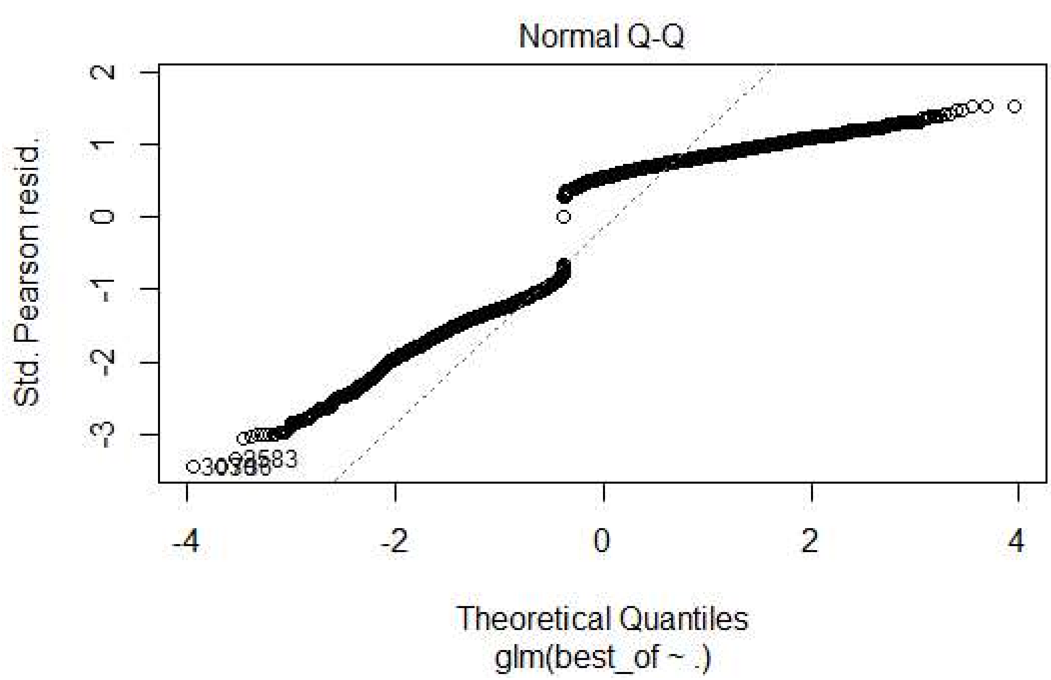
```
Residual deviance: 15807  on 12792  degrees of freedom
AIC: 15879

Number of Fisher Scoring iterations: 11
```

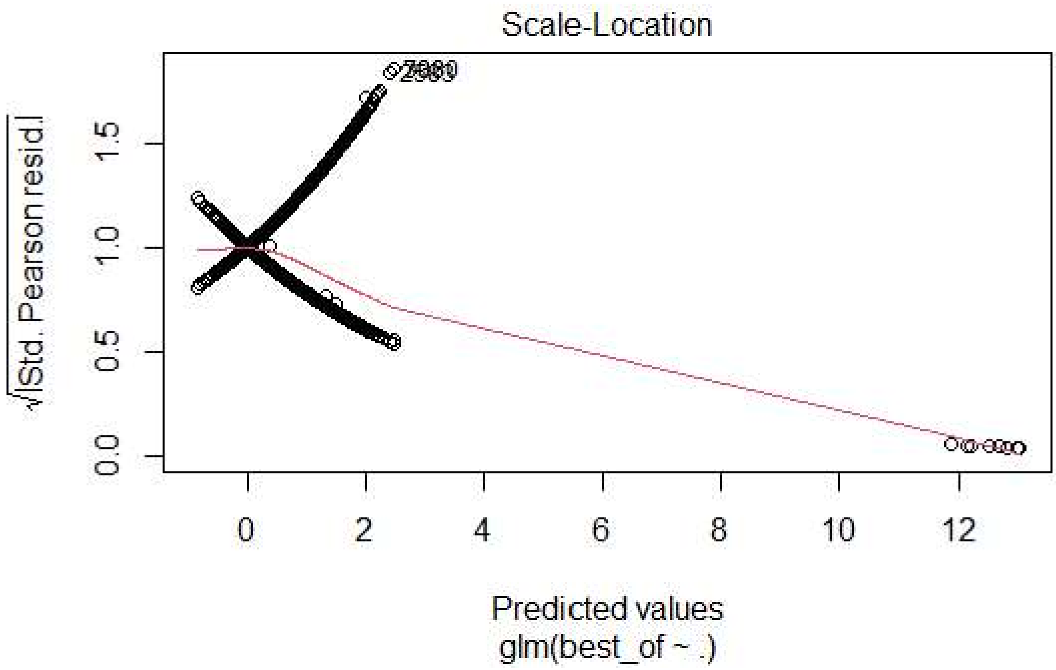
Hide

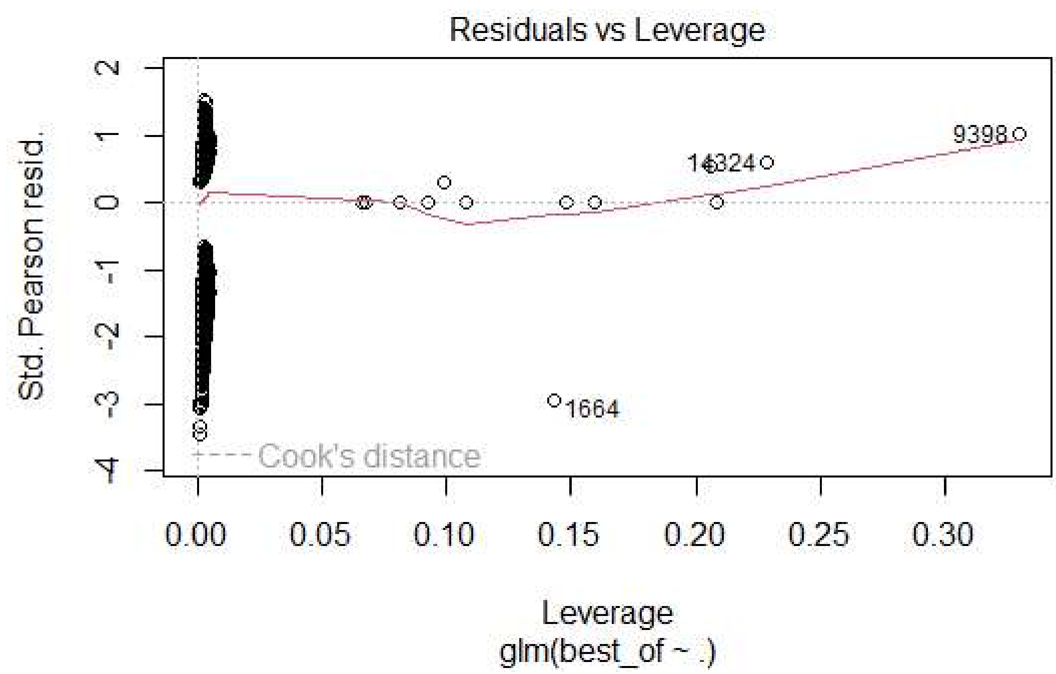
```
plot(glm1)
```

```
 警告:   てこ比 1 の観測データは図示しません:
  2619
```









…we got some interesting looking graphs and residuals, probably pointing towards logistic regression not being too hot for this idea

Hide

```
probs <- predict(glm1, newdata=test, type="response")
pred <- ifelse(probs>0.5, 1, 0)
acc <- mean(pred==test$best_of)
print(paste("accuracy=", acc))
```

```
[1] "accuracy= 0.283442469597755"
```

Hide

```
table(pred, test$best_of)
```

```

pred    1    2    3
   0  210   10  154
   1  909   44 1880
```

Wow the accuracy is bad but at least the levels are accurate.

Let’s move onto kNN predicting with classification - maybe that’ll be more accurate. We will need to change our other columns into numerics for this to work though

Hide

```
library(class)

#need to transfer to numeric
df$t1_removed_1 <- as.numeric(df$t1_removed_1)
df$t1_removed_2 <- as.numeric(df$t1_removed_2)
df$t2_removed_1 <- as.numeric(df$t2_removed_1)
df$t2_removed_2 <- as.numeric(df$t2_removed_2)

#re-randomize?
set.seed(1234)
i <- sample(1:nrow(df), .8*nrow(df), replace=FALSE)
train <- df[i,]
test <- df[-i,]

kNN_pred <- knn(train=train[2:5], test=test[2:5], cl=train$best_of, k=50)
```

Hide

```
results <- kNN_pred == test$best_of
acc <- length(which(results==TRUE)) / length(results)
table(results, kNN_pred)
```

```
       kNN_pred
results    1    2    3
  FALSE   73    0 1097
  TRUE    75    0 1962
```

Hide

```
acc
```

```
[1] 0.6351731
```

The accuracy is much better - still not near a threshold that I’d be confident with but much better than the logistic regression.

Finally, let's try using decision trees

Hide

```
library(tree)
tree1 <- tree(best_of~., data=train)
pred <- predict(tree1, newdata=test, type="class")
table(pred, test$best_of)
```

```


pred    1    2    3
   1    0    0    0
   2    0    0    0
   3 1119   54 2034
```

Hide

```
mean(pred==test$best_of)
```

```
[1] 0.6342376
```

A similar accuracy to kNN, meaning it's much better than the logistic regression.

Analysis

The difference between the results of logistic regression and both kNN and Decision trees is most likely due to the nature of the data - the presented idea of finding the best_of series based off of banned maps is inherently going to be harder to draw a definitive conclusion from basic logistic regression. Decision trees and kNN classification have a higher chance of having higher accuracy simply due to the better form of generalization it uses - logistic regression predicted alot of best of 3's compared to kNN and decision trees.