Tecnologies

Llorens

8/12/2019

# Motivation: Clash Royale

As I am a big fan of Clash Royale I am going to analyse the dynamics of this mobile game.

First of all, from wikipedia : “Clash Royale is a freemium real-time strategy video game developed and published by Supercell. The game combines elements from collectible card games, tower defense, and multiplayer online battle arena.”

In order to achive this analysis succesfully I would split this document in the following parts: Data collecting, Data wrangling or Data processing, Exploratory Data Analysis, Modelling and conclusions.

In this analysis I expect to show the usage of the cards and its winrate in the “Ladder” encounters. It is also interesting find out the interaction beetween cards.

Also, as I am a “Golem” player I will make a predictive model to predict the edge against other cards.

Finally I will make a predictive model in order to predict the number of crowns you can get depending the cards you play.

The source where I get the data: <https://developer.clashroyale.com/#/>

# Data collecting

I will not extend much explaining this part as is more interesting what comes later.

The data comes from an API, so there is the need to set up an account there and make a token verification. A token verification is a string you posses as an authorization to log in to a web an API, whatever it request. It is like your signature to be recorded.

From the API i want to get a battle log, that is a record of the battles that players have done. The API has its limitations, for example you can not request all the battles you like for a player, just the last 24. Also, you just can ask for 1000 player tags (players identification). Knowing this thought about how I am going to get the data. So, I will make three requests:

* A players list
* A battle log
* Cards list

I will make the requests in python as its been easier for me to connect to the API.

#import requests  
#import json  
#import pandas as pd  
  
# Players list  
  
#v1\_url\_ranking = "https://api.clashroyale.com/v1/locations/global/rankings/players"  
#header1\_Accept = "application/json"  
#v2\_header2\_token = "Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiIsImtpZCI6IjI4YTMxOGY3LTAwMDAtYTFlYi03ZmExLTJjNzQzM2M2Y2NhNSJ9.eyJpc3MiOiJzdXBlcmNlbGwiLCJhdWQiOiJzdXBlcmNlbGw6Z2FtZWFwaSIsImp0aSI6IjVkNzFiYWU4LTVjZTQtNGRlMS1iOTNjLTk4OGYzNWJmZjI4MyIsImlhdCI6MTU3NzYzMzA1OCwic3ViIjoiZGV2ZWxvcGVyL2U0ZmE0MTQ4LTZkNmUtY2VkNS1hOTBmLWVmNmI4YmZhOGMyMyIsInNjb3BlcyI6WyJyb3lhbGUiXSwibGltaXRzIjpbeyJ0aWVyIjoiZGV2ZWxvcGVyL3NpbHZlciIsInR5cGUiOiJ0aHJvdHRsaW5nIn0seyJjaWRycyI6WyI4MC4zOS4yMi4yNDgiXSwidHlwZSI6ImNsaWVudCJ9XX0.NZrKTdOBFUUupkaeA2HC44-PPoSP2fPkaGtrDs0gnT4qox\_PHcqABoQAaBo2SQnTvjWD4vX\_7Cuq\_EJC-98oMA"  
#num\_players = 1000  
  
#r1 = requests.get(v1\_url\_ranking, headers={"Accept":header1\_Accept, "authorization":v2\_header2\_token}, params = {"limit":num\_players})  
#t1 = json.loads(json.dumps(r1.json(), indent = 2)) # codigo json  
#players\_list = pd.DataFrame.from\_dict(t1["items"]).loc[:,["tag"]]  
#players\_list = players\_list.values.tolist()

# Battle log  
  
#battle\_log\_acum = pd.DataFrame()  
  
#for i in players\_list:  
 #tag\_player = str(i)[3:-2]  
 #v1\_url\_tag = "https://api.clashroyale.com/v1/players/%23"+tag\_player+"/battlelog"  
 #header1\_Accept = "application/json"  
 #v2\_header2\_token = "Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiIsImtpZCI6IjI4YTMxOGY3LTAwMDAtYTFlYi03ZmExLTJjNzQzM2M2Y2NhNSJ9.eyJpc3MiOiJzdXBlcmNlbGwiLCJhdWQiOiJzdXBlcmNlbGw6Z2FtZWFwaSIsImp0aSI6IjVkNzFiYWU4LTVjZTQtNGRlMS1iOTNjLTk4OGYzNWJmZjI4MyIsImlhdCI6MTU3NzYzMzA1OCwic3ViIjoiZGV2ZWxvcGVyL2U0ZmE0MTQ4LTZkNmUtY2VkNS1hOTBmLWVmNmI4YmZhOGMyMyIsInNjb3BlcyI6WyJyb3lhbGUiXSwibGltaXRzIjpbeyJ0aWVyIjoiZGV2ZWxvcGVyL3NpbHZlciIsInR5cGUiOiJ0aHJvdHRsaW5nIn0seyJjaWRycyI6WyI4MC4zOS4yMi4yNDgiXSwidHlwZSI6ImNsaWVudCJ9XX0.NZrKTdOBFUUupkaeA2HC44-PPoSP2fPkaGtrDs0gnT4qox\_PHcqABoQAaBo2SQnTvjWD4vX\_7Cuq\_EJC-98oMA"  
 #num\_bat = 30  
 #r2 = requests.get(v1\_url\_tag, headers={"Accept":header1\_Accept, "authorization":v2\_header2\_token}, params = {"limit":num\_bat})  
 #t2 = json.loads(json.dumps(r2.json(), indent = 2))  
 #battle\_log = pd.DataFrame(t2)  
 #battle\_log\_acum = battle\_log\_acum.append(battle\_log)

# battle\_log\_acum.to\_csv("battle\_log\_acum\_08122019.csv")

# Cards  
  
#v1\_url\_cards = "https://api.clashroyale.com/v1/cards"  
#header1\_Accept = "application/json"  
#v2\_header2\_token = "Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiIsImtpZCI6IjI4YTMxOGY3LTAwMDAtYTFlYi03ZmExLTJjNzQzM2M2Y2NhNSJ9.eyJpc3MiOiJzdXBlcmNlbGwiLCJhdWQiOiJzdXBlcmNlbGw6Z2FtZWFwaSIsImp0aSI6IjVkNzFiYWU4LTVjZTQtNGRlMS1iOTNjLTk4OGYzNWJmZjI4MyIsImlhdCI6MTU3NzYzMzA1OCwic3ViIjoiZGV2ZWxvcGVyL2U0ZmE0MTQ4LTZkNmUtY2VkNS1hOTBmLWVmNmI4YmZhOGMyMyIsInNjb3BlcyI6WyJyb3lhbGUiXSwibGltaXRzIjpbeyJ0aWVyIjoiZGV2ZWxvcGVyL3NpbHZlciIsInR5cGUiOiJ0aHJvdHRsaW5nIn0seyJjaWRycyI6WyI4MC4zOS4yMi4yNDgiXSwidHlwZSI6ImNsaWVudCJ9XX0.NZrKTdOBFUUupkaeA2HC44-PPoSP2fPkaGtrDs0gnT4qox\_PHcqABoQAaBo2SQnTvjWD4vX\_7Cuq\_EJC-98oMA"  
#num\_players = 10  
  
#r3 = requests.get(v1\_url\_cards, headers={"Accept":header1\_Accept, "authorization":v2\_header2\_token}, params = {"limit":num\_players})  
#t3 = json.loads(json.dumps(r3.json(), indent = 2))  
#cards = pd.DataFrame(t3)

Once I got the three files I save them as rds objects, in this way I can load them quickly and keep working with the same data instead to update again the requests.

#saveRDS(object = py$players\_list, file = "players.rds")  
#readRDS("players.rds")  
  
#saveRDS(object = py$battle\_log\_acum, file = "batallas.rds")  
#readRDS("batallas.rds")  
  
#saveRDS(object = py$cards, file = "cards.rds")  
#readRDS("cards.rds")

I can continue the analysis using the python object or the rds object.

#players <- py$players\_list  
players <- readRDS("players.rds")  
#batallas <- py$battle\_log\_acum  
batallas <- readRDS("batallas.rds")  
#cartas <- py$cards  
cartas <-readRDS("cards.rds")

# Data wrangling

Exploring the data, I can see as there are few list-columns that may complicate the manipulation.

glimpse(batallas)

## Observations: 25,498  
## Variables: 12  
## $ arena <list> [[54000039, "Legendary Arena"], [54000039,...  
## $ battleTime <chr> "20200104T193336.000Z", "20200104T193007.00...  
## $ challengeId <dbl> 6.5e+07, 6.5e+07, 6.5e+07, 6.5e+07, 6.5e+07...  
## $ challengeTitle <list> ["Grand Challenge", "Grand Challenge", "Gr...  
## $ challengeWinCountBefore <dbl> 2, 2, 2, 1, 0, 6, 5, 4, 4, 4, 3, 2, 1, 0, N...  
## $ deckSelection <chr> "collection", "collection", "collection", "...  
## $ gameMode <list> [[72000010, "Challenge"], [72000010, "Chal...  
## $ isLadderTournament <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, F...  
## $ opponent <list> [[["#RPJLUY28", "HELMI", 6623, 1, 3216, <9...  
## $ team <list> [[["#2QLL8UCCG", "<U+306F><U+3061>", 6302, 0, 4008, 2309...  
## $ tournamentTag <list> [NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, N...  
## $ type <chr> "challenge", "challenge", "challenge", "cha...

First, I filter for the “Ladder” battles as its the scoope of the analysis.

Then, I modify the variables ‘arena’ and ‘gameMode’. in order to get just the names of the arenas and the names of the gamemodes.

The list-columns of Opponent and Team is where there is almost the most important information for this analysis, the decks. I thought the better way to tidy the data is having one variable for each deck card, that means, the deck is compounded by 8 cards, then will need 8 variables for the Opponent and 8 variables for Team. Each variable will be a card as factor.

Also I removed the battles without trophy change, that means, the possible draws. I have now a clean dataset “batallas”. I removed “Opp” and “Team” to have a lighter workspace without superfluous objects.

batallas <- batallas %>% filter(do.call(rbind, batallas$gameMode)[,2] == "Ladder")  
   
batallas$arena <- do.call(rbind, batallas$arena)[,2]  
  
batallas$gameMode <- do.call(rbind, batallas$gameMode)[,2]   
  
Opp <- do.call(rbind, batallas$opponent)  
Opp <- Opp[,1]  
Opp <- tibble(Opp)  
Opp <- Opp %>% unnest\_wider(Opp)  
Opp <- Opp %>% unnest\_wider(cards)  
  
nombres <- paste("carta\_", seq(1:8),sep = "")  
names(Opp)[grep(pattern = "...[1-8]", names(Opp))] <- nombres  
  
Opp$carta\_1 <- Opp$carta\_1 %>% tibble(carta\_1 = .) %>% unnest\_wider(carta\_1) %>% .$name  
Opp$carta\_2 <- Opp$carta\_2 %>% tibble(carta\_2 = .) %>% unnest\_wider(carta\_2) %>% .$name  
Opp$carta\_3 <- Opp$carta\_3 %>% tibble(carta\_3 = .) %>% unnest\_wider(carta\_3) %>% .$name  
Opp$carta\_4 <- Opp$carta\_4 %>% tibble(carta\_4 = .) %>% unnest\_wider(carta\_4) %>% .$name  
Opp$carta\_5 <- Opp$carta\_5 %>% tibble(carta\_5 = .) %>% unnest\_wider(carta\_5) %>% .$name  
Opp$carta\_6 <- Opp$carta\_6 %>% tibble(carta\_6 = .) %>% unnest\_wider(carta\_6) %>% .$name  
Opp$carta\_7 <- Opp$carta\_7 %>% tibble(carta\_7 = .) %>% unnest\_wider(carta\_7) %>% .$name  
Opp$carta\_8 <- Opp$carta\_8 %>% tibble(carta\_8 = .) %>% unnest\_wider(carta\_8) %>% .$name  
  
Team <- do.call(rbind, batallas$team)  
Team <- Team[,1]  
Team <- tibble(Team)  
Team <- Team %>% unnest\_wider(Team)  
Team <- Team %>% unnest\_wider(cards)  
  
nombres <- paste("carta\_", seq(1:8),sep = "")  
names(Team)[grep(pattern = "...[1-8]", names(Team))] <- nombres  
  
Team$carta\_1 <- Team$carta\_1 %>% tibble(carta\_1 = .) %>% unnest\_wider(carta\_1) %>% .$name  
Team$carta\_2 <- Team$carta\_2 %>% tibble(carta\_2 = .) %>% unnest\_wider(carta\_2) %>% .$name  
Team$carta\_3 <- Team$carta\_3 %>% tibble(carta\_3 = .) %>% unnest\_wider(carta\_3) %>% .$name  
Team$carta\_4 <- Team$carta\_4 %>% tibble(carta\_4 = .) %>% unnest\_wider(carta\_4) %>% .$name  
Team$carta\_5 <- Team$carta\_5 %>% tibble(carta\_5 = .) %>% unnest\_wider(carta\_5) %>% .$name  
Team$carta\_6 <- Team$carta\_6 %>% tibble(carta\_6 = .) %>% unnest\_wider(carta\_6) %>% .$name  
Team$carta\_7 <- Team$carta\_7 %>% tibble(carta\_7 = .) %>% unnest\_wider(carta\_7) %>% .$name  
Team$carta\_8 <- Team$carta\_8 %>% tibble(carta\_8 = .) %>% unnest\_wider(carta\_8) %>% .$name  
   
batallas <- data.frame(batallas, Team, Opp[,8:15]) %>% filter(is.na(trophyChange)==F)  
  
rm(Opp, Team)

Now, the data is tidy and clean. Next step is factorise the cards columns. The file “cartas” has the information needed.

glimpse(cartas)

## Observations: 97  
## Variables: 1  
## $ items <list> [["Knight", 26000000, 13, ["https://api-assets.clashroyale.c...

nombres\_cartas <- as.character(trimws(as\_vector(unnest\_wider(cartas, items)[,1])))  
id\_cartas <- as.character(unnest\_wider(cartas, items)$id)  
  
# Factorise  
batallas$carta\_1 <- factor(batallas$carta\_1, levels = nombres\_cartas)  
batallas$carta\_2 <- factor(batallas$carta\_2, levels = nombres\_cartas)  
batallas$carta\_3 <- factor(batallas$carta\_3, levels = nombres\_cartas)  
batallas$carta\_4 <- factor(batallas$carta\_4, levels = nombres\_cartas)  
batallas$carta\_5 <- factor(batallas$carta\_5, levels = nombres\_cartas)  
batallas$carta\_6 <- factor(batallas$carta\_6, levels = nombres\_cartas)  
batallas$carta\_7 <- factor(batallas$carta\_7, levels = nombres\_cartas)  
batallas$carta\_8 <- factor(batallas$carta\_8, levels = nombres\_cartas)  
batallas$carta\_1.1 <- factor(batallas$carta\_1.1, levels = nombres\_cartas)  
batallas$carta\_2.1 <- factor(batallas$carta\_2.1, levels = nombres\_cartas)  
batallas$carta\_3.1 <- factor(batallas$carta\_3.1, levels = nombres\_cartas)  
batallas$carta\_4.1 <- factor(batallas$carta\_4.1, levels = nombres\_cartas)  
batallas$carta\_5.1 <- factor(batallas$carta\_5.1, levels = nombres\_cartas)  
batallas$carta\_6.1 <- factor(batallas$carta\_6.1, levels = nombres\_cartas)  
batallas$carta\_7.1 <- factor(batallas$carta\_7.1, levels = nombres\_cartas)  
batallas$carta\_8.1 <- factor(batallas$carta\_8.1, levels = nombres\_cartas)  
batallas$crowns <- as.factor(batallas$crowns)  
batallas <- batallas %>% mutate(vicder = factor(ifelse(batallas$trophyChange>0, 1, 0), levels = c(0, 1)))

# Exploratory Data Analysis

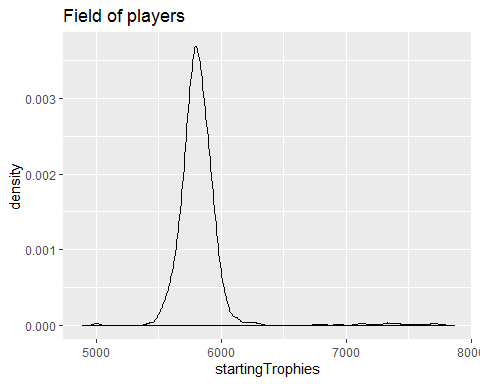
As later I am going to build a model to predict the number of crowns its important see how is distributed the variable and its possible predictors.

The Data collected in the previous step is obtained for the first 1000 players in the ranking. This means, these battles might have a little bias through win, so I expect to find more victories than defeats.

Another speculation I might do is about the crowns distribution. My “know-how” of the game tells me, that I am not able to win always, that when I win most of the times I have lost one tower but that is because I play “Golem” so, my intuition here is the distribution depends on the decks played, however my bet is I will find more battles won by 1 crown than the other cases together.

It’s interesting to see which cards are the most used and less used, also see where Golem stands.

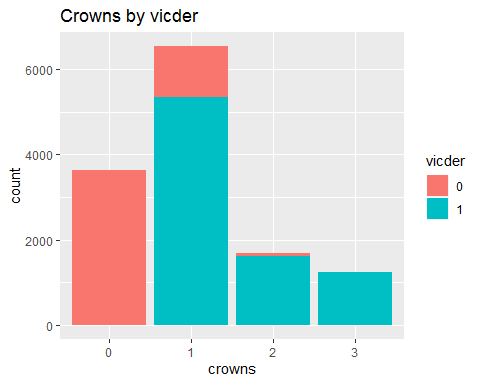
ggplot(batallas, aes(x = startingTrophies))+  
 geom\_density()+  
 ggtitle("Field of players")



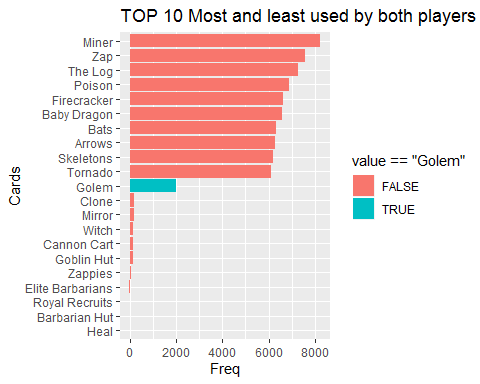
summary(batallas$vicder)

## 0 1   
## 4882 8215

ggplot(batallas, aes(x = crowns, fill = vicder))+  
 geom\_histogram(stat = "count")+  
 ggtitle("Crowns by vicder")



rbind(batallas %>%  
 select(contains("carta\_")) %>%   
 gather(key = 'key', value = 'value') %>%  
 count(value) %>%  
 arrange(desc(n)) %>%  
 head(n = 10),  
 batallas %>%  
 select(contains("carta\_")) %>%   
 gather(key = 'key', value = 'value') %>%  
 count(value) %>%  
 arrange(desc(n)) %>%  
 filter(value == "Golem"),  
 batallas %>%   
 select(contains("carta\_")) %>%   
 gather(key = 'key', value = 'value') %>%  
 count(value) %>%  
 arrange(desc(n)) %>%  
 tail(n = 10)) %>%  
 ggplot(aes(x = reorder(value, n), y = n, fill = value == "Golem"))+  
 geom\_col()+  
 coord\_flip()+  
 ggtitle('TOP 10 Most and least used by both players')+  
 xlab("Cards")+  
 ylab("Freq")



I didn’t fail in my speculations, first the field of players is pretty high, the average is near 5700 trophies (just to compare myself was arround 4900). The data was collected in the beggining of season 6, early december. When the season ends, the TOP 3 can be arround 8000 trophies.

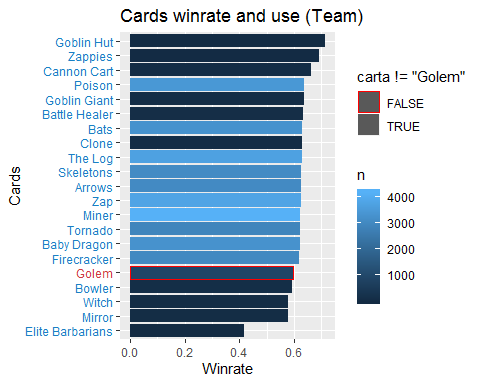
The number of victories is much bigger than the defeats.

The crowns distribution when “Team” wins is likely I said before, 1 crown is the larger.

Finally, the “Miner”, “Zap” and “The log” are the most used. “Golem” stands quite far from the TOP sadly.

To continue exploring the dynamics behind the crowns distribution, I think that the card’s winrate could explain something.

newdata <- batallas %>%  
 select(-contains('.')) %>%  
 pivot\_longer(cols = contains('carta\_'), names\_to = 'key', values\_to = 'carta') %>%  
 select('carta', 'vicder') %>%  
 count(carta, vicder) %>%   
 spread(key = 'vicder', value = 'n') %>%  
 na.fill(0) %>%  
 as.data.frame() %>%  
 plyr::rename(c("0" = "der", "1" = "vic")) %>%  
 mutate(color = ifelse(carta == "Golem", "#CC3D3D", "#1A80C4"))  
  
newdata[,2:3] <- unfactor(newdata[,2:3])  
  
newdata <- rbind(newdata %>%   
 group\_by(carta) %>%  
 mutate(win = vic/(vic+der), n = vic+der) %>%  
 select(-c('der', 'vic')) %>%  
 arrange(desc(n)) %>%  
 head(n=10),  
 newdata %>%   
 group\_by(carta) %>%  
 mutate(win = vic/(vic+der), n = vic+der) %>%  
 select(-c('der', 'vic')) %>%  
 arrange(desc(n)) %>%  
 filter(carta == "Golem"),  
 newdata %>%   
 group\_by(carta) %>%  
 mutate(win = vic/(vic+der), n = vic+der) %>%  
 select(-c('der', 'vic')) %>%  
 arrange(desc(n)) %>%  
 tail(n=10)) %>%  
 arrange(desc(win))  
  
ggplot(newdata, aes(x = reorder(carta, win), y = win, fill = n, color = carta != "Golem"))+  
 geom\_col()+  
 scale\_color\_manual(values = c("red", "#00ff0000"))+  
 coord\_flip()+  
 ggtitle('Cards winrate and use (Team)')+  
 xlab("Cards")+  
 ylab("Winrate")+  
 theme(axis.text.y = element\_text(color = newdata$color[order(newdata$win)]))



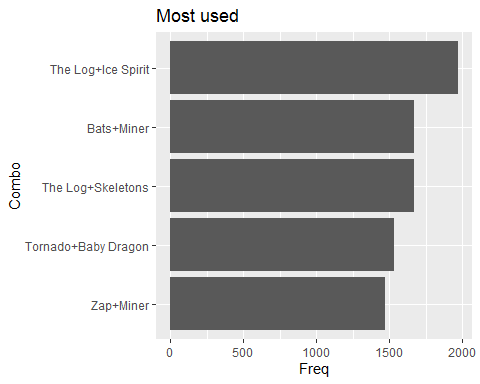
summary(newdata$win)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.4167 0.6168 0.6250 0.6184 0.6328 0.7143

I see above as the winrate goes from a 70% at top and decreases to 40% to the bottom and its average is 61%. Maybe I could think that the more winrate has a card more likely to win by three crowns. However, this effect disapear if the opponent also play the same cards. It’s necessary keep exploring.

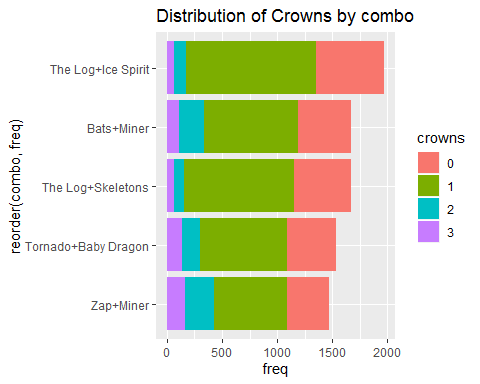
Let’s continue investigating what is behind the intuition said before “the number of crowns depends on the decks”. Maybe the interaction or the association of many cards makes the difference.

# Dumieando cartas Team  
newdata <- batallas %>%  
 select(-contains(".")) %>%  
 select(contains("carta\_")) %>%  
 mutate(carta\_1 = factor(batallas$carta\_1, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 carta\_2 = factor(batallas$carta\_2, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 carta\_3 = factor(batallas$carta\_3, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 carta\_4 = factor(batallas$carta\_4, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 carta\_5 = factor(batallas$carta\_5, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 carta\_6 = factor(batallas$carta\_6, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 carta\_7 = factor(batallas$carta\_7, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 carta\_8 = factor(batallas$carta\_8, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 mazo = paste(carta\_1, carta\_2, carta\_3, carta\_4, carta\_5, carta\_6, carta\_7, carta\_8, sep = ", "))  
  
#Onehotencode  
m <- matrix(str\_match(newdata$mazo[1], pattern = id\_cartas), ncol = length(nombres\_cartas))  
for (i in 2:nrow(newdata)){  
 p = matrix(str\_match(newdata$mazo[i], pattern = id\_cartas), ncol = length(nombres\_cartas))  
 m = rbind(m, p)  
}  
m <- as.data.frame(m)  
names(m) <- nombres\_cartas  
m <- ifelse(is.na(m), 0, 1) %>% as.data.frame()  
  
# Dumieando cartas Opponent  
newdata <- batallas %>%  
 select(contains("carta\_")) %>%  
 select(contains(".")) %>%  
 mutate(carta\_1.1 = factor(batallas$carta\_1, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 carta\_2.1 = factor(batallas$carta\_2, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 carta\_3.1 = factor(batallas$carta\_3, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 carta\_4.1 = factor(batallas$carta\_4, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 carta\_5.1 = factor(batallas$carta\_5, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 carta\_6.1 = factor(batallas$carta\_6, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 carta\_7.1 = factor(batallas$carta\_7, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 carta\_8.1 = factor(batallas$carta\_8, levels = nombres\_cartas, labels = unnest\_wider(cartas, items)$id),  
 mazo = paste(carta\_1.1, carta\_2.1, carta\_3.1, carta\_4.1, carta\_5.1, carta\_6.1, carta\_7.1, carta\_8.1, sep = ", "))  
  
#Onehotencode  
o <- matrix(str\_match(newdata$mazo[1], pattern = id\_cartas), ncol = length(nombres\_cartas))  
for (i in 2:nrow(newdata)){  
 p = matrix(str\_match(newdata$mazo[i], pattern = id\_cartas), ncol = length(nombres\_cartas))  
 o = rbind(o, p)  
}  
o <- as.data.frame(o)  
names(o) <- paste(nombres\_cartas, "\_Opp", sep = "")  
o <- ifelse(is.na(o), 0, 1) %>% as.data.frame()  
  
#Creamos matriz interacciones  
mm <- matrix(ncol = length(m), nrow = length(m))  
for (i in 1:length(m)) {  
 for (j in 1:length(m)) {  
 mm[i, j] <- as.integer(count(filter(m, m[,i]\*m[,j] == 1)))   
 }  
}  
  
mm <- mm %>% data.frame(row.names = nombres\_cartas)   
names(mm) <- nombres\_cartas  
mm[upper.tri(mm, diag = T)] <- 0  
  
mm %>%   
 mutate(fila = rownames(mm)) %>%  
 gather(-'fila', key = 'key', value = 'value') %>%  
 filter(fila != key) %>%   
 arrange(desc(value)) %>%  
 mutate(combo = paste(fila, key, sep = "+")) %>%  
 select(combo, value) %>%  
 head(n = 5) %>%  
 ggplot(aes(x = reorder(combo, value), y = value))+  
 xlab("Combo")+  
 ylab("Freq")+  
 geom\_col()+  
 coord\_flip()+  
 ggtitle("Most used")



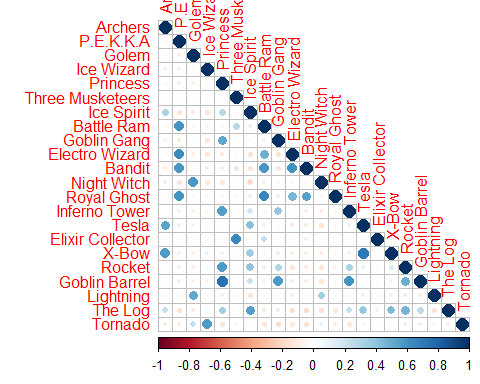
Above I see the TOP 10 combination most used, that doesn’t say anything about the crowns distribution. However is a good scoope to investigate.

rbind(cbind(batallas$vicder, batallas$crowns, m) %>%  
 as.data.frame() %>%  
 plyr::rename(c("batallas$vicder" = "vicder", "batallas$crowns" = "crowns")) %>%  
 filter(Zap == 1, Miner == 1) %>%  
 group\_by(crowns) %>%  
 summarise(freq = n()) %>%  
 mutate(combo = "Zap+Miner"),  
cbind(batallas$vicder, batallas$crowns, m) %>%  
 as.data.frame() %>%  
 plyr::rename(c("batallas$vicder" = "vicder", "batallas$crowns" = "crowns")) %>%  
 filter(Tornado == 1, `Baby Dragon` == 1) %>%  
 group\_by(crowns) %>%  
 summarise(freq = n()) %>%  
 mutate(combo = "Tornado+Baby Dragon"),  
cbind(batallas$vicder, batallas$crowns, m) %>%  
 as.data.frame() %>%  
 plyr::rename(c("batallas$vicder" = "vicder", "batallas$crowns" = "crowns")) %>%  
 filter(`The Log` == 1, Skeletons == 1) %>%  
 group\_by(crowns) %>%  
 summarise(freq = n()) %>%  
 mutate(combo = "The Log+Skeletons"),  
cbind(batallas$vicder, batallas$crowns, m) %>%  
 as.data.frame() %>%  
 plyr::rename(c("batallas$vicder" = "vicder", "batallas$crowns" = "crowns")) %>%  
 filter(Bats == 1, Miner == 1) %>%  
 group\_by(crowns) %>%  
 summarise(freq = n()) %>%  
 mutate(combo = "Bats+Miner"),  
cbind(batallas$vicder, batallas$crowns, m) %>%  
 as.data.frame() %>%  
 plyr::rename(c("batallas$vicder" = "vicder", "batallas$crowns" = "crowns")) %>%  
 filter(`The Log` == 1, `Ice Spirit` == 1) %>%  
 group\_by(crowns) %>%  
 summarise(freq = n()) %>%  
 mutate(combo = "The Log+Ice Spirit")) %>%  
ggplot(aes(x = reorder(combo, freq), y = freq, fill = crowns))+  
 geom\_col()+  
 coord\_flip()+  
 ggtitle("Distribution of Crowns by combo")



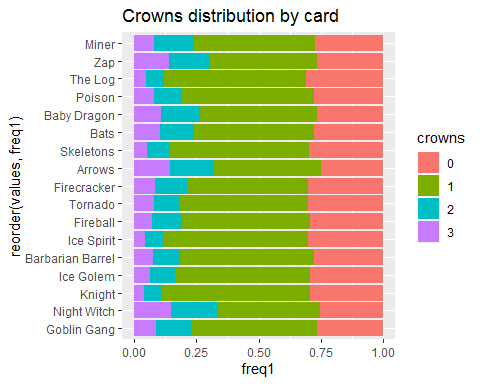
I see a little trend here, as much is the combo used, much percentage of get 0 and 1 crown. However I wouldn’t belive that conclusion as the sample is small. Another way to understand the interaction between cards is by the correlation. I filtered the cards that have at least a correlation above 0.5 with one other random card.

newdata <- as.data.frame(cbind(m, o))  
  
correlation <- cor(newdata[,c(1:97)])  
correlation[is.na(correlation)] <- 0   
diag(correlation) <- 0  
correlation <- correlation %>%  
 as.data.frame() %>%  
 select\_if(apply(., 2, max)> 0.5 & apply(., 2, max) != 1) %>%  
 .[apply(., 1, max) > 0.5,]   
diag(correlation) <- 1  
corrplot::corrplot(as.matrix(correlation), method = "circle", type = "lower")



Let’s conclude the exploring part with the crowns distribution by card. I focus on the scoope of the cards used more than 2k times. Here is clear which cards contribute more to get 2 or 3 crowns: zap, arrows, night witch.

newdata <- batallas %>%  
 select(crowns, contains("carta")) %>%  
 select(-contains(".")) %>%  
 pivot\_longer(-c(crowns), names\_to = "carta", values\_to = "values") %>%  
 select(-carta) %>%  
 group\_by(values, crowns) %>%  
 summarise(freq = n()) %>%   
 pivot\_wider(names\_from = crowns, values\_from = freq) %>%  
 na.fill(0) %>%  
 as.data.frame()  
  
newdata[,2:5] <- unfactor(newdata[,2:5])  
  
newdata %>%   
 mutate(freq = newdata$`0`+newdata$`1`+newdata$`2`+newdata$`3`) %>%  
 filter(freq > 2000) %>%  
 pivot\_longer(-c(values, freq), names\_to = "crowns", values\_to = "freq1") %>%  
 select(-freq) %>%  
 ggplot(aes(x = reorder(values, freq1), y = freq1, fill = crowns))+  
 geom\_col(position = "fill")+  
 coord\_flip()+  
 ggtitle("Crowns distribution by card")



# Modelling

I already have an idea of how the data is distributed, now is the time to make predictions. In the beginning of the document I said to build a model able to predict the number of crowns.

First let’s preprocess our data before fit the model. Then split the data into 75% for train and 25% test, also set the seed.

# Modeling   
newdata <- batallas %>%   
 dplyr::select(vicder, crowns, contains("carta\_")) %>%  
 mutate(carta\_1 = as.integer(factor(batallas$carta\_1, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 carta\_2 = as.integer(factor(batallas$carta\_2, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 carta\_3 = as.integer(factor(batallas$carta\_3, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 carta\_4 = as.integer(factor(batallas$carta\_4, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 carta\_5 = as.integer(factor(batallas$carta\_5, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 carta\_6 = as.integer(factor(batallas$carta\_6, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 carta\_7 = as.integer(factor(batallas$carta\_7, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 carta\_8 = as.integer(factor(batallas$carta\_8, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 mazo = paste(carta\_1, carta\_2, carta\_3, carta\_4, carta\_5, carta\_6, carta\_7, carta\_8, sep = ", "),  
 carta\_1.1 = as.integer(factor(batallas$carta\_1.1, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 carta\_2.1 = as.integer(factor(batallas$carta\_2.1, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 carta\_3.1 = as.integer(factor(batallas$carta\_3.1, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 carta\_4.1 = as.integer(factor(batallas$carta\_4.1, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 carta\_5.1 = as.integer(factor(batallas$carta\_5.1, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 carta\_6.1 = as.integer(factor(batallas$carta\_6.1, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 carta\_7.1 = as.integer(factor(batallas$carta\_7.1, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 carta\_8.1 = as.integer(factor(batallas$carta\_8.1, levels = nombres\_cartas, labels = c(1:length(nombres\_cartas)))),  
 mazo.1 = paste(carta\_1.1, carta\_2.1, carta\_3.1, carta\_4.1, carta\_5.1, carta\_6.1, carta\_7.1, carta\_8.1, sep = ", ")) %>%  
 dplyr::select(-mazo, -mazo.1)  
  
# Split dataset  
set.seed(101)  
split = sample.split(newdata$crowns, SplitRatio = 0.75)  
training\_set = subset(newdata, split == T)  
testing\_set = subset(newdata, split == F)

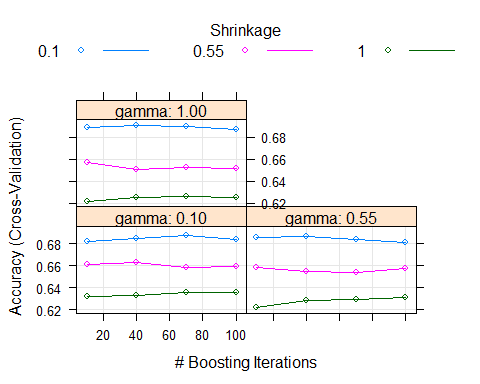
First model is a Extrem Gradient Boosting where I do a manual search of the hyperparameters with 5 folds CV and plot the results. I let the variable ‘vicder’ to help the models to predict.

# XGBoost  
set.seed(101)  
control <- trainControl(method = "cv",   
 number = 5,  
 search = 'grid',  
 verboseIter = T)  
  
grid <- expand.grid(nrounds = seq(10,100, length.out = 4),  
 max\_depth = 17,  
 eta = seq(0.1, 1, length.out = 3),  
 gamma = seq(0.1, 1, length.out = 3),  
 colsample\_bytree = 1,  
 min\_child\_weight = 1,  
 subsample = 0.5)  
  
  
  
#Classifier with train  
classifier.xgb <- train(data = training\_set,  
 crowns~. ,  
 method = "xgbTree",   
 trControl = control,  
 tuneGrid = grid,  
 metric = "Accuracy")

## + Fold1: eta=0.10, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold1: eta=0.10, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold1: eta=0.10, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold1: eta=0.10, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold1: eta=0.10, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold1: eta=0.10, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold1: eta=0.55, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold1: eta=0.55, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold1: eta=0.55, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold1: eta=0.55, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold1: eta=0.55, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold1: eta=0.55, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold1: eta=1.00, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold1: eta=1.00, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold1: eta=1.00, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold1: eta=1.00, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold1: eta=1.00, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold1: eta=1.00, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold2: eta=0.10, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold2: eta=0.10, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold2: eta=0.10, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold2: eta=0.10, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold2: eta=0.10, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold2: eta=0.10, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold2: eta=0.55, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold2: eta=0.55, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold2: eta=0.55, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold2: eta=0.55, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold2: eta=0.55, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold2: eta=0.55, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold2: eta=1.00, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold2: eta=1.00, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold2: eta=1.00, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold2: eta=1.00, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold2: eta=1.00, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold2: eta=1.00, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold3: eta=0.10, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold3: eta=0.10, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold3: eta=0.10, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold3: eta=0.10, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold3: eta=0.10, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold3: eta=0.10, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold3: eta=0.55, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold3: eta=0.55, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold3: eta=0.55, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold3: eta=0.55, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold3: eta=0.55, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold3: eta=0.55, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold3: eta=1.00, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold3: eta=1.00, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold3: eta=1.00, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold3: eta=1.00, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold3: eta=1.00, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold3: eta=1.00, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold4: eta=0.10, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold4: eta=0.10, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold4: eta=0.10, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold4: eta=0.10, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold4: eta=0.10, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold4: eta=0.10, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold4: eta=0.55, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold4: eta=0.55, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold4: eta=0.55, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold4: eta=0.55, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold4: eta=0.55, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold4: eta=0.55, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold4: eta=1.00, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold4: eta=1.00, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold4: eta=1.00, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold4: eta=1.00, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold4: eta=1.00, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold4: eta=1.00, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold5: eta=0.10, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold5: eta=0.10, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold5: eta=0.10, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold5: eta=0.10, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold5: eta=0.10, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold5: eta=0.10, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold5: eta=0.55, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold5: eta=0.55, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold5: eta=0.55, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold5: eta=0.55, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold5: eta=0.55, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold5: eta=0.55, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold5: eta=1.00, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold5: eta=1.00, max\_depth=17, gamma=0.10, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold5: eta=1.00, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold5: eta=1.00, max\_depth=17, gamma=0.55, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## + Fold5: eta=1.00, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## - Fold5: eta=1.00, max\_depth=17, gamma=1.00, colsample\_bytree=1, min\_child\_weight=1, subsample=0.5, nrounds=100   
## Aggregating results  
## Selecting tuning parameters  
## Fitting nrounds = 40, max\_depth = 17, eta = 0.1, gamma = 1, colsample\_bytree = 1, min\_child\_weight = 1, subsample = 0.5 on full training set

# Prediction with test  
y\_pred.xgb = predict(classifier.xgb, newdata = testing\_set[,-2])  
  
# Confusion Matrix  
cm.xgb <- table(testing\_set[,2], y\_pred.xgb)  
accuracy.xgb <- sum(diag(cm.xgb))/nrow(testing\_set)

plot(classifier.xgb)



cm.xgb

## y\_pred.xgb  
## 0 1 2 3  
## 0 871 37 0 0  
## 1 244 1332 49 10  
## 2 15 352 36 19  
## 3 0 248 29 32

accuracy.xgb

## [1] 0.6936469

Clearly I see as if the model has a slow learning rate, the results are better, then as more rounds does the model worse is the accuracy (overfitting) and finally a high gamma seems improve a little the accuracy for that combinations of shrinkage and number of rounds. The model predict pretty good the 0 and 1 but that is because ‘crowns’ is high correlated with ’vicder’and have more problems with the 2 and 3 crowns.

Finally I set up and algorithm to run 5 different models with a random search of the hyperparamenters to see if there is the chance to improve the accuracy.

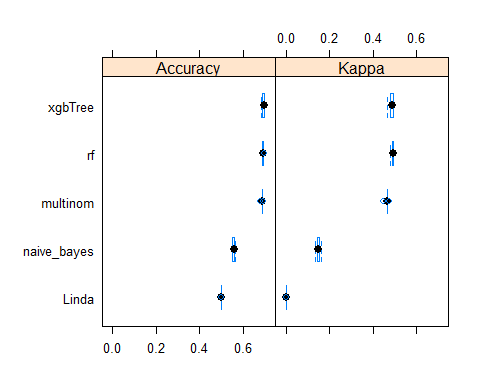
* XgbTree: again, just in case with random search find a better grid. (Extrem Gradient Boosting)
* rf: another ensamble model to face the problem. Maybe the random selection of predictors achieve better results than XgbTree. (RandomForest)
* Linda: maybe there is some kind of clustering here that I could miss, this model can help. (Linear Discriminant Analysis)
* naive\_bayes: Perhaps facing this problem by conditional probabilities improve results. (Naives\_bayes)
* multinom: afterall maybe a simple logistic regression is the best model.

Which will win?

set.seed(101)  
  
models <- c("xgbTree", "rf", "Linda", "naive\_bayes", "multinom")  
  
control <- trainControl(method = "cv",  
 number = 5,  
 search = "random",  
 verboseIter = T,  
 allowParallel = F)  
results <- list()  
predicts <- list()  
cms <- list()  
accuracies <- list()  
for (i in models){  
 classifier <- train(data = training\_set,  
 crowns~.,  
 method = i,  
 trControl = control,  
 tuneLength = 5,  
 metric = "Accuracy",  
 ntree = 100)  
   
 results[[i]] <- classifier  
 y\_pred <- predict(classifier, newdata = testing\_set[,-2], type = "raw")  
 predicts[[i]] <- y\_pred  
 cm <- table(testing\_set[,2], y\_pred)  
 cms[[i]] <- cm  
 accuracy <- sum(diag(cm))/nrow(testing\_set)  
 accuracies[[i]] <- accuracy  
}

## + Fold1: eta=0.02413, max\_depth=6, gamma=9.2332, colsample\_bytree=0.5637, min\_child\_weight=15, subsample=0.4250, nrounds=430   
## - Fold1: eta=0.02413, max\_depth=6, gamma=9.2332, colsample\_bytree=0.5637, min\_child\_weight=15, subsample=0.4250, nrounds=430   
## + Fold1: eta=0.12880, max\_depth=9, gamma=3.8941, colsample\_bytree=0.3791, min\_child\_weight=10, subsample=0.8288, nrounds=442   
## - Fold1: eta=0.12880, max\_depth=9, gamma=3.8941, colsample\_bytree=0.3791, min\_child\_weight=10, subsample=0.8288, nrounds=442   
## + Fold1: eta=0.39698, max\_depth=3, gamma=4.0645, colsample\_bytree=0.3653, min\_child\_weight=20, subsample=0.3307, nrounds=351   
## - Fold1: eta=0.39698, max\_depth=3, gamma=4.0645, colsample\_bytree=0.3653, min\_child\_weight=20, subsample=0.3307, nrounds=351   
## + Fold1: eta=0.42073, max\_depth=3, gamma=7.9572, colsample\_bytree=0.4693, min\_child\_weight=20, subsample=0.3037, nrounds= 95   
## - Fold1: eta=0.42073, max\_depth=3, gamma=7.9572, colsample\_bytree=0.4693, min\_child\_weight=20, subsample=0.3037, nrounds= 95   
## + Fold1: eta=0.57415, max\_depth=3, gamma=0.7121, colsample\_bytree=0.4284, min\_child\_weight= 9, subsample=0.9350, nrounds=209   
## - Fold1: eta=0.57415, max\_depth=3, gamma=0.7121, colsample\_bytree=0.4284, min\_child\_weight= 9, subsample=0.9350, nrounds=209   
## + Fold2: eta=0.02413, max\_depth=6, gamma=9.2332, colsample\_bytree=0.5637, min\_child\_weight=15, subsample=0.4250, nrounds=430   
## - Fold2: eta=0.02413, max\_depth=6, gamma=9.2332, colsample\_bytree=0.5637, min\_child\_weight=15, subsample=0.4250, nrounds=430   
## + Fold2: eta=0.12880, max\_depth=9, gamma=3.8941, colsample\_bytree=0.3791, min\_child\_weight=10, subsample=0.8288, nrounds=442   
## - Fold2: eta=0.12880, max\_depth=9, gamma=3.8941, colsample\_bytree=0.3791, min\_child\_weight=10, subsample=0.8288, nrounds=442   
## + Fold2: eta=0.39698, max\_depth=3, gamma=4.0645, colsample\_bytree=0.3653, min\_child\_weight=20, subsample=0.3307, nrounds=351   
## - Fold2: eta=0.39698, max\_depth=3, gamma=4.0645, colsample\_bytree=0.3653, min\_child\_weight=20, subsample=0.3307, nrounds=351   
## + Fold2: eta=0.42073, max\_depth=3, gamma=7.9572, colsample\_bytree=0.4693, min\_child\_weight=20, subsample=0.3037, nrounds= 95   
## - Fold2: eta=0.42073, max\_depth=3, gamma=7.9572, colsample\_bytree=0.4693, min\_child\_weight=20, subsample=0.3037, nrounds= 95   
## + Fold2: eta=0.57415, max\_depth=3, gamma=0.7121, colsample\_bytree=0.4284, min\_child\_weight= 9, subsample=0.9350, nrounds=209   
## - Fold2: eta=0.57415, max\_depth=3, gamma=0.7121, colsample\_bytree=0.4284, min\_child\_weight= 9, subsample=0.9350, nrounds=209   
## + Fold3: eta=0.02413, max\_depth=6, gamma=9.2332, colsample\_bytree=0.5637, min\_child\_weight=15, subsample=0.4250, nrounds=430   
## - Fold3: eta=0.02413, max\_depth=6, gamma=9.2332, colsample\_bytree=0.5637, min\_child\_weight=15, subsample=0.4250, nrounds=430   
## + Fold3: eta=0.12880, max\_depth=9, gamma=3.8941, colsample\_bytree=0.3791, min\_child\_weight=10, subsample=0.8288, nrounds=442   
## - Fold3: eta=0.12880, max\_depth=9, gamma=3.8941, colsample\_bytree=0.3791, min\_child\_weight=10, subsample=0.8288, nrounds=442   
## + Fold3: eta=0.39698, max\_depth=3, gamma=4.0645, colsample\_bytree=0.3653, min\_child\_weight=20, subsample=0.3307, nrounds=351   
## - Fold3: eta=0.39698, max\_depth=3, gamma=4.0645, colsample\_bytree=0.3653, min\_child\_weight=20, subsample=0.3307, nrounds=351   
## + Fold3: eta=0.42073, max\_depth=3, gamma=7.9572, colsample\_bytree=0.4693, min\_child\_weight=20, subsample=0.3037, nrounds= 95   
## - Fold3: eta=0.42073, max\_depth=3, gamma=7.9572, colsample\_bytree=0.4693, min\_child\_weight=20, subsample=0.3037, nrounds= 95   
## + Fold3: eta=0.57415, max\_depth=3, gamma=0.7121, colsample\_bytree=0.4284, min\_child\_weight= 9, subsample=0.9350, nrounds=209   
## - Fold3: eta=0.57415, max\_depth=3, gamma=0.7121, colsample\_bytree=0.4284, min\_child\_weight= 9, subsample=0.9350, nrounds=209   
## + Fold4: eta=0.02413, max\_depth=6, gamma=9.2332, colsample\_bytree=0.5637, min\_child\_weight=15, subsample=0.4250, nrounds=430   
## - Fold4: eta=0.02413, max\_depth=6, gamma=9.2332, colsample\_bytree=0.5637, min\_child\_weight=15, subsample=0.4250, nrounds=430   
## + Fold4: eta=0.12880, max\_depth=9, gamma=3.8941, colsample\_bytree=0.3791, min\_child\_weight=10, subsample=0.8288, nrounds=442   
## - Fold4: eta=0.12880, max\_depth=9, gamma=3.8941, colsample\_bytree=0.3791, min\_child\_weight=10, subsample=0.8288, nrounds=442   
## + Fold4: eta=0.39698, max\_depth=3, gamma=4.0645, colsample\_bytree=0.3653, min\_child\_weight=20, subsample=0.3307, nrounds=351   
## - Fold4: eta=0.39698, max\_depth=3, gamma=4.0645, colsample\_bytree=0.3653, min\_child\_weight=20, subsample=0.3307, nrounds=351   
## + Fold4: eta=0.42073, max\_depth=3, gamma=7.9572, colsample\_bytree=0.4693, min\_child\_weight=20, subsample=0.3037, nrounds= 95   
## - Fold4: eta=0.42073, max\_depth=3, gamma=7.9572, colsample\_bytree=0.4693, min\_child\_weight=20, subsample=0.3037, nrounds= 95   
## + Fold4: eta=0.57415, max\_depth=3, gamma=0.7121, colsample\_bytree=0.4284, min\_child\_weight= 9, subsample=0.9350, nrounds=209   
## - Fold4: eta=0.57415, max\_depth=3, gamma=0.7121, colsample\_bytree=0.4284, min\_child\_weight= 9, subsample=0.9350, nrounds=209   
## + Fold5: eta=0.02413, max\_depth=6, gamma=9.2332, colsample\_bytree=0.5637, min\_child\_weight=15, subsample=0.4250, nrounds=430   
## - Fold5: eta=0.02413, max\_depth=6, gamma=9.2332, colsample\_bytree=0.5637, min\_child\_weight=15, subsample=0.4250, nrounds=430   
## + Fold5: eta=0.12880, max\_depth=9, gamma=3.8941, colsample\_bytree=0.3791, min\_child\_weight=10, subsample=0.8288, nrounds=442   
## - Fold5: eta=0.12880, max\_depth=9, gamma=3.8941, colsample\_bytree=0.3791, min\_child\_weight=10, subsample=0.8288, nrounds=442   
## + Fold5: eta=0.39698, max\_depth=3, gamma=4.0645, colsample\_bytree=0.3653, min\_child\_weight=20, subsample=0.3307, nrounds=351   
## - Fold5: eta=0.39698, max\_depth=3, gamma=4.0645, colsample\_bytree=0.3653, min\_child\_weight=20, subsample=0.3307, nrounds=351   
## + Fold5: eta=0.42073, max\_depth=3, gamma=7.9572, colsample\_bytree=0.4693, min\_child\_weight=20, subsample=0.3037, nrounds= 95   
## - Fold5: eta=0.42073, max\_depth=3, gamma=7.9572, colsample\_bytree=0.4693, min\_child\_weight=20, subsample=0.3037, nrounds= 95   
## + Fold5: eta=0.57415, max\_depth=3, gamma=0.7121, colsample\_bytree=0.4284, min\_child\_weight= 9, subsample=0.9350, nrounds=209   
## - Fold5: eta=0.57415, max\_depth=3, gamma=0.7121, colsample\_bytree=0.4284, min\_child\_weight= 9, subsample=0.9350, nrounds=209   
## Aggregating results  
## Selecting tuning parameters  
## Fitting nrounds = 442, max\_depth = 9, eta = 0.129, gamma = 3.89, colsample\_bytree = 0.379, min\_child\_weight = 10, subsample = 0.829 on full training set  
## + Fold1: mtry= 1   
## - Fold1: mtry= 1   
## + Fold1: mtry=12   
## - Fold1: mtry=12   
## + Fold1: mtry=17   
## - Fold1: mtry=17   
## + Fold1: mtry=10   
## - Fold1: mtry=10   
## + Fold1: mtry= 6   
## - Fold1: mtry= 6   
## + Fold2: mtry= 1   
## - Fold2: mtry= 1   
## + Fold2: mtry=12   
## - Fold2: mtry=12   
## + Fold2: mtry=17   
## - Fold2: mtry=17   
## + Fold2: mtry=10   
## - Fold2: mtry=10   
## + Fold2: mtry= 6   
## - Fold2: mtry= 6   
## + Fold3: mtry= 1   
## - Fold3: mtry= 1   
## + Fold3: mtry=12   
## - Fold3: mtry=12   
## + Fold3: mtry=17   
## - Fold3: mtry=17   
## + Fold3: mtry=10   
## - Fold3: mtry=10   
## + Fold3: mtry= 6   
## - Fold3: mtry= 6   
## + Fold4: mtry= 1   
## - Fold4: mtry= 1   
## + Fold4: mtry=12   
## - Fold4: mtry=12   
## + Fold4: mtry=17   
## - Fold4: mtry=17   
## + Fold4: mtry=10   
## - Fold4: mtry=10   
## + Fold4: mtry= 6   
## - Fold4: mtry= 6   
## + Fold5: mtry= 1   
## - Fold5: mtry= 1   
## + Fold5: mtry=12   
## - Fold5: mtry=12   
## + Fold5: mtry=17   
## - Fold5: mtry=17   
## + Fold5: mtry=10   
## - Fold5: mtry=10   
## + Fold5: mtry= 6   
## - Fold5: mtry= 6   
## Aggregating results  
## Selecting tuning parameters  
## Fitting mtry = 12 on full training set  
## + Fold1: parameter=none   
## - Fold1: parameter=none   
## + Fold2: parameter=none   
## - Fold2: parameter=none   
## + Fold3: parameter=none   
## - Fold3: parameter=none   
## + Fold4: parameter=none   
## - Fold4: parameter=none   
## + Fold5: parameter=none   
## - Fold5: parameter=none   
## Aggregating results  
## Fitting final model on full training set  
## + Fold1: usekernel= TRUE, laplace=0, adjust=1   
## - Fold1: usekernel= TRUE, laplace=0, adjust=1   
## + Fold1: usekernel=FALSE, laplace=0, adjust=1   
## - Fold1: usekernel=FALSE, laplace=0, adjust=1   
## + Fold2: usekernel= TRUE, laplace=0, adjust=1   
## - Fold2: usekernel= TRUE, laplace=0, adjust=1   
## + Fold2: usekernel=FALSE, laplace=0, adjust=1   
## - Fold2: usekernel=FALSE, laplace=0, adjust=1   
## + Fold3: usekernel= TRUE, laplace=0, adjust=1   
## - Fold3: usekernel= TRUE, laplace=0, adjust=1   
## + Fold3: usekernel=FALSE, laplace=0, adjust=1   
## - Fold3: usekernel=FALSE, laplace=0, adjust=1   
## + Fold4: usekernel= TRUE, laplace=0, adjust=1   
## - Fold4: usekernel= TRUE, laplace=0, adjust=1   
## + Fold4: usekernel=FALSE, laplace=0, adjust=1   
## - Fold4: usekernel=FALSE, laplace=0, adjust=1   
## + Fold5: usekernel= TRUE, laplace=0, adjust=1   
## - Fold5: usekernel= TRUE, laplace=0, adjust=1   
## + Fold5: usekernel=FALSE, laplace=0, adjust=1   
## - Fold5: usekernel=FALSE, laplace=0, adjust=1   
## Aggregating results  
## Selecting tuning parameters  
## Fitting laplace = 0, usekernel = TRUE, adjust = 1 on full training set  
## + Fold1: decay=0.1183   
## # weights: 76 (54 variable)  
## initial value 10894.887384   
## iter 10 value 9945.098285  
## iter 20 value 8842.432493  
## iter 30 value 7634.269336  
## iter 40 value 7035.905401  
## iter 50 value 6637.084024  
## iter 60 value 6048.203245  
## iter 70 value 6006.516499  
## final value 6006.396678   
## converged  
## - Fold1: decay=0.1183   
## + Fold1: decay=0.6080   
## # weights: 76 (54 variable)  
## initial value 10894.887384   
## iter 10 value 9945.098869  
## iter 20 value 8848.337157  
## iter 30 value 7665.406900  
## iter 40 value 7091.273737  
## iter 50 value 6776.334880  
## iter 60 value 6139.953553  
## iter 70 value 6122.745827  
## final value 6122.742585   
## converged  
## - Fold1: decay=0.6080   
## + Fold1: decay=6.8126   
## # weights: 76 (54 variable)  
## initial value 10894.887384   
## iter 10 value 9945.106280  
## iter 20 value 8919.246439  
## iter 30 value 8003.368855  
## iter 40 value 7581.449218  
## iter 50 value 7373.357109  
## iter 60 value 6723.754510  
## final value 6718.911663   
## converged  
## - Fold1: decay=6.8126   
## + Fold1: decay=9.8652   
## # weights: 76 (54 variable)  
## initial value 10894.887384   
## iter 10 value 9945.109926  
## iter 20 value 8951.665253  
## iter 30 value 8138.917698  
## iter 40 value 7795.486437  
## iter 50 value 7521.606739  
## iter 60 value 6892.991089  
## iter 70 value 6887.010367  
## iter 70 value 6887.010349  
## iter 70 value 6887.010348  
## final value 6887.010348   
## converged  
## - Fold1: decay=9.8652   
## + Fold1: decay=1.0141   
## # weights: 76 (54 variable)  
## initial value 10894.887384   
## iter 10 value 9945.099355  
## iter 20 value 8853.198360  
## iter 30 value 7690.662023  
## iter 40 value 7128.259394  
## iter 50 value 6822.738958  
## iter 60 value 6218.510954  
## iter 70 value 6192.685941  
## final value 6192.684192   
## converged  
## - Fold1: decay=1.0141   
## + Fold2: decay=0.1183   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9963.070168  
## iter 20 value 8804.329217  
## iter 30 value 7939.615634  
## iter 40 value 7673.458706  
## iter 50 value 7503.634731  
## iter 60 value 6072.437576  
## iter 70 value 6024.571525  
## final value 6024.343621   
## converged  
## - Fold2: decay=0.1183   
## + Fold2: decay=0.6080   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9963.070839  
## iter 20 value 8809.508086  
## iter 30 value 7638.180868  
## iter 40 value 7392.952923  
## iter 50 value 6784.469045  
## iter 60 value 6167.680700  
## iter 70 value 6138.799304  
## final value 6138.797393   
## converged  
## - Fold2: decay=0.6080   
## + Fold2: decay=6.8126   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9963.079338  
## iter 20 value 8872.038761  
## iter 30 value 7892.864532  
## iter 40 value 7715.772383  
## iter 50 value 7336.186177  
## iter 60 value 6738.313605  
## final value 6731.425017   
## converged  
## - Fold2: decay=6.8126   
## + Fold2: decay=9.8652   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9963.083519  
## iter 20 value 8900.850431  
## iter 30 value 7988.346967  
## iter 40 value 7867.419402  
## iter 50 value 7303.840122  
## iter 60 value 6900.573900  
## final value 6898.921363   
## converged  
## - Fold2: decay=9.8652   
## + Fold2: decay=1.0141   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9963.071395  
## iter 20 value 8813.774855  
## iter 30 value 7656.883269  
## iter 40 value 7416.811279  
## iter 50 value 6836.150477  
## iter 60 value 6235.616813  
## iter 70 value 6207.957628  
## final value 6207.957344   
## converged  
## - Fold2: decay=1.0141   
## + Fold3: decay=0.1183   
## # weights: 76 (54 variable)  
## initial value 10894.887384   
## iter 10 value 9945.224109  
## iter 20 value 8934.205677  
## iter 30 value 7720.177688  
## iter 40 value 7306.249277  
## iter 50 value 7123.002656  
## iter 60 value 6020.291911  
## iter 70 value 5991.216679  
## final value 5991.085336   
## converged  
## - Fold3: decay=0.1183   
## + Fold3: decay=0.6080   
## # weights: 76 (54 variable)  
## initial value 10894.887384   
## iter 10 value 9945.224755  
## iter 20 value 8941.038018  
## iter 30 value 7753.958043  
## iter 40 value 7349.205309  
## iter 50 value 7160.318891  
## iter 60 value 6143.255981  
## iter 70 value 6104.801209  
## iter 70 value 6104.801168  
## iter 70 value 6104.801168  
## final value 6104.801168   
## converged  
## - Fold3: decay=0.6080   
## + Fold3: decay=6.8126   
## # weights: 76 (54 variable)  
## initial value 10894.887384   
## iter 10 value 9945.232933  
## iter 20 value 9022.563179  
## iter 30 value 8111.205874  
## iter 40 value 7827.091295  
## iter 50 value 7450.373062  
## iter 60 value 6709.273348  
## final value 6700.518724   
## converged  
## - Fold3: decay=6.8126   
## + Fold3: decay=9.8652   
## # weights: 76 (54 variable)  
## initial value 10894.887384   
## iter 10 value 9945.236955  
## iter 20 value 9059.490645  
## iter 30 value 8250.281250  
## iter 40 value 7985.313373  
## iter 50 value 7480.155197  
## iter 60 value 6876.000160  
## final value 6869.494045   
## converged  
## - Fold3: decay=9.8652   
## + Fold3: decay=1.0141   
## # weights: 76 (54 variable)  
## initial value 10894.887384   
## iter 10 value 9945.225290  
## iter 20 value 8946.658121  
## iter 30 value 7781.240513  
## iter 40 value 7384.131564  
## iter 50 value 7197.387469  
## iter 60 value 6207.481540  
## iter 70 value 6173.881999  
## final value 6173.880242   
## converged  
## - Fold3: decay=1.0141   
## + Fold4: decay=0.1183   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9934.740565  
## iter 20 value 9037.231686  
## iter 30 value 8039.736783  
## iter 40 value 7895.097550  
## iter 50 value 7717.006757  
## iter 60 value 6070.948655  
## iter 70 value 6009.410741  
## final value 6009.181430   
## converged  
## - Fold4: decay=0.1183   
## + Fold4: decay=0.6080   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9934.741254  
## iter 20 value 9044.548198  
## iter 30 value 8119.682473  
## iter 40 value 7983.628228  
## iter 50 value 7790.500548  
## iter 60 value 6145.519532  
## iter 70 value 6122.815675  
## final value 6122.810634   
## converged  
## - Fold4: decay=0.6080   
## + Fold4: decay=6.8126   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9934.749990  
## iter 20 value 9131.823541  
## iter 30 value 8109.398961  
## iter 40 value 7954.212148  
## iter 50 value 7188.472409  
## iter 60 value 6724.326993  
## final value 6712.316104   
## converged  
## - Fold4: decay=6.8126   
## + Fold4: decay=9.8652   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9934.754287  
## iter 20 value 9171.377742  
## iter 30 value 8170.492302  
## iter 40 value 8100.870096  
## iter 50 value 7657.560935  
## iter 60 value 6898.004176  
## final value 6879.490823   
## converged  
## - Fold4: decay=9.8652   
## + Fold4: decay=1.0141   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9934.741826  
## iter 20 value 9050.565949  
## iter 30 value 8189.705825  
## iter 40 value 8060.588221  
## iter 50 value 7832.845027  
## iter 60 value 6241.914721  
## iter 70 value 6191.466662  
## final value 6191.453899   
## converged  
## - Fold4: decay=1.0141   
## + Fold5: decay=0.1183   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9949.587431  
## iter 20 value 8839.872532  
## iter 30 value 7646.687147  
## iter 40 value 7007.749938  
## iter 50 value 6827.511391  
## iter 60 value 6053.726083  
## iter 70 value 6024.782790  
## final value 6024.563081   
## converged  
## - Fold5: decay=0.1183   
## + Fold5: decay=0.6080   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9949.588080  
## iter 20 value 8845.413363  
## iter 30 value 7676.594920  
## iter 40 value 7050.981208  
## iter 50 value 6885.556333  
## iter 60 value 6160.710729  
## iter 70 value 6138.018488  
## final value 6138.017177   
## converged  
## - Fold5: decay=0.6080   
## + Fold5: decay=6.8126   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9949.596302  
## iter 20 value 8911.877708  
## iter 30 value 7996.993028  
## iter 40 value 7585.133782  
## iter 50 value 7384.679925  
## iter 60 value 6748.403368  
## final value 6729.073233   
## converged  
## - Fold5: decay=6.8126   
## + Fold5: decay=9.8652   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9949.600346  
## iter 20 value 8942.211813  
## iter 30 value 8123.897159  
## iter 40 value 7803.684384  
## iter 50 value 7610.913715  
## iter 60 value 6918.918535  
## final value 6896.538985   
## converged  
## - Fold5: decay=9.8652   
## + Fold5: decay=1.0141   
## # weights: 76 (54 variable)  
## initial value 10893.501090   
## iter 10 value 9949.588618  
## iter 20 value 8849.974359  
## iter 30 value 7700.811790  
## iter 40 value 7086.339011  
## iter 50 value 6943.936378  
## iter 60 value 6213.670115  
## iter 70 value 6206.738699  
## final value 6206.738527   
## converged  
## - Fold5: decay=1.0141   
## Aggregating results  
## Selecting tuning parameters  
## Fitting decay = 0.118 on full training set  
## # weights: 76 (54 variable)  
## initial value 13617.569509   
## iter 10 value 12218.893995  
## iter 20 value 10290.580305  
## iter 30 value 9492.963699  
## iter 40 value 9423.898906  
## iter 50 value 9174.932404  
## iter 60 value 7550.906126  
## iter 70 value 7510.436492  
## iter 80 value 7510.210716  
## iter 80 value 7510.210643  
## iter 80 value 7510.210643  
## final value 7510.210643   
## converged

bwplot(resamples(results))



cms

## $xgbTree  
## y\_pred  
## 0 1 2 3  
## 0 868 40 0 0  
## 1 241 1342 35 17  
## 2 15 338 50 19  
## 3 0 246 33 30  
##   
## $rf  
## y\_pred  
## 0 1 2 3  
## 0 874 34 0 0  
## 1 253 1328 37 17  
## 2 15 334 49 24  
## 3 0 234 37 38  
##   
## $Linda  
## y\_pred  
## 0 1 2 3  
## 0 0 908 0 0  
## 1 0 1635 0 0  
## 2 0 422 0 0  
## 3 0 309 0 0  
##   
## $naive\_bayes  
## y\_pred  
## 0 1 2 3  
## 0 243 665 0 0  
## 1 50 1584 1 0  
## 2 1 418 3 0  
## 3 0 306 3 0  
##   
## $multinom  
## y\_pred  
## 0 1 2 3  
## 0 908 0 0 0  
## 1 272 1359 4 0  
## 2 19 395 7 1  
## 3 0 304 5 0

accuracies

## $xgbTree  
## [1] 0.6994502  
##   
## $rf  
## [1] 0.6991448  
##   
## $Linda  
## [1] 0.4993891  
##   
## $naive\_bayes  
## [1] 0.5589493  
##   
## $multinom  
## [1] 0.6945632

The plot shows XgbTree as the best model to predict crowns with a 0.6994502% of accuracy and Linda the worst with 0.4993891. It’s relevant say that the logistic regression did very good job, let the accuracy at the level of the Extrem Gradient Boosting. If analyse the confusion matrices, I see as multinom is not able to classify 2 and 3 crowns. So if i haveto choose one of those models I will go for the first or the second. Maybe the second as it has less parameters to tune and it has been faster.