Haciendo Big Data a League of Legends

Jorge de Andrés

16 de enero de 2019

Limpiamos environment:

rm(list=ls())

Importamos los paquetes necesarios:

#install.packages("knitr")  
#install.packages("dplyr")  
#install.packages("pryr")  
#install.packages("corrplot")  
#install.packages("rjson")  
#install.packages("plyr")  
#install.packages("wordcloud")  
#install.packages("ggplot2")  
#install.packages("hexbin")  
#install.packages("RColorBrewer")  
#install.packages("FactoMineR")  
#devtools::install\_github("kassambara/factoextra")  
#install.packages("factoextra")  
#install.packages("arules")  
#install.packages("arulesViz")  
#install.packages("fastDummies")  
#install.packages("caret")  
#install.packages("nnet")  
#install.packages("gmodels")  
#install.packages("class")  
#install.packages("randomforest")  
#install.packages("e1071")  
#install.packages("aplpack")  
#install.packages("cluster")  
#install.packages("fpc")  
#install.packages("purrr")  
#install.packages("RWeka")  
  
library("corrplot")

## corrplot 0.84 loaded

library("dplyr")

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library("pryr")  
library("rjson")  
library("plyr")

## -------------------------------------------------------------------------

## You have loaded plyr after dplyr - this is likely to cause problems.  
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:  
## library(plyr); library(dplyr)

## -------------------------------------------------------------------------

##   
## Attaching package: 'plyr'

## The following objects are masked from 'package:dplyr':  
##   
## arrange, count, desc, failwith, id, mutate, rename, summarise,  
## summarize

library("wordcloud")

## Loading required package: RColorBrewer

library("ggplot2")  
library("hexbin")  
library("RColorBrewer")  
library("FactoMineR")  
library("factoextra")

## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at https://goo.gl/13EFCZ

require("arulesViz")

## Loading required package: arulesViz

## Loading required package: arules

## Loading required package: Matrix

##   
## Attaching package: 'arules'

## The following object is masked from 'package:pryr':  
##   
## inspect

## The following object is masked from 'package:dplyr':  
##   
## recode

## The following objects are masked from 'package:base':  
##   
## abbreviate, write

## Loading required package: grid

require("arules")  
library("Matrix")  
library("fastDummies")  
library("caret")

## Loading required package: lattice

library("nnet")  
library("gmodels")  
library("class")  
library("randomForest")

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':  
##   
## margin

## The following object is masked from 'package:dplyr':  
##   
## combine

library("e1071")  
library("aplpack")

## Loading required package: tcltk

library("cluster")  
library("fpc")  
library("purrr")

##   
## Attaching package: 'purrr'

## The following object is masked from 'package:caret':  
##   
## lift

## The following object is masked from 'package:plyr':  
##   
## compact

## The following objects are masked from 'package:pryr':  
##   
## compose, partial

library("RWeka")  
library("data.table")

##   
## Attaching package: 'data.table'

## The following object is masked from 'package:purrr':  
##   
## transpose

## The following object is masked from 'package:pryr':  
##   
## address

## The following objects are masked from 'package:dplyr':  
##   
## between, first, last

Lo primero que voy a hacer es importar todos los datasets:

data.games <- read.csv("../../data/games.csv")  
data.champs <- read.csv("../../data/champs.csv")

Hemos importado varios CSV que harán las tablas de la base de datos.

Importamos los json:

lista.championInfo <- fromJSON(file = "../../data/champion\_info.json")  
lista.championInfo2 <- fromJSON(file="../../data/champion\_info\_2.json")  
lista.summonerSpell <- fromJSON(file="../../data/summoner\_spell\_info.json")

Tamaño de mis datasets…

print( paste0("El tamaño de la tabla de games es de: "))

## [1] "El tamaño de la tabla de games es de: "

object\_size(data.games)

## 13 MB

Esto me devuelve las filas que tengan NA:

data.games[!complete.cases(data.games),]

## [1] gameId creationTime gameDuration   
## [4] seasonId winner firstBlood   
## [7] firstTower firstInhibitor firstBaron   
## [10] firstDragon firstRiftHerald t1\_champ1id   
## [13] t1\_champ1\_sum1 t1\_champ1\_sum2 t1\_champ2id   
## [16] t1\_champ2\_sum1 t1\_champ2\_sum2 t1\_champ3id   
## [19] t1\_champ3\_sum1 t1\_champ3\_sum2 t1\_champ4id   
## [22] t1\_champ4\_sum1 t1\_champ4\_sum2 t1\_champ5id   
## [25] t1\_champ5\_sum1 t1\_champ5\_sum2 t1\_towerKills   
## [28] t1\_inhibitorKills t1\_baronKills t1\_dragonKills   
## [31] t1\_riftHeraldKills t1\_ban1 t1\_ban2   
## [34] t1\_ban3 t1\_ban4 t1\_ban5   
## [37] t2\_champ1id t2\_champ1\_sum1 t2\_champ1\_sum2   
## [40] t2\_champ2id t2\_champ2\_sum1 t2\_champ2\_sum2   
## [43] t2\_champ3id t2\_champ3\_sum1 t2\_champ3\_sum2   
## [46] t2\_champ4id t2\_champ4\_sum1 t2\_champ4\_sum2   
## [49] t2\_champ5id t2\_champ5\_sum1 t2\_champ5\_sum2   
## [52] t2\_towerKills t2\_inhibitorKills t2\_baronKills   
## [55] t2\_dragonKills t2\_riftHeraldKills t2\_ban1   
## [58] t2\_ban2 t2\_ban3 t2\_ban4   
## [61] t2\_ban5   
## <0 rows> (or 0-length row.names)

Como podemos ver, no tenemos filas con NA.

Como voy a ir haciendo “pequeños proyectos” para ir sacando conocimiento por ahora no voy a fusionar ninguna tabla de primeras porque no se lo que voy a necesitar. Según se vayan planteando los interrogantes iré jugando con las tablas para conseguir los objetivos

# Análisis Exploratorio

Empezamos por el análisis exploratorio de los datos. Para ello, vamos intentar ver conclusiones sobre los mismos:

print("Resumen de data.games")

## [1] "Resumen de data.games"

summary(data.games) # Las únicas columnas interesantes son la 5 y la 6

## gameId creationTime gameDuration seasonId  
## Min. :3.215e+09 Min. :1.497e+12 Min. : 190 Min. :9   
## 1st Qu.:3.292e+09 1st Qu.:1.502e+12 1st Qu.:1531 1st Qu.:9   
## Median :3.320e+09 Median :1.504e+12 Median :1833 Median :9   
## Mean :3.306e+09 Mean :1.503e+12 Mean :1832 Mean :9   
## 3rd Qu.:3.327e+09 3rd Qu.:1.504e+12 3rd Qu.:2148 3rd Qu.:9   
## Max. :3.332e+09 Max. :1.505e+12 Max. :4728 Max. :9   
## winner firstBlood firstTower firstInhibitor   
## Min. :1.000 Min. :0.000 Min. :0.000 Min. :0.000   
## 1st Qu.:1.000 1st Qu.:1.000 1st Qu.:1.000 1st Qu.:1.000   
## Median :1.000 Median :1.000 Median :1.000 Median :1.000   
## Mean :1.494 Mean :1.471 Mean :1.451 Mean :1.308   
## 3rd Qu.:2.000 3rd Qu.:2.000 3rd Qu.:2.000 3rd Qu.:2.000   
## Max. :2.000 Max. :2.000 Max. :2.000 Max. :2.000   
## firstBaron firstDragon firstRiftHerald t1\_champ1id   
## Min. :0.0000 Min. :0.000 Min. :0.0000 Min. : 1.0   
## 1st Qu.:0.0000 1st Qu.:1.000 1st Qu.:0.0000 1st Qu.: 35.0   
## Median :1.0000 Median :1.000 Median :0.0000 Median : 79.0   
## Mean :0.9265 Mean :1.443 Mean :0.7317 Mean :114.3   
## 3rd Qu.:2.0000 3rd Qu.:2.000 3rd Qu.:1.0000 3rd Qu.:136.0   
## Max. :2.0000 Max. :2.000 Max. :2.0000 Max. :516.0   
## t1\_champ1\_sum1 t1\_champ1\_sum2 t1\_champ2id t1\_champ2\_sum1   
## Min. : 1.000 Min. : 1.000 Min. : 1.0 Min. : 1.000   
## 1st Qu.: 4.000 1st Qu.: 4.000 1st Qu.: 35.0 1st Qu.: 4.000   
## Median : 4.000 Median : 4.000 Median : 79.0 Median : 4.000   
## Mean : 6.602 Mean : 7.334 Mean :118.1 Mean : 6.548   
## 3rd Qu.:11.000 3rd Qu.:11.000 3rd Qu.:141.0 3rd Qu.:11.000   
## Max. :21.000 Max. :21.000 Max. :516.0 Max. :21.000   
## t1\_champ2\_sum2 t1\_champ3id t1\_champ3\_sum1 t1\_champ3\_sum2   
## Min. : 1.000 Min. : 1.0 Min. : 1.000 Min. : 1.000   
## 1st Qu.: 4.000 1st Qu.: 35.0 1st Qu.: 4.000 1st Qu.: 4.000   
## Median : 4.000 Median : 78.0 Median : 4.000 Median : 4.000   
## Mean : 7.198 Mean :116.9 Mean : 6.542 Mean : 7.201   
## 3rd Qu.:11.000 3rd Qu.:141.0 3rd Qu.:11.000 3rd Qu.:11.000   
## Max. :21.000 Max. :516.0 Max. :21.000 Max. :21.000   
## t1\_champ4id t1\_champ4\_sum1 t1\_champ4\_sum2 t1\_champ5id   
## Min. : 1.0 Min. : 1.000 Min. : 1.000 Min. : 1.0   
## 1st Qu.: 36.0 1st Qu.: 4.000 1st Qu.: 4.000 1st Qu.: 35.0   
## Median : 79.0 Median : 4.000 Median : 4.000 Median : 78.0   
## Mean :117.7 Mean : 6.531 Mean : 7.221 Mean :114.6   
## 3rd Qu.:141.0 3rd Qu.:11.000 3rd Qu.:11.000 3rd Qu.:136.0   
## Max. :516.0 Max. :21.000 Max. :21.000 Max. :516.0   
## t1\_champ5\_sum1 t1\_champ5\_sum2 t1\_towerKills t1\_inhibitorKills  
## Min. : 1.000 Min. : 1.000 Min. : 0.000 Min. : 0.000   
## 1st Qu.: 4.000 1st Qu.: 4.000 1st Qu.: 2.000 1st Qu.: 0.000   
## Median : 4.000 Median : 4.000 Median : 6.000 Median : 1.000   
## Mean : 6.622 Mean : 7.261 Mean : 5.699 Mean : 1.018   
## 3rd Qu.:11.000 3rd Qu.:11.000 3rd Qu.: 9.000 3rd Qu.: 2.000   
## Max. :21.000 Max. :21.000 Max. :11.000 Max. :10.000   
## t1\_baronKills t1\_dragonKills t1\_riftHeraldKills t1\_ban1   
## Min. :0.0000 Min. :0.000 Min. :0.0000 Min. : -1.0   
## 1st Qu.:0.0000 1st Qu.:0.000 1st Qu.:0.0000 1st Qu.: 38.0   
## Median :0.0000 Median :1.000 Median :0.0000 Median : 90.0   
## Mean :0.3723 Mean :1.387 Mean :0.2515 Mean :108.3   
## 3rd Qu.:1.0000 3rd Qu.:2.000 3rd Qu.:1.0000 3rd Qu.:141.0   
## Max. :5.0000 Max. :6.000 Max. :1.0000 Max. :516.0   
## t1\_ban2 t1\_ban3 t1\_ban4 t1\_ban5   
## Min. : -1.0 Min. : -1.0 Min. : -1.0 Min. : -1   
## 1st Qu.: 38.0 1st Qu.: 37.0 1st Qu.: 38.0 1st Qu.: 38   
## Median : 90.0 Median : 90.0 Median : 89.0 Median : 90   
## Mean :108.8 Mean :108.2 Mean :107.6 Mean :109   
## 3rd Qu.:141.0 3rd Qu.:141.0 3rd Qu.:141.0 3rd Qu.:141   
## Max. :516.0 Max. :516.0 Max. :516.0 Max. :516   
## t2\_champ1id t2\_champ1\_sum1 t2\_champ1\_sum2 t2\_champ2id   
## Min. : 1.0 Min. : 1.000 Min. : 1.000 Min. : 1.0   
## 1st Qu.: 35.0 1st Qu.: 4.000 1st Qu.: 4.000 1st Qu.: 35.0   
## Median : 78.0 Median : 4.000 Median : 4.000 Median : 79.0   
## Mean :115.9 Mean : 6.595 Mean : 7.305 Mean :117.6   
## 3rd Qu.:141.0 3rd Qu.:11.000 3rd Qu.:11.000 3rd Qu.:141.0   
## Max. :516.0 Max. :21.000 Max. :21.000 Max. :516.0   
## t2\_champ2\_sum1 t2\_champ2\_sum2 t2\_champ3id t2\_champ3\_sum1   
## Min. : 1.000 Min. : 1.000 Min. : 1.0 Min. : 1.000   
## 1st Qu.: 4.000 1st Qu.: 4.000 1st Qu.: 36.0 1st Qu.: 4.000   
## Median : 4.000 Median : 4.000 Median : 79.0 Median : 4.000   
## Mean : 6.547 Mean : 7.231 Mean :117.5 Mean : 6.522   
## 3rd Qu.:11.000 3rd Qu.:11.000 3rd Qu.:141.0 3rd Qu.:11.000   
## Max. :21.000 Max. :21.000 Max. :516.0 Max. :21.000   
## t2\_champ3\_sum2 t2\_champ4id t2\_champ4\_sum1 t2\_champ4\_sum2   
## Min. : 1.000 Min. : 1.0 Min. : 1.000 Min. : 1.000   
## 1st Qu.: 4.000 1st Qu.: 35.0 1st Qu.: 4.000 1st Qu.: 4.000   
## Median : 4.000 Median : 79.0 Median : 4.000 Median : 4.000   
## Mean : 7.227 Mean :118.2 Mean : 6.535 Mean : 7.201   
## 3rd Qu.:11.000 3rd Qu.:141.0 3rd Qu.:11.000 3rd Qu.:11.000   
## Max. :21.000 Max. :516.0 Max. :21.000 Max. :21.000   
## t2\_champ5id t2\_champ5\_sum1 t2\_champ5\_sum2 t2\_towerKills   
## Min. : 1.0 Min. : 1.000 Min. : 1.00 Min. : 0.000   
## 1st Qu.: 35.0 1st Qu.: 4.000 1st Qu.: 4.00 1st Qu.: 2.000   
## Median : 78.0 Median : 4.000 Median : 4.00 Median : 6.000   
## Mean :115.9 Mean : 6.613 Mean : 7.25 Mean : 5.549   
## 3rd Qu.:141.0 3rd Qu.:11.000 3rd Qu.:11.00 3rd Qu.: 9.000   
## Max. :516.0 Max. :21.000 Max. :21.00 Max. :11.000   
## t2\_inhibitorKills t2\_baronKills t2\_dragonKills t2\_riftHeraldKills  
## Min. : 0.0000 Min. :0.0000 Min. :0.000 Min. :0.0000   
## 1st Qu.: 0.0000 1st Qu.:0.0000 1st Qu.:0.000 1st Qu.:0.0000   
## Median : 0.0000 Median :0.0000 Median :1.000 Median :0.0000   
## Mean : 0.9851 Mean :0.4145 Mean :1.404 Mean :0.2401   
## 3rd Qu.: 2.0000 3rd Qu.:1.0000 3rd Qu.:2.000 3rd Qu.:0.0000   
## Max. :10.0000 Max. :4.0000 Max. :6.000 Max. :1.0000   
## t2\_ban1 t2\_ban2 t2\_ban3 t2\_ban4   
## Min. : -1.0 Min. : -1.0 Min. : -1.0 Min. : -1.0   
## 1st Qu.: 38.0 1st Qu.: 37.0 1st Qu.: 38.0 1st Qu.: 38.0   
## Median : 90.0 Median : 90.0 Median : 90.0 Median : 90.0   
## Mean :108.2 Mean :107.9 Mean :108.7 Mean :108.6   
## 3rd Qu.:141.0 3rd Qu.:141.0 3rd Qu.:141.0 3rd Qu.:141.0   
## Max. :516.0 Max. :516.0 Max. :516.0 Max. :516.0   
## t2\_ban5   
## Min. : -1.0   
## 1st Qu.: 38.0   
## Median : 90.0   
## Mean :108.1   
## 3rd Qu.:141.0   
## Max. :516.0

Una vez que tenemos los estadísticos básicos principales, vamos a limpiar algunos datasets de variables que no necesitamos para poder hacer análisis de variables con su correlación y similares…

# Ahora voy a hacer una tabla de games especial solo con las variables de las que calcularemos las correlaciones  
# Game ID es una variable que no necesito para nada, por lo que la voy a eliminar:  
  
data.games.corr <- data.games[, c(-1, -4)]  
#head(data.games.corr)  
data.games.corr <- data.games.corr[, -10:-24]  
#head(data.games.corr)  
data.games.corr <- data.games.corr[, -15:-34]  
#head(data.games.corr)  
data.games.corr <- data.games.corr[, -20:-24]

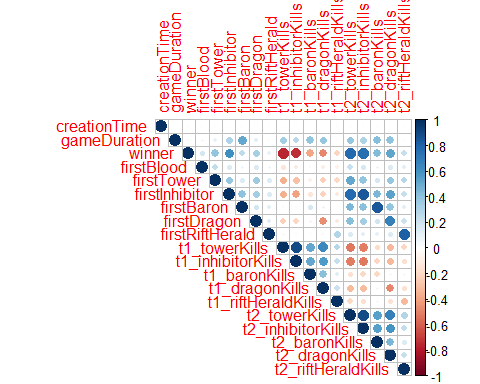
## Correlation Plots

Lo primero que voy a hacer son los correlation plot de cada una de las tablas. Vamos a ver qué relaciones tenemos entre los datos, para poder ver si podemos ir sacando algunas conclusiones…

res.games <- cor(data.games.corr, method = "spearman")  
options(width = 100)  
res.games.round <- round(res.games, 2)

El primer gráfico que muestro es entre la duración de las partidas y la temporada en la que se está jugando:

corrplot(res.games.round, method="circle", type = "upper")



Exportamos el corrplot a PDF para tenerlo vectorial y poderlo ver mejor…

pdf("../imagenes/corrplot.pdf")  
corrplot(res.games.round, method="circle", type = "upper")  
dev.off

## function (which = dev.cur())   
## {  
## if (which == 1)   
## stop("cannot shut down device 1 (the null device)")  
## .External(C\_devoff, as.integer(which))  
## dev.cur()  
## }  
## <bytecode: 0x00000000134d26f0>  
## <environment: namespace:grDevices>

Podemos apreciar las siguientes correlaciones sobre destruir torres:

* Destruir más torres tiene una relación directa con destruir inhibidores
* Destruir más torres tiene una cierta relación directa con matar dragones
* Destruir más torres tiene una cierta relación inversa con que el equipo contrario destruya tus torres
* Destruir más torres tiene uan cierta relación con que el equipo contrario mate menos dragones

Respecto a ganar, obtenemos las siguientes correlaciones:

* Muchas veces, quien destruye el primer inhibidor es el equipo que acaba ganando la partida
* Un aumento en que el equipo 1 destruya más torres tiene que ver con que tengan más posibilidades de victoria. Esto se da con una relación inversa, ya que la victoria del equipo 1 se marca con un 1, la del 2 con un dos, y el aumento de towerkills de t1 implica una bajada en win (1 en vez de dos)
* Lo mismi pasa con la cantidad de inhibidores destruidos, donde es también una relación muy fuerte la que hay.
* Con el equipo 2 pasa lo mismo, lo que pasa es que, por lo explicado anteriormente, en este caso se muestra como relación direta, y de este modo podemos ver las mismas correlaciones de una parte que de otra.

Respecto a dragones, podemos ver:

* Es más importante para el equipo 2 hacerse el primer dragón de cara a hacerse más que el que el equipo 1 haga el mismo.

Finalmente, respecto a el momento de la creación de la partida, podemos observar:

* No tiene ninguna relación el paso del tiempo cno el aumento o disminución de ninguna de las variables.

## PCA

Vamos a hacer ahora un análisis de las componentes principales: Para los cálculos, uso la matriz con el centrado y escalado ya hechos

resultado.pca <- PCA(data.games.corr, graph = FALSE)  
  
#Con la siguiente línea podemos ver que podemos hacer con esto calculado  
print(resultado.pca)

## \*\*Results for the Principal Component Analysis (PCA)\*\*  
## The analysis was performed on 51490 individuals, described by 19 variables  
## \*The results are available in the following objects:  
##   
## name description   
## 1 "$eig" "eigenvalues"   
## 2 "$var" "results for the variables"   
## 3 "$var$coord" "coord. for the variables"   
## 4 "$var$cor" "correlations variables - dimensions"  
## 5 "$var$cos2" "cos2 for the variables"   
## 6 "$var$contrib" "contributions of the variables"   
## 7 "$ind" "results for the individuals"   
## 8 "$ind$coord" "coord. for the individuals"   
## 9 "$ind$cos2" "cos2 for the individuals"   
## 10 "$ind$contrib" "contributions of the individuals"   
## 11 "$call" "summary statistics"   
## 12 "$call$centre" "mean of the variables"   
## 13 "$call$ecart.type" "standard error of the variables"   
## 14 "$call$row.w" "weights for the individuals"   
## 15 "$call$col.w" "weights for the variables"

Nos interesa ver los eigenvalues, que son los que presentarán la cantidad de varianza que aportan las variables:

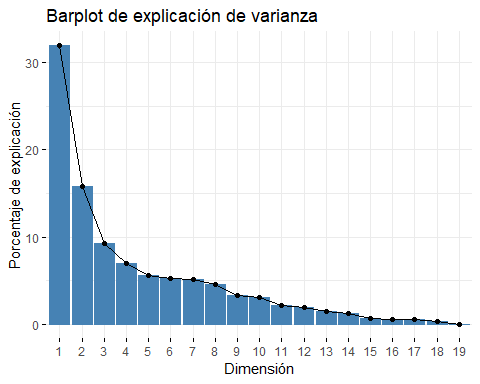
eigenvalues.PCA <- resultado.pca$eig  
eigenvalues.PCA

## eigenvalue percentage of variance cumulative percentage of variance  
## comp 1 6.064459e+00 3.191820e+01 31.91820  
## comp 2 3.000738e+00 1.579336e+01 47.71156  
## comp 3 1.771844e+00 9.325495e+00 57.03706  
## comp 4 1.316075e+00 6.926710e+00 63.96377  
## comp 5 1.066890e+00 5.615210e+00 69.57898  
## comp 6 9.958756e-01 5.241450e+00 74.82043  
## comp 7 9.704525e-01 5.107645e+00 79.92807  
## comp 8 8.740157e-01 4.600083e+00 84.52816  
## comp 9 6.403057e-01 3.370030e+00 87.89819  
## comp 10 5.797901e-01 3.051527e+00 90.94971  
## comp 11 4.053622e-01 2.133485e+00 93.08320  
## comp 12 3.670873e-01 1.932039e+00 95.01524  
## comp 13 2.816983e-01 1.482623e+00 96.49786  
## comp 14 2.465127e-01 1.297435e+00 97.79529  
## comp 15 1.253587e-01 6.597825e-01 98.45508  
## comp 16 1.175431e-01 6.186477e-01 99.07372  
## comp 17 1.007805e-01 5.304239e-01 99.60415  
## comp 18 7.521185e-02 3.958518e-01 100.00000  
## comp 19 7.248996e-26 3.815261e-25 100.00000

Como podemos ver, tenemos 19 componentes principales (una por cada dimensión), y vemos que solo con 10 variables ya tenemos un 90% de explicación. Además, de cara a la representación en dos dimensiones, podemos ver que con solo las dos variables con más varianza tenemos una explicación del 47,7%.

Ahora, para completar este apartado de PCA, lo que voy a hacer es sacar la gráfica de la varianza acumulada con los valores anteriores:

plotPCA <- fviz\_screeplot(resultado.pca, ncp=19, main="Barplot de explicación de varianza", ylab="Porcentaje de explicación", xlab="Dimensión")  
plot(plotPCA)



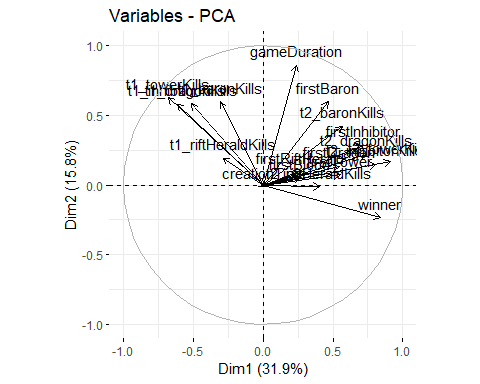
Ahora voy a sacar un “Factor Map” de las variables. Esto lo puedo hacer gracias a las coordenadas que me da una de las variables tras hacer el PCA. Así, voy primero a ver la tabla y luego voy a sacar el mapa:

head(resultado.pca$var$coord)

## Dim.1 Dim.2 Dim.3 Dim.4 Dim.5  
## creationTime 0.01723817 -0.01149443 -0.06290260 0.04393011 -0.25111184  
## gameDuration 0.24050089 0.86002372 -0.09989262 0.13764973 -0.07795331  
## winner 0.84048823 -0.23457181 -0.14685233 -0.11282396 -0.04006251  
## firstBlood 0.26968344 0.05182651 0.13357189 0.34780625 -0.23151329  
## firstTower 0.55683600 0.07472362 0.19566560 0.37066228 -0.14932506  
## firstInhibitor 0.72099264 0.28414084 -0.07367165 -0.02461892 -0.03806185

Como se puede ver, me está poniendo mis 24 variables en 5 dimensiones, con unas coordenadas concretas. Ahora, lo que voy a hacer, es representarlo. Con esta representación podré sacar algunas conclusiones:

fviz\_pca\_var(resultado.pca)

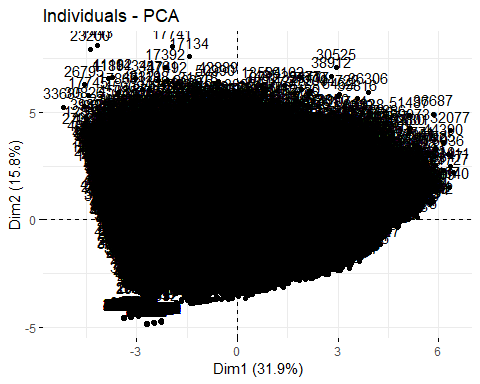


Es interesante ver como no hay nada que vaya en la misma dirección que la victoria. T1 tiene que ser interpretado de una manera en espejo respecto al eje de las Y, y vemos como muchísimas variables como el heraldo, first blood, número de dragones… todas tienen más o menos la misma importancia de cara a conseguir la victoria.

Vemos como la duración del juego forma un ángulo de 90 grados con la victoria, lo cual significa que no tiene nada que ver.

Ahora vamos a ver en un mapa, sobre las dos componentes principales, todas las partidas:

fviz\_pca\_ind(resultado.pca)



Exportamos estos gráficos del PCA de los individuos…

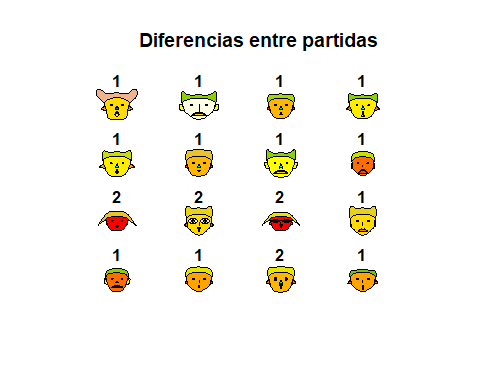
pdf("../imagenes/PCAIndividuos.pdf")  
  
plot(plotPCA)  
fviz\_pca\_var(resultado.pca)  
fviz\_pca\_ind(resultado.pca)  
  
dev.off

## function (which = dev.cur())   
## {  
## if (which == 1)   
## stop("cannot shut down device 1 (the null device)")  
## .External(C\_devoff, as.integer(which))  
## dev.cur()  
## }  
## <bytecode: 0x00000000134d26f0>  
## <environment: namespace:grDevices>

Como se puede apreciar en este gráfico también, prácticamente todas las partidas forman un gran cluster (representado sobre las dos componentes con más variabilidad), y esto nos demuestra que no hay demasiada varianza entre unas partidas y otras. En otras palabras, no podemos distinguir fácilmente varios tipos de partidas, todas son parecidas y no hay diferencias suficientes para clasificar en grupos, más que en uno grande.

Para analizar la variabilidad de algunas partidas, vamos a verlo mediante el método de las caras de Chernoff:

muestra\_partidas <- data.games[1:16, 27:31]  
muestra\_partidas <- cbind(muestra\_partidas, data.games[1:16, 52:56])  
faces(muestra\_partidas[1:10], face.type=1,labels=data.games[1:16, 5], main = "Diferencias entre partidas", scale = TRUE, cex = 1.5)



## effect of variables:  
## modified item Var   
## "height of face " "t1\_towerKills"   
## "width of face " "t1\_inhibitorKills"   
## "structure of face" "t1\_baronKills"   
## "height of mouth " "t1\_dragonKills"   
## "width of mouth " "t1\_riftHeraldKills"  
## "smiling " "t2\_towerKills"   
## "height of eyes " "t2\_inhibitorKills"   
## "width of eyes " "t2\_baronKills"   
## "height of hair " "t2\_dragonKills"   
## "width of hair " "t2\_riftHeraldKills"  
## "style of hair " "t1\_towerKills"   
## "height of nose " "t1\_inhibitorKills"   
## "width of nose " "t1\_baronKills"   
## "width of ear " "t1\_dragonKills"   
## "height of ear " "t1\_riftHeraldKills"

Lo exportamos a PDF para tener más calidad al pasarlo a vectorial…

pdf("../imagenes/chernoff partidas.pdf")  
faces(muestra\_partidas[1:10], face.type=1,labels=data.games[1:16, 5], main = "Diferencias entre partidas", scale = TRUE, cex = 1.5)

## effect of variables:  
## modified item Var   
## "height of face " "t1\_towerKills"   
## "width of face " "t1\_inhibitorKills"   
## "structure of face" "t1\_baronKills"   
## "height of mouth " "t1\_dragonKills"   
## "width of mouth " "t1\_riftHeraldKills"  
## "smiling " "t2\_towerKills"   
## "height of eyes " "t2\_inhibitorKills"   
## "width of eyes " "t2\_baronKills"   
## "height of hair " "t2\_dragonKills"   
## "width of hair " "t2\_riftHeraldKills"  
## "style of hair " "t1\_towerKills"   
## "height of nose " "t1\_inhibitorKills"   
## "width of nose " "t1\_baronKills"   
## "width of ear " "t1\_dragonKills"   
## "height of ear " "t1\_riftHeraldKills"

dev.off

## function (which = dev.cur())   
## {  
## if (which == 1)   
## stop("cannot shut down device 1 (the null device)")  
## .External(C\_devoff, as.integer(which))  
## dev.cur()  
## }  
## <bytecode: 0x00000000134d26f0>  
## <environment: namespace:grDevices>

Como podemos ver, hay diferencias entre las partidas ganadas por el equipo 1 (caras en general más grandes y geométricas, pelo hacia arriba) y las del equipo 2 (pelo hacia abajo y amarillo, cara roja en la mitad, cara redondeada…)

## Preguntas

Ahora que tenemos las correlaciones vistas, podemos pasar a hacernos preguntas.

### ¿Campeón más baneado por temporada?

Uno de los elementos que más se tienen en cuenta a la hora de analizar el League of Legends en los e-sports es el campeón más baneado por temporada o campeonato. Esto es porque desde Riot Games hacen cambios a las habilidades y fuerza de los personajes, de tal manera que algunas veces algunos campeones son demasiado fuertes y se pueden bloquear al inicio de la partida. Vamos a ver cuáles han sido:

Para analizar esto, contaré la cantidad de ocurrencias y veré, en estas más de 50.000 partidas, cual es el campeón más temido entre los jugadores, tras sustiruir por los nombres.

# Lo que voy a hacer es contar por columnas, y luego sumo todas las columnas que he contado. Así obtengo el recuento final  
  
# TEAM 1  
  
cont.bans.t1.1 <- ddply(data.games,.(t1\_ban1),nrow)  
cont.bans.t1.1 <- cont.bans.t1.1[order(cont.bans.t1.1$V1, decreasing = TRUE), ]  
cont.bans.t1.2 <- ddply(data.games,.(t1\_ban2),nrow)  
cont.bans.t1.2 <- cont.bans.t1.2[order(cont.bans.t1.2$V1, decreasing = TRUE), ]  
cont.bans.t1.3 <- ddply(data.games,.(t1\_ban3),nrow)  
cont.bans.t1.3 <- cont.bans.t1.3[order(cont.bans.t1.3$V1, decreasing = TRUE), ]  
cont.bans.t1.4 <- ddply(data.games,.(t1\_ban4),nrow)  
cont.bans.t1.4 <- cont.bans.t1.4[order(cont.bans.t1.4$V1, decreasing = TRUE), ]  
cont.bans.t1.5 <- ddply(data.games,.(t1\_ban5),nrow)  
cont.bans.t1.5 <- cont.bans.t1.5[order(cont.bans.t1.5$V1, decreasing = TRUE), ]  
  
# TEAM 2  
  
cont.bans.t2.1 <- ddply(data.games,.(t2\_ban1),nrow)  
cont.bans.t2.1 <- cont.bans.t2.1[order(cont.bans.t2.1$V1, decreasing = TRUE), ]  
cont.bans.t2.2 <- ddply(data.games,.(t2\_ban2),nrow)  
cont.bans.t2.2 <- cont.bans.t2.2[order(cont.bans.t2.2$V1, decreasing = TRUE), ]  
cont.bans.t2.3 <- ddply(data.games,.(t2\_ban3),nrow)  
cont.bans.t2.3 <- cont.bans.t2.3[order(cont.bans.t2.3$V1, decreasing = TRUE), ]  
cont.bans.t2.4 <- ddply(data.games,.(t2\_ban4),nrow)  
cont.bans.t2.4 <- cont.bans.t2.4[order(cont.bans.t2.4$V1, decreasing = TRUE), ]  
cont.bans.t2.5 <- ddply(data.games,.(t2\_ban5),nrow)  
cont.bans.t2.5 <- cont.bans.t2.5[order(cont.bans.t2.5$V1, decreasing = TRUE), ]  
  
# Ahora lo que tengo que hacer es sumar todas estas columnas de V1 según el valor de name...  
  
df.bans <- left\_join(cont.bans.t1.1, cont.bans.t1.2, by =c("t1\_ban1" = "t1\_ban2"))  
df.bans <- left\_join(df.bans, cont.bans.t1.3, by =c("t1\_ban1" = "t1\_ban3"))  
df.bans <- left\_join(df.bans, cont.bans.t1.4, by =c("t1\_ban1" = "t1\_ban4"))  
df.bans <- left\_join(df.bans, cont.bans.t1.5, by =c("t1\_ban1" = "t1\_ban5"))  
  
df.bans <- left\_join(df.bans, cont.bans.t2.1, by =c("t1\_ban1" = "t2\_ban1"))  
df.bans <- left\_join(df.bans, cont.bans.t2.2, by =c("t1\_ban1" = "t2\_ban2"))  
df.bans <- left\_join(df.bans, cont.bans.t2.3, by =c("t1\_ban1" = "t2\_ban3"))  
df.bans <- left\_join(df.bans, cont.bans.t2.4, by =c("t1\_ban1" = "t2\_ban4"))  
df.bans <- left\_join(df.bans, cont.bans.t2.5, by =c("t1\_ban1" = "t2\_ban5"))  
  
df.bans$total <- rowSums( df.bans[,2:11] )  
  
remove(cont.bans.t1.1)  
remove(cont.bans.t1.2)  
remove(cont.bans.t1.3)  
remove(cont.bans.t1.4)  
remove(cont.bans.t1.5)  
remove(cont.bans.t2.1)  
remove(cont.bans.t2.2)  
remove(cont.bans.t2.3)  
remove(cont.bans.t2.4)  
remove(cont.bans.t2.5)  
  
df.bans <- df.bans[order(df.bans$total, decreasing = TRUE), ]  
df.bans <- df.bans[, -2:-11]  
head(df.bans)

## t1\_ban1 total  
## 1 157 33015  
## 2 238 25393  
## 3 31 25175  
## 4 122 22870  
## 5 40 21390  
## 6 119 20262

# Finalmente, junto con la tabla de data.champs para ponerles nombre...  
  
df.bans <- left\_join(df.bans, data.champs, by=c("t1\_ban1" = "id"))  
df.bans <- df.bans[, -1]  
head(df.bans)

## total name  
## 1 33015 Yasuo  
## 2 25393 Zed  
## 3 25175 ChoGath  
## 4 22870 Darius  
## 5 21390 Janna  
## 6 20262 Draven

Ahora que tenemos todos los bans contados, es hora de hacer un wordcloud para poder verlo visualmente:

set.seed(9999) # Para el mantenimiento del mismo patrón  
  
wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, random.order=FALSE,   
 rot.per=0.5, colors=c("Orange","Purple","Pink", "Red", "Yellow", "Green", "Blue", "Black"))

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Fiddlesticks could  
## not be fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Nidalee could not  
## be fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Mordekaiser could  
## not be fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Taric could not be  
## fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Lissandra could not  
## be fit on page. It will not be plotted.

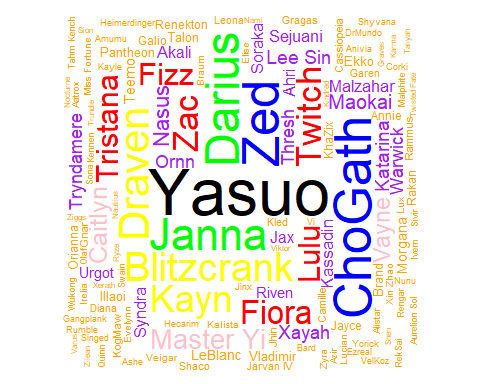
## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Volibear could not  
## be fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Poppy could not be  
## fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Karthus could not  
## be fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Udyr could not be  
## fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Skarner could not  
## be fit on page. It will not be plotted.



# Ratio del más baneado  
  
print("El porcentaje de baneo a Yasuo es de: ")

## [1] "El porcentaje de baneo a Yasuo es de: "

ratio.ban.yasuo <- df.bans$total[1]/sum(df.bans$total)  
print(ratio.ban.yasuo)

## [1] 0.06411925

Exportamos el wordcloud a PDF para tenerlo vectorial…

set.seed(9999) # Para el mantenimiento del mismo patrón  
  
wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, random.order=FALSE,   
 rot.per=0.5, colors=c("Orange","Purple","Pink", "Red", "Yellow", "Green", "Blue", "Black"))

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Fiddlesticks could  
## not be fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Nidalee could not  
## be fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Mordekaiser could  
## not be fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Taric could not be  
## fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Lissandra could not  
## be fit on page. It will not be plotted.

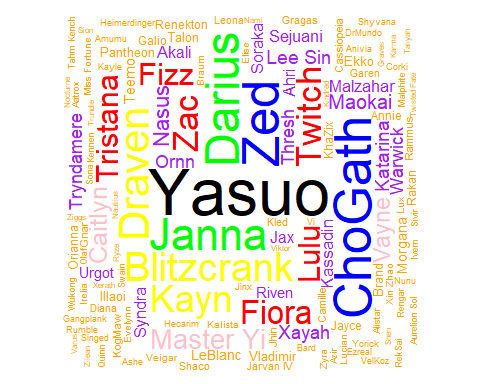
## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Volibear could not  
## be fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Poppy could not be  
## fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Karthus could not  
## be fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Udyr could not be  
## fit on page. It will not be plotted.

## Warning in wordcloud(words = df.bans$name, freq = df.bans$total, min.freq = 1, : Skarner could not  
## be fit on page. It will not be plotted.



dev.off

## function (which = dev.cur())   
## {  
## if (which == 1)   
## stop("cannot shut down device 1 (the null device)")  
## .External(C\_devoff, as.integer(which))  
## dev.cur()  
## }  
## <bytecode: 0x00000000134d26f0>  
## <environment: namespace:grDevices>

Pregunta respondida.

### ¿Matar un mayor número de dragones aumenta las posibilidades de victoria?

Esto lo vamos a responder para el equipo 1 y para el equipo 2, de tal manera que podamos ver si tiene más influencia en uno u en otro.

Para esto, necesito la cantidad de dragones matados por el equipo que venció y por el equipo que perdió, y esos datos los tengo en el dataset de data.games.

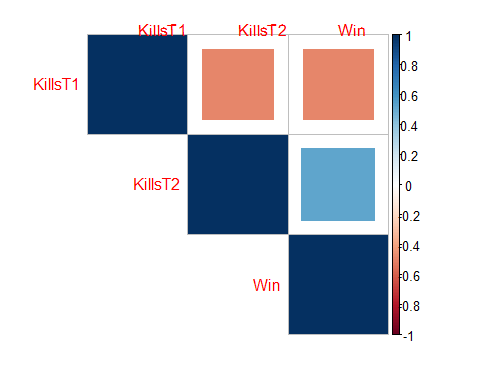
data.dragons.win <- cbind(data.games$t1\_dragonKills, data.games$t2\_dragonKills, data.games$winner)  
colnames(data.dragons.win) <- c("KillsT1", "KillsT2", "Win")  
head(data.dragons.win)

## KillsT1 KillsT2 Win  
## [1,] 3 1 1  
## [2,] 2 0 1  
## [3,] 1 1 1  
## [4,] 2 0 1  
## [5,] 3 1 1  
## [6,] 1 3 1

Primero vamos a estudiar la correlación. La hemos visto anteriormente, pero vamos a hacerlo ahora en especial de esta tabla para verlo de una manera más grande:

res.dragons.win <- cor(data.dragons.win, method = "spearman")  
options(width = 100)  
res.dragons.win.round <- round(res.dragons.win, 2)

corrplot(res.dragons.win.round, method="square", type = "upper", tl.srt = 0.7)



Lo exportamos a PDF para poderlo ver con mayor calidad…

pdf("../imagenes/DrakesCorr.pdf")  
  
corrplot(res.dragons.win.round, method="square", type = "upper", tl.srt = 0.7)  
  
dev.off

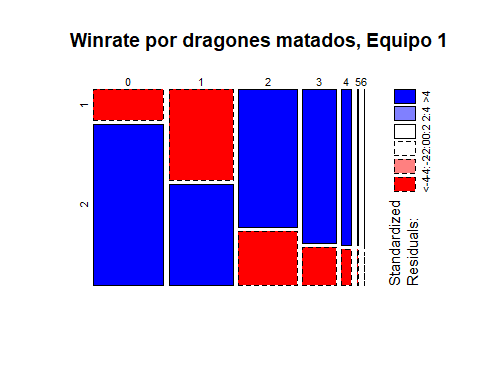
## function (which = dev.cur())   
## {  
## if (which == 1)   
## stop("cannot shut down device 1 (the null device)")  
## .External(C\_devoff, as.integer(which))  
## dev.cur()  
## }  
## <bytecode: 0x00000000134d26f0>  
## <environment: namespace:grDevices>

Como podemos ver, tenemos una correlación de aproximadamente el 50% entre el aumento del número de dragones matados y conseguir la victoria.

#### Equipo 1

Vamos a ver el número de dragones matados por el equipo 1 respecto a la victoria:

mosaicplot(table(data.dragons.win[, 1], data.dragons.win[, 3]), main='Winrate por dragones matados, Equipo 1', shade=TRUE)



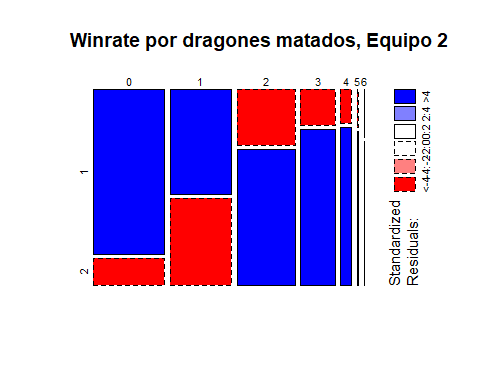
Como podemos ver, el equipo 1, si no mata a ningún dragón, tiene serias posibilidades de no ser el equipo que gane. Conforme va matando dragones, aumenta la posibilidad, especialmente con el paso de 1 a 2 dragones, donde se planta con una mayor parte de las posibilidades.

A partir de los 3 dragones matados, el hecho de matar más no tiene prácticamente influencia en el resultado final.

#### Equipo 2

Procedemos igual con el equipo 2:

mosaicplot(table(data.dragons.win[, 2], data.dragons.win[, 3]), main='Winrate por dragones matados, Equipo 2', shade=TRUE)



Exportamos los MosaicPlot a PDF para tenerlos vectoriales…

pdf("../imagenes/Drakes MosaicPlot.pdf")  
  
mosaicplot(table(data.dragons.win[, 1], data.dragons.win[, 3]), main='Winrate por dragones matados, Equipo 1', shade=TRUE)  
mosaicplot(table(data.dragons.win[, 2], data.dragons.win[, 3]), main='Winrate por dragones matados, Equipo 2', shade=TRUE)  
  
dev.off

## function (which = dev.cur())   
## {  
## if (which == 1)   
## stop("cannot shut down device 1 (the null device)")  
## .External(C\_devoff, as.integer(which))  
## dev.cur()  
## }  
## <bytecode: 0x00000000134d26f0>  
## <environment: namespace:grDevices>

Como se puede observar, las distribuciones son prácticamente similares, solo que en espejo como es normal.

### ¿Cuales son los campeones más escogidos?

Sabemos que Yasuo es el campeón más baneado, con un 6% de banrate. Ahora me pregunto cual es el campeón más escogido para jugar.

Para ello, tenemos que seguir la misma metodología que en la parte de los baneos:

# Lo que voy a hacer es contar por columnas, y luego sumo todas las columnas que he contado. Así obtengo el recuento final  
  
# TEAM 1  
  
cont.picks.t1.1 <- ddply(data.games,.(t1\_champ1id),nrow)  
cont.picks.t1.1 <- cont.picks.t1.1[order(cont.picks.t1.1$V1, decreasing = TRUE), ]  
cont.picks.t1.2 <- ddply(data.games,.(t1\_champ2id),nrow)  
cont.picks.t1.2 <- cont.picks.t1.2[order(cont.picks.t1.2$V1, decreasing = TRUE), ]  
cont.picks.t1.3 <- ddply(data.games,.(t1\_champ3id),nrow)  
cont.picks.t1.3 <- cont.picks.t1.3[order(cont.picks.t1.3$V1, decreasing = TRUE), ]  
cont.picks.t1.4 <- ddply(data.games,.(t1\_champ4id),nrow)  
cont.picks.t1.4 <- cont.picks.t1.4[order(cont.picks.t1.4$V1, decreasing = TRUE), ]  
cont.picks.t1.5 <- ddply(data.games,.(t1\_champ5id),nrow)  
cont.picks.t1.5 <- cont.picks.t1.5[order(cont.picks.t1.5$V1, decreasing = TRUE), ]  
  
# TEAM 2  
  
cont.picks.t2.1 <- ddply(data.games,.(t2\_champ1id),nrow)  
cont.picks.t2.1 <- cont.picks.t2.1[order(cont.picks.t2.1$V1, decreasing = TRUE), ]  
cont.picks.t2.2 <- ddply(data.games,.(t2\_champ2id),nrow)  
cont.picks.t2.2 <- cont.picks.t2.2[order(cont.picks.t2.2$V1, decreasing = TRUE), ]  
cont.picks.t2.3 <- ddply(data.games,.(t2\_champ3id),nrow)  
cont.picks.t2.3 <- cont.picks.t2.3[order(cont.picks.t2.3$V1, decreasing = TRUE), ]  
cont.picks.t2.4 <- ddply(data.games,.(t2\_champ4id),nrow)  
cont.picks.t2.4 <- cont.picks.t2.4[order(cont.picks.t2.4$V1, decreasing = TRUE), ]  
cont.picks.t2.5 <- ddply(data.games,.(t2\_champ5id),nrow)  
cont.picks.t2.5 <- cont.picks.t2.5[order(cont.picks.t2.5$V1, decreasing = TRUE), ]  
  
# Ahora lo que tengo que hacer es sumar todas estas columnas de V1 según el valor de name...  
  
df.picks <- left\_join(cont.picks.t1.1, cont.picks.t1.2, by =c("t1\_champ1id" = "t1\_champ2id"))  
df.picks <- left\_join(df.picks, cont.picks.t1.3, by =c("t1\_champ1id" = "t1\_champ3id"))  
df.picks <- left\_join(df.picks, cont.picks.t1.4, by =c("t1\_champ1id" = "t1\_champ4id"))  
df.picks <- left\_join(df.picks, cont.picks.t1.5, by =c("t1\_champ1id" = "t1\_champ5id"))  
  
df.picks <- left\_join(df.picks, cont.picks.t2.1, by =c("t1\_champ1id" = "t2\_champ1id"))  
df.picks <- left\_join(df.picks, cont.picks.t2.2, by =c("t1\_champ1id" = "t2\_champ2id"))  
df.picks <- left\_join(df.picks, cont.picks.t2.3, by =c("t1\_champ1id" = "t2\_champ3id"))  
df.picks <- left\_join(df.picks, cont.picks.t2.4, by =c("t1\_champ1id" = "t2\_champ4id"))  
df.picks <- left\_join(df.picks, cont.picks.t2.5, by =c("t1\_champ1id" = "t2\_champ5id"))  
  
df.picks$total <- rowSums( df.picks[,2:11] )  
  
remove(cont.picks.t1.1)  
remove(cont.picks.t1.2)  
remove(cont.picks.t1.3)  
remove(cont.picks.t1.4)  
remove(cont.picks.t1.5)  
remove(cont.picks.t2.1)  
remove(cont.picks.t2.2)  
remove(cont.picks.t2.3)  
remove(cont.picks.t2.4)  
remove(cont.picks.t2.5)  
  
df.picks <- df.picks[order(df.picks$total, decreasing = TRUE), ]  
df.picks <- df.picks[, -2:-11]  
head(df.picks)

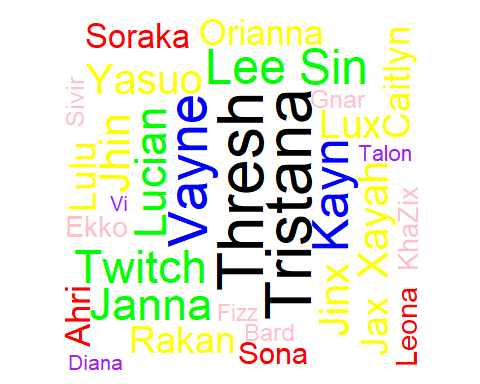
## t1\_champ1id total  
## 2 412 13002  
## 1 18 12983  
## 3 67 10658  
## 4 141 9853  
## 5 64 9188  
## 6 29 8838

# Finalmente, junto con la tabla de data.champs para ponerles nombre...  
  
df.picks <- left\_join(df.picks, data.champs, by=c("t1\_champ1id" = "id"))  
df.picks <- df.picks[, -1]  
head(df.picks)

## total name  
## 1 13002 Thresh  
## 2 12983 Tristana  
## 3 10658 Vayne  
## 4 9853 Kayn  
## 5 9188 Lee Sin  
## 6 8838 Twitch

Ahora que tenemos todos los picks contados, es hora de hacer un wordcloud para poder verlo visualmente:

set.seed(9998) # Para el mantenimiento del mismo patrón  
  
wordcloud(words = df.picks$name, freq = df.picks$total, min.freq = 3000, random.order=FALSE,  
 rot.per=0.5, colors=c("Orange","Purple","Pink", "Red", "Yellow", "Green", "Blue", "Black")  
 )



# Ratio del más baneado  
  
print("El porcentaje de pick de Thresh es de: ")

## [1] "El porcentaje de pick de Thresh es de: "

ratio.pick.thresh <- df.picks$total[1]/sum(df.picks$total)  
print(ratio.pick.thresh)

## [1] 0.02525151

Exporto el Wordcloud a PDF para tenerlo vectorial…

pdf("../imagenes/wordcloudPicks.pdf")  
  
set.seed(9998) # Para el mantenimiento del mismo patrón  
  
wordcloud(words = df.picks$name, freq = df.picks$total, min.freq = 3000, random.order=FALSE,  
 rot.per=0.5, colors=c("Orange","Purple","Pink", "Red", "Yellow", "Green", "Blue", "Black")  
 )  
  
dev.off

## function (which = dev.cur())   
## {  
## if (which == 1)   
## stop("cannot shut down device 1 (the null device)")  
## .External(C\_devoff, as.integer(which))  
## dev.cur()  
## }  
## <bytecode: 0x00000000134d26f0>  
## <environment: namespace:grDevices>

Está muy bien que Thresh y Tristana sean los más elegidos, pero… ¿esto es porque conllevan mayor victoria de partidas? Vamos a calcularlo:

### ¿Qué camepones conllevan un mayor winrate?

Para esto, lo primero que tenemos que hacer es crear un nuevo dataframe con los campeones que han ganado cada partida:

dataframe.winners.1 <- filter(data.games, winner == "1")   
  
# En este dataframe, solo me interesan las columnas de los campeones elegidos por el equipo 1, por lo que solo me quedo con ellas:  
  
dataframe.winners.1 <- dataframe.winners.1[, c(12, 15, 18, 21, 24)]  
colnames(dataframe.winners.1) <- c("Pick1", "Pick2", "Pick3", "Pick4", "Pick5")  
  
# Hacemos lo mismo con los ganadores del equipo 2  
  
dataframe.winners.2 <- filter(data.games, winner == "2")   
  
# En este dataframe, solo me interesan las columnas de los campeones elegidos por el equipo 1, por lo que solo me quedo con ellas:  
  
dataframe.winners.2 <- dataframe.winners.2[, c(37, 40, 43, 46, 49)]  
colnames(dataframe.winners.2) <- c("Pick1", "Pick2", "Pick3", "Pick4", "Pick5")  
  
# Ahora los juntamos...  
  
dataframe.winners <- rbind(dataframe.winners.1, dataframe.winners.2)  
  
remove(dataframe.winners.1)  
remove(dataframe.winners.2)

Ahora ya tenemos preparado el dataframe para poder jugar con él. Procedemos contando el número de ocurrencias que hay de cada campeón, y con esto veremos cual es el campeón que más se repite en este dataframe donde solo están las victorias:

# Ahora voy a llevar todas las columnas a una:  
  
dataframe.winners <- data.frame(all = c(dataframe.winners[,"Pick1"],  
 dataframe.winners[,"Pick2"],  
 dataframe.winners[,"Pick3"],   
 dataframe.winners[,"Pick4"],   
 dataframe.winners[,"Pick5"]))  
  
head(dataframe.winners)

## all  
## 1 8  
## 2 119  
## 3 18  
## 4 57  
## 5 19  
## 6 40

dataframe.winners <- left\_join(dataframe.winners, data.champs, by=c("all" = "id"))  
  
dataframe.winners <- dataframe.winners[, -1]  
  
head(dataframe.winners)

## [1] Vladimir Draven Tristana Maokai Warwick Janna   
## 138 Levels: Aatrox Ahri Akali Alistar Amumu Anivia Annie Ashe Aurelion Sol Azir Bard ... Zyra

# Ahora tenemos todo el dataframe en una única columna, donde solo nos queda ordenar y contar el número de ocurrencias:  
  
dataframe.winners <- dataframe.winners[order(dataframe.winners)]  
head(dataframe.winners)

## [1] Aatrox Aatrox Aatrox Aatrox Aatrox Aatrox  
## 138 Levels: Aatrox Ahri Akali Alistar Amumu Anivia Annie Ashe Aurelion Sol Azir Bard ... Zyra

# Contamos  
  
cantidad <- as.data.frame(table(dataframe.winners))  
  
cantidad <- cantidad[order(cantidad$Freq, decreasing = TRUE), ]  
colnames(cantidad) <- c("name", "freq")  
head(cantidad)

## name freq  
## 112 Tristana 6713  
## 111 Thresh 6143  
## 120 Vayne 5498  
## 42 Janna 4826  
## 54 Kayn 4807  
## 116 Twitch 4665

Como podemos ver, los campeones más elegidos son Tristana y Thresh, que casualmente, como acabamos de probar, son los que tienen más victorias en su haber. También, la tercera con más picks es Vayne, y también es la tercera en cantidad de victorias.

Vamos a calcular el ratio pick-victory:

# Para hacer esto, voy a juntar los dos dataframes en uno solo, juntando por el valor del campeón (nombre), y una vez hecho esto podré restar entre mismas columnas  
  
# No puedo hacer un join normal porque todos mis elementos de texto son factors, por eso lo reconvierto  
  
cantidad <- data.frame(cantidad, stringsAsFactors = FALSE)  
colnames(cantidad) <- c("name", "freq\_win")  
  
df.picks <- data.frame(df.picks, stringsAsFactors = FALSE)  
colnames(df.picks) <- c("total\_picks", "name")  
  
dataframe.pick.win <- inner\_join(cantidad, df.picks, by="name")  
  
# Ahora que tengo los datos a mano, podemos hacer las divisiones:  
  
dataframe.pick.win <- transform(dataframe.pick.win, ratio = dataframe.pick.win[, 2] / dataframe.pick.win[, 3])  
  
dataframe.pick.win <- dataframe.pick.win[order(dataframe.pick.win$ratio, decreasing = TRUE), ]  
  
head(dataframe.pick.win)

## name freq\_win total\_picks ratio  
## 4 Janna 4826 8691 0.5552871  
## 29 Sona 2942 5429 0.5419046  
## 120 Yorick 744 1378 0.5399129  
## 66 Rammus 1614 2997 0.5385385  
## 87 Anivia 1207 2252 0.5359680  
## 118 Singed 762 1425 0.5347368

Es interesante ver que, a la vista de estos resultados, la gente banea lo que ve mucho, no lo que de verdad gana partidas. Como podemos observar, Janna, Sona y Yorick son los 3 campeones que más porcentaje de partidas ganan. En cambio, a la hora de escogerlos los vemos en las posiciones 7, 30 y 124. Respecto a bans, si tantas partidas ganan la masa debería de banearlos, pero se encuentran en las posiciones 5, 91 y 96, por lo que la gente no se da cuenta del peligro de estos campeones en las manos adecuadas.

### ¿El lado rojo (equipo 2) pierde más partidas?

Una de las afirmaciones que corre por la comunidad de League of Legends es la creencia de que el lado rojo, correspondiente con el equipo 2 en nuestro dataset, es el que pierde un mayor número de partidas. Para visualizar esto, lo único que hay que hacer es obtener el winrate del equipo rojo respecto al total de partidas:

columna.wins <- data.games$winner  
cantidad.wins <- as.data.frame(table(columna.wins))  
colnames(cantidad.wins) <- c("winner", "frequency")  
head(cantidad.wins)

## winner frequency  
## 1 1 26077  
## 2 2 25413

# Si calculamos el ratio...  
  
ratio.red <- cantidad.wins[2, 2] / (cantidad.wins[1, 2] + cantidad.wins[2, 2])  
ratio.red

## [1] 0.4935521

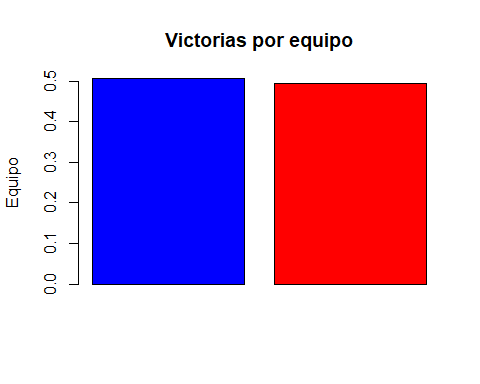
ratio.blue <- cantidad.wins[1, 2] / (cantidad.wins[1, 2] + cantidad.wins[2, 2])  
ratio.blue

## [1] 0.5064479

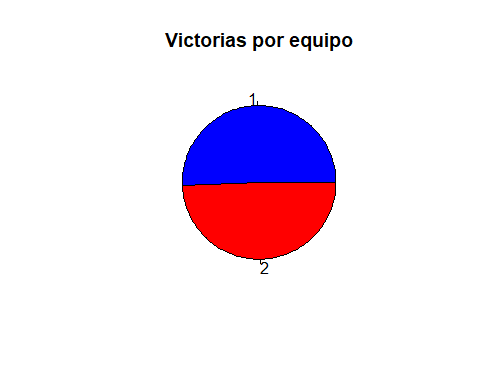
Es decir, el grupo que juega en el lado rojo gana el 49,35% de las partidas, mientras que el que juega en el lado azul gana el 50,64%.

Vamos a verlo en forma de gráfica:

ratios\_win <- c(ratio.blue, ratio.red)  
barplot(ratios\_win, main="Victorias por equipo", ylab="Equipo", col = c("Blue", "Red"))



pie(table(data.games$winner), main = "Victorias por equipo", col = c("Blue", "Red"))



Lo exportamos a PDF…

pdf("../imagenes/Winrate Graphs.pdf")  
barplot(ratios\_win, main="Victorias por equipo", ylab="Equipo", col = c("Blue", "Red"))  
pie(table(data.games$winner), main = "Victorias por equipo", col = c("Blue", "Red"))  
dev.off

## function (which = dev.cur())   
## {  
## if (which == 1)   
## stop("cannot shut down device 1 (the null device)")  
## .External(C\_devoff, as.integer(which))  
## dev.cur()  
## }  
## <bytecode: 0x00000000134d26f0>  
## <environment: namespace:grDevices>

### ¿Quién consigue más primeros objetivos?

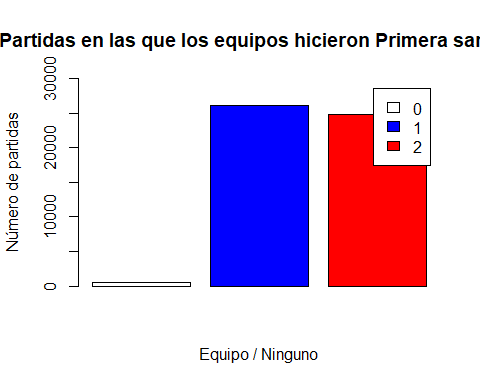
Los primeros objetivos son cruciales en la partida, ya que algunos dan extra de oro, pero todos empiezan a decantar la partida a tu favor. Debido a ello, el análisis de si existe alguna diferencia entre los equipos a la hora de conseguir los primeros objetivos se plantea crucial.

Para ello, contaré para cada columna de primeros objetivos qué equipo ha conseguido ser el mejor, y por qué porcentaje, y entonces determinaré si hay alguna ventaja en alguno de los objetivos.

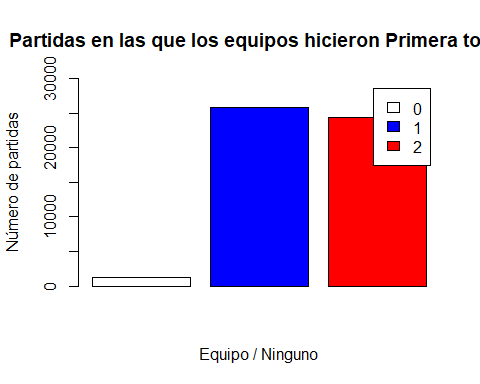
# Los datos de esto los tenemos en data.games. Para aclarar, vamos a hacer un dataset específico con los datos de los objetivos...  
  
data.first.objectives <- data.games[, 6:11]  
  
# Es importante remarcar que aquí hay 3 valores: 0 si nadie lo ha conseguido, 1 si lo ha conseguido el equipo azul y 2 si lo ha conseguido el equipo rojo.  
  
head(data.first.objectives)

## firstBlood firstTower firstInhibitor firstBaron firstDragon firstRiftHerald  
## 1 2 1 1 1 1 2  
## 2 1 1 1 0 1 1  
## 3 2 1 1 1 2 0  
## 4 1 1 1 1 1 0  
## 5 2 1 1 1 1 0  
## 6 2 2 1 1 2 0

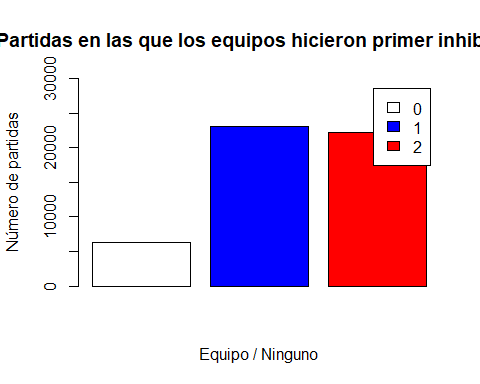
# Ahora mi objetivo es contar de cada una de las líneas lo que hay, y de aquí sacar conclusiones:  
  
firstblood.qty <- as.data.frame(table(data.first.objectives[, 1]))  
firsttower.qty <- as.data.frame(table(data.first.objectives[, 2]))  
firstinhib.qty <- as.data.frame(table(data.first.objectives[, 3]))  
firstbaron.qty <- as.data.frame(table(data.first.objectives[, 4]))  
firstdragon.qty <- as.data.frame(table(data.first.objectives[, 5]))  
firstherald.qty <- as.data.frame(table(data.first.objectives[, 6]))  
  
# Ahora lo vemos bien en barplots...  
  
array.firstblood <- c(firstblood.qty[1, 2],firstblood.qty[2, 2], firstblood.qty[3, 2])  
barplot(array.firstblood, main="Partidas en las que los equipos hicieron Primera sangre", xlab="Equipo / Ninguno", ylab = "Número de partidas",   
 ylim = c(0,30000), col = c("White", "Blue", "Red"), legend.text=firstblood.qty$Var1)



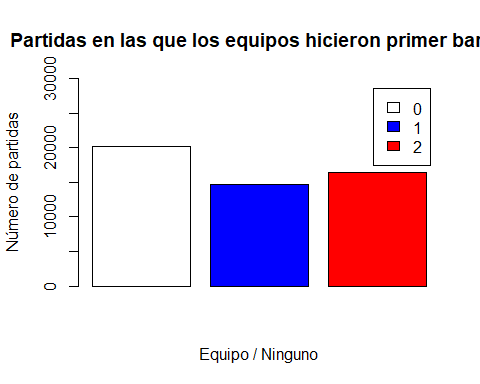
array.firsttower <- c(firsttower.qty[1, 2],firsttower.qty[2, 2], firsttower.qty[3, 2])  
barplot(array.firsttower, main="Partidas en las que los equipos hicieron Primera torre", xlab="Equipo / Ninguno", ylab = "Número de partidas",  
 ylim = c(0,30000), col = c("White", "Blue", "Red"), legend.text=firsttower.qty$Var1)



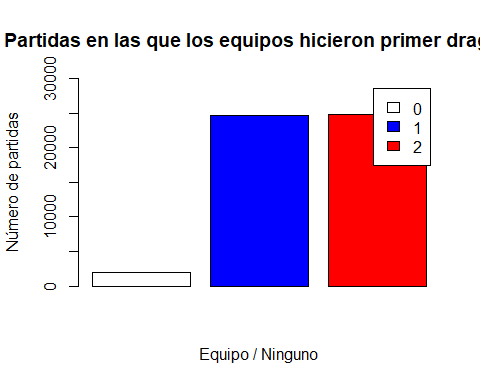
array.firstinhib <- c(firstinhib.qty[1, 2],firstinhib.qty[2, 2], firstinhib.qty[3, 2])  
barplot(array.firstinhib, main="Partidas en las que los equipos hicieron primer inhibidor", xlab="Equipo / Ninguno", ylab = "Número de partidas",  
 ylim = c(0,30000), col = c("White", "Blue", "Red"), legend.text=firstinhib.qty$Var1)



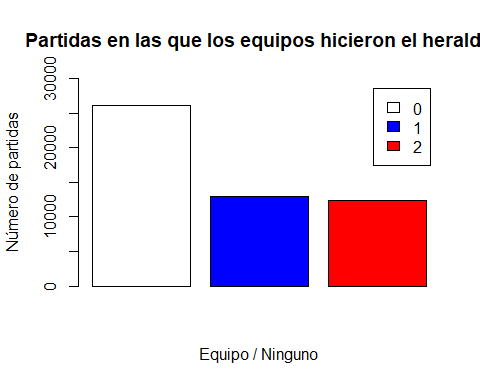
array.firstbaron <- c(firstbaron.qty[1, 2],firstbaron.qty[2, 2], firstbaron.qty[3, 2])  
barplot(array.firstbaron, main="Partidas en las que los equipos hicieron primer barón", xlab="Equipo / Ninguno", ylab = "Número de partidas",   
 ylim = c(0,30000), col = c("White", "Blue", "Red"), legend.text=firstbaron.qty$Var1)



array.firstdragon <- c(firstdragon.qty[1, 2],firstdragon.qty[2, 2], firstdragon.qty[3, 2])  
barplot(array.firstdragon, main="Partidas en las que los equipos hicieron primer dragón", xlab="Equipo / Ninguno", ylab = "Número de partidas",  
 ylim = c(0,30000), col = c("White", "Blue", "Red"), legend.text=firstdragon.qty$Var1)

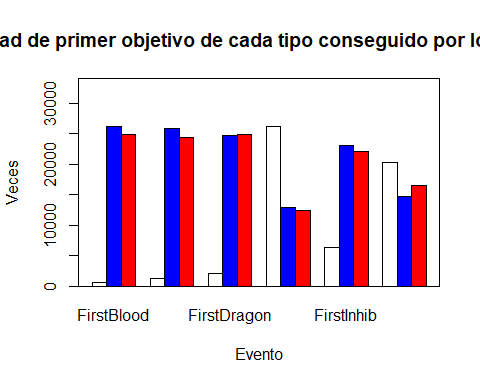


array.firstherald <- c(firstherald.qty[1, 2],firstherald.qty[2, 2], firstherald.qty[3, 2])  
barplot(array.firstherald, main="Partidas en las que los equipos hicieron el heraldo", xlab="Equipo / Ninguno", ylab = "Número de partidas",  
 ylim = c(0,30000), col = c("White", "Blue", "Red"), legend.text=firstherald.qty$Var1)



Si ponemos todos estos en orden cronológico a cuando suelen pasar, obtenemos lo siguiente:

dataframe.firstObjective <- cbind(array.firstblood, array.firsttower, array.firstdragon, array.firstherald, array.firstinhib, array.firstbaron)  
colnames(dataframe.firstObjective) <- c('FirstBlood', 'FirstTower', 'FirstDragon', 'FirstHerald', 'FirstInhib', 'FirstBaron')  
  
colours = c("white","blue","red")  
  
barplot(dataframe.firstObjective, main='Cantidad de primer objetivo de cada tipo conseguido por los equipos',ylab='Veces', xlab='Evento',beside = TRUE,   
 col=colours, ylim=c(0,max(dataframe.firstObjective)\*1.3))  
  
box()



Exportamos a PDF este gráfico…

pdf("../imagenes/Barplot Cronología Partida FO.pdf")  
  
barplot(dataframe.firstObjective, main='Cantidad de primer objetivo de cada tipo conseguido por los equipos',ylab='Veces', xlab='Evento',beside = TRUE,   
 col=colours, ylim=c(0,max(dataframe.firstObjective)\*1.3))  
  
box()  
  
dev.off

## function (which = dev.cur())   
## {  
## if (which == 1)   
## stop("cannot shut down device 1 (the null device)")  
## .External(C\_devoff, as.integer(which))  
## dev.cur()  
## }  
## <bytecode: 0x00000000134d26f0>  
## <environment: namespace:grDevices>

Como podemos ver en los gráficos, la primera sangre y primera torre suelen caer un poco más del lado del equipo azul, siendo estos elementos que se dan al principio de la partida.

Respecto al primer dragón y el heraldo, que se dan en el medio de la partida, se igualan las tornas.

Sin embargo, con los objetivos de final de partida, como es el primer barón, es el equipo rojo quien lo suele conseguir. En cambio, el equipo azul suele conseguir con un poco más de frecuencia el primer inhibidor.

## Machine Learning

### Preparación datos machine learning

#### Dataset total

Ahora usaremos algoritmos de Machine Learning para ver si somos capaces de predecir quién va a ganar en base a una serie de características.

Para hacer esto, vamos a crear un dataset a partir de data.games solo para lo que va a ser el Machine Learning. En él, dejaré las variables que creo necesarias para la predicción de una partida y así usaremos este dataset, tras un centrado y escalado.

Creación del Dataset:

data.games.names <- inner\_join(data.games, data.champs, by = c("t1\_champ1id" = "id"))  
data.games.names <- inner\_join(data.games.names, data.champs, by = c("t1\_champ2id" = "id"))  
data.games.names <- inner\_join(data.games.names, data.champs, by = c("t1\_champ3id" = "id"))  
data.games.names <- inner\_join(data.games.names, data.champs, by = c("t1\_champ4id" = "id"))  
data.games.names <- inner\_join(data.games.names, data.champs, by = c("t1\_champ5id" = "id"))  
data.games.names <- inner\_join(data.games.names, data.champs, by = c("t2\_champ1id" = "id"))  
data.games.names <- inner\_join(data.games.names, data.champs, by = c("t2\_champ2id" = "id"))  
data.games.names <- inner\_join(data.games.names, data.champs, by = c("t2\_champ3id" = "id"))  
data.games.names <- inner\_join(data.games.names, data.champs, by = c("t2\_champ4id" = "id"))  
data.games.names <- inner\_join(data.games.names, data.champs, by = c("t2\_champ5id" = "id"))  
  
champs.names <- data.games.names[,62:71]  
colnames(champs.names) <- c("P1T1", "P2T1", "P3T1", "P4T1", "P5T1", "P1T2", "P2T2", "P3T2", "P4T2", "P5T2")  
  
  
# Ahora paso todos los campeones a variables binarias. Esto hace que para todas las predicciones no tenga que usar distancias ni categorías entre ellos, lo que me permite que sean más justas.  
  
vectors\_champs <- dummy\_cols(champs.names)  
vectors\_champs <- vectors\_champs[, -1:-10]  
  
# Ahora lo único que tengo que hacer es crear el dataset final para el machine learning, quitando aquellas variables que no deseo que estén (entre ellas el código de los campeones) y añadiendo esta codificación binaria que acabo de crear.  
  
data.ml <- data.games[, c(-1, -2, -4, -13, -14, -16, -17, -19, -20, -22, -23, -25, -26, -38, -39, -41, -42, -44, -45, -47, -48)]  
data.ml <- data.ml[, -9:-13]  
data.ml <- data.ml[, -19:-25]  
  
# Ahora hago lo mismo de antes para los nombres pero con los bans. Antes que nada, tengo que añadir None como la elección de ningún campeón para banear en el df de campeones.  
  
df.tonto <- data.frame("None", as.integer(1000))  
names(df.tonto) <- c("name", "id")  
data.champs <- rbind(data.champs, df.tonto)  
  
data.games[data.games=="-1"]<-1000  
  
data.games.bans <- inner\_join(data.games, data.champs, by = c("t1\_ban1" = "id"))  
data.games.bans <- inner\_join(data.games.bans, data.champs, by = c("t1\_ban2" = "id"))  
data.games.bans <- inner\_join(data.games.bans, data.champs, by = c("t1\_ban3" = "id"))  
data.games.bans <- inner\_join(data.games.bans, data.champs, by = c("t1\_ban4" = "id"))  
data.games.bans <- inner\_join(data.games.bans, data.champs, by = c("t1\_ban5" = "id"))  
data.games.bans <- inner\_join(data.games.bans, data.champs, by = c("t2\_ban1" = "id"))  
data.games.bans <- inner\_join(data.games.bans, data.champs, by = c("t2\_ban2" = "id"))  
data.games.bans <- inner\_join(data.games.bans, data.champs, by = c("t2\_ban3" = "id"))  
data.games.bans <- inner\_join(data.games.bans, data.champs, by = c("t2\_ban4" = "id"))  
data.games.bans <- inner\_join(data.games.bans, data.champs, by = c("t2\_ban5" = "id"))  
  
champs.bans.names <- data.games.bans[,62:71]  
colnames(champs.bans.names) <- c("B1T1", "B2T1", "B3T1", "B4T1", "B5T1", "B1T2", "B2T2", "B3T2", "B4T2", "B5T2")  
  
  
vectors\_champs.bans <- dummy\_cols(champs.bans.names)  
vectors\_champs.bans <- vectors\_champs.bans[, -1:-10]  
  
# Finalmente, hago las últimas transformaciones y adiciones a data.ml para obtener el resultado final  
  
data.ml <- data.ml[, -14:-18]  
data.ml <- data.ml[, -19:-23]  
  
data.ml <- cbind(data.ml, vectors\_champs, vectors\_champs.bans)  
  
# Finalmente,hago una segunda matriz sin el resultado, que será la que centre y escale:  
  
data.ml.centscal <- data.ml[, -2]

Ahora hacemos un centrado y escalado de los datos para evitar distorsionar la predicción:

preObjeto <- preProcess(data.ml.centscal, method=c("center", "scale")) # Quiero hacer un centrado y escalado  
data.ml.centscal <- predict(preObjeto, data.ml.centscal)

Estos archivos son grandes, veamos su tamaño:

object\_size(data.ml)

## 575 MB

object\_size(data.ml.centscal)

## 1.15 GB

resultado.pca.ml <- PCA(data.ml.centscal, graph = FALSE)  
  
#Con la siguiente línea podemos ver que podemos hacer con esto calculado  
print(resultado.pca.ml)

## \*\*Results for the Principal Component Analysis (PCA)\*\*  
## The analysis was performed on 51490 individuals, described by 2787 variables  
## \*The results are available in the following objects:  
##   
## name description   
## 1 "$eig" "eigenvalues"   
## 2 "$var" "results for the variables"   
## 3 "$var$coord" "coord. for the variables"   
## 4 "$var$cor" "correlations variables - dimensions"  
## 5 "$var$cos2" "cos2 for the variables"   
## 6 "$var$contrib" "contributions of the variables"   
## 7 "$ind" "results for the individuals"   
## 8 "$ind$coord" "coord. for the individuals"   
## 9 "$ind$cos2" "cos2 for the individuals"   
## 10 "$ind$contrib" "contributions of the individuals"   
## 11 "$call" "summary statistics"   
## 12 "$call$centre" "mean of the variables"   
## 13 "$call$ecart.type" "standard error of the variables"   
## 14 "$call$row.w" "weights for the individuals"   
## 15 "$call$col.w" "weights for the variables"

Nos interesa ver los eigenvalues, que son los que presentarán la cantidad de varianza que aportan las variables:

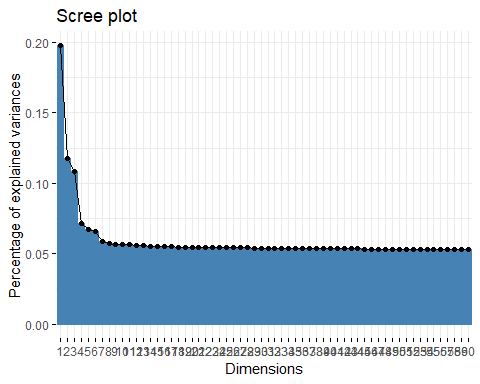
eigenvalues.PCA.ml <- resultado.pca.ml$eig  
head(eigenvalues.PCA.ml)

## eigenvalue percentage of variance cumulative percentage of variance  
## comp 1 5.508868 0.19766300 0.1976630  
## comp 2 3.274071 0.11747655 0.3151395  
## comp 3 3.015903 0.10821324 0.4233528  
## comp 4 1.994136 0.07155134 0.4949041  
## comp 5 1.877788 0.06737667 0.5622808  
## comp 6 1.833950 0.06580373 0.6280845

Como se puede comprobar, de las 24 variables (componentes) que tenemos, la mitad de la varianza la conseguimos con aproximadamente 5 variables. También se puede ver que a parti de las 17 variables prácticamente no hay un aumento de la varianza. En el caso de un problema grande, sería interesante la eliminación de algunas de las variables, para dejar un dataset más pequeño con el que poder trabajar. En nuestro caso, nuestro problema es pequeño, y además las variables están escogidas a mano, por lo que no haré una reducción del dataset.

Ahora, para completar este apartado de PCA, lo que voy a hacer es sacar la gráfica de la varianza acumulada con los valores anteriores:

plotPCA.ml <- fviz\_screeplot(resultado.pca.ml, ncp=60)  
plot(plotPCA.ml)



Exportamos a PDF…

pdf("../imagenes/PCATotal.pdf")  
plot(plotPCA.ml)  
dev.off

## function (which = dev.cur())   
## {  
## if (which == 1)   
## stop("cannot shut down device 1 (the null device)")  
## .External(C\_devoff, as.integer(which))  
## dev.cur()  
## }  
## <bytecode: 0x00000000134d26f0>  
## <environment: namespace:grDevices>

Como vemos, todas las dimensiones van aportando su “granito de arena”, pero ninguna de ellas (según estamos viendo en las 60 primeras) deja de aportar un poco.

Si vamos al dataframe generado en eigenvalues.PCA.ml, vemos que hasta que no llegamos a las 2000 componentes principales no obtenemos al menos un 80% de explicación, mientras que hasta que no llegamos a unas pocas dimensiones del final no obtenemos la explicación completa. Por ello, no reduciré las dimensiones del dataframe, y trabajaremos con él a pesar de que sea más lento y pesado.

#### Dataset campeones

A partir del dataset total, voy a hacer un dataset solo con los campeones (ya codificados), de tal manera que se pueda predecir también solo por ellos:

data.ml.campeones <- data.ml[, 19:2788]  
  
#matriz.sparse.campeones <- Matrix(data.ml.campeones, sparse = T)

Ahora, al igual que antes, calculamos el PCA:

resultado.pca.campeones.ml <- PCA(data.ml.campeones, graph = FALSE)  
  
#Con la siguiente línea podemos ver que podemos hacer con esto calculado  
print(resultado.pca.campeones.ml)

## \*\*Results for the Principal Component Analysis (PCA)\*\*  
## The analysis was performed on 51490 individuals, described by 2770 variables  
## \*The results are available in the following objects:  
##   
## name description   
## 1 "$eig" "eigenvalues"   
## 2 "$var" "results for the variables"   
## 3 "$var$coord" "coord. for the variables"   
## 4 "$var$cor" "correlations variables - dimensions"  
## 5 "$var$cos2" "cos2 for the variables"   
## 6 "$var$contrib" "contributions of the variables"   
## 7 "$ind" "results for the individuals"   
## 8 "$ind$coord" "coord. for the individuals"   
## 9 "$ind$cos2" "cos2 for the individuals"   
## 10 "$ind$contrib" "contributions of the individuals"   
## 11 "$call" "summary statistics"   
## 12 "$call$centre" "mean of the variables"   
## 13 "$call$ecart.type" "standard error of the variables"   
## 14 "$call$row.w" "weights for the individuals"   
## 15 "$call$col.w" "weights for the variables"

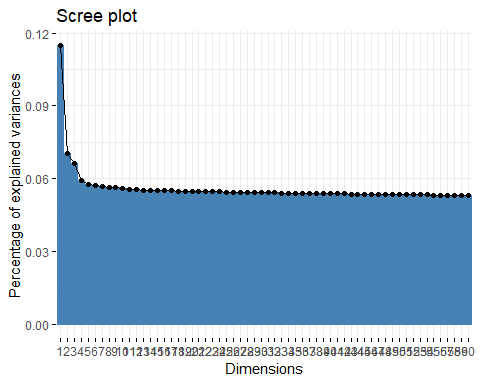
Nos interesa ver los eigenvalues, que son los que presentarán la cantidad de varianza que aportan las variables:

eigenvalues.PCA.campeones.ml <- resultado.pca.campeones.ml$eig  
head(eigenvalues.PCA.campeones.ml)

## eigenvalue percentage of variance cumulative percentage of variance  
## comp 1 3.181314 0.11484890 0.1148489  
## comp 2 1.947903 0.07032139 0.1851703  
## comp 3 1.839567 0.06641037 0.2515807  
## comp 4 1.643525 0.05933303 0.3109137  
## comp 5 1.598470 0.05770651 0.3686202  
## comp 6 1.585067 0.05722264 0.4258428

Ahora, para completar este apartado de PCA, lo que voy a hacer es sacar la gráfica de la varianza acumulada con los valores anteriores:

plotPCA.campeones.ml <- fviz\_screeplot(resultado.pca.campeones.ml, ncp=60)  
plot(plotPCA.campeones.ml)



Exportamos a PDF…

pdf("../imagenes/PCACampones.pdf")  
plot(plotPCA.campeones.ml)  
dev.off

## function (which = dev.cur())   
## {  
## if (which == 1)   
## stop("cannot shut down device 1 (the null device)")  
## .External(C\_devoff, as.integer(which))  
## dev.cur()  
## }  
## <bytecode: 0x00000000134d26f0>  
## <environment: namespace:grDevices>

#### Dataset datos equipos

A partir del dataset total, voy a hacer un dataset solo con las estadísticas de los equipos, de tal manera que se pueda predecir también solo por ellos, creando un dataset más pequeño. Con este dataset corroboraremos que no hacen falta los campeones para predecir el ganador de una partida, sino que con las estadísticas vale:

data.ml.stats.centscal <- data.ml[, 1:18]  
data.ml.stats.centscal <- data.ml.stats.centscal[, -2] #Quitamos el ganador  
  
preObjeto.stats <- preProcess(data.ml.stats.centscal, method=c("center", "scale")) # Quiero hacer un centrado y escalado  
data.ml.stats.centscal <- predict(preObjeto.stats, data.ml.stats.centscal)

Ahora, al igual que antes, calculamos el PCA:

resultado.pca.stats.ml <- PCA(data.ml.stats.centscal, graph = FALSE)  
  
#Con la siguiente línea podemos ver que podemos hacer con esto calculado  
print(resultado.pca.stats.ml)

## \*\*Results for the Principal Component Analysis (PCA)\*\*  
## The analysis was performed on 51490 individuals, described by 17 variables  
## \*The results are available in the following objects:  
##   
## name description   
## 1 "$eig" "eigenvalues"   
## 2 "$var" "results for the variables"   
## 3 "$var$coord" "coord. for the variables"   
## 4 "$var$cor" "correlations variables - dimensions"  
## 5 "$var$cos2" "cos2 for the variables"   
## 6 "$var$contrib" "contributions of the variables"   
## 7 "$ind" "results for the individuals"   
## 8 "$ind$coord" "coord. for the individuals"   
## 9 "$ind$cos2" "cos2 for the individuals"   
## 10 "$ind$contrib" "contributions of the individuals"   
## 11 "$call" "summary statistics"   
## 12 "$call$centre" "mean of the variables"   
## 13 "$call$ecart.type" "standard error of the variables"   
## 14 "$call$row.w" "weights for the individuals"   
## 15 "$call$col.w" "weights for the variables"

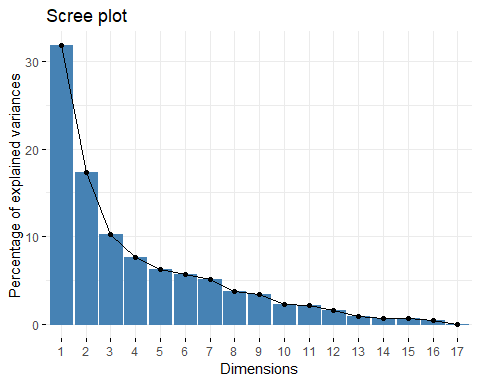
Nos interesa ver los eigenvalues, que son los que presentarán la cantidad de varianza que aportan las variables:

eigenvalues.PCA.stats.ml <- resultado.pca.stats.ml$eig  
head(eigenvalues.PCA.stats.ml)

## eigenvalue percentage of variance cumulative percentage of variance  
## comp 1 5.4055231 31.797194 31.79719  
## comp 2 2.9381181 17.283047 49.08024  
## comp 3 1.7478086 10.281227 59.36147  
## comp 4 1.3032125 7.665956 67.02743  
## comp 5 1.0609948 6.241146 73.26857  
## comp 6 0.9598527 5.646193 78.91476

Ahora, para completar este apartado de PCA, lo que voy a hacer es sacar la gráfica de la varianza acumulada con los valores anteriores:

plotPCA.stats.ml <- fviz\_screeplot(resultado.pca.stats.ml, ncp=60)  
plot(plotPCA.stats.ml)



Exportamos a PDF:

pdf("../imagenes/PCAstatsML.pdf")  
plot(plotPCA.stats.ml)  
dev.off

## function (which = dev.cur())   
## {  
## if (which == 1)   
## stop("cannot shut down device 1 (the null device)")  
## .External(C\_devoff, as.integer(which))  
## dev.cur()  
## }  
## <bytecode: 0x00000000134d26f0>  
## <environment: namespace:grDevices>

### Algoritmos Supervisados

Pasamos ahora a una revisión / uso de algoritmos supervisados:

#### Perceptrón Multicapa

El algoritmo supervisado por el que siempre se debe de empezar es el perceptrón multicapa. Con este algoritmo de clasificación voy a intentar predecir el resultado de la partida. Este algoritmo se basa en una red neuronal con una capa intermedia (en este caso), mediante la cual podemos procesar datos más complejos que con un perceptrón monocapa. Con este algoritmo comprobaremos si las partidas ganadas por unos y por otros son linealmente separables.

Ahora vamos a importar la librería nnet, que nos sirve para hacer perceptrones

Ahora lo que hago es coger un conjunto muy grande de los datos para hacer el entrenamiento

conjuntoEntrenamiento <- sample(1:nrow(data.ml.stats.centscal), 45000)

Creo una tabla para guardar los resultados de todos los tests del perceptrón:

resultados.perceptron <- data.frame(stringsAsFactors=FALSE)

1 NEURONA #################################################### ###############################################################

Lo que voy a hacer ahora es entrenar la red neuronal con diferente cantidad de neuronas,y voy a ir comparando el resultado…

SIN SOFTMAX #############################

# Ahora creamos un dataframe para guardar los resultados de este bucle  
   
 dataframe.resultados.1neu <- data.frame(Ent\_1neu=numeric(),  
 Test\_1neu=numeric())  
  
  
for (i in 1:10)  
{  
   
 partidas.1neu <- nnet( data.ml.stats.centscal[conjuntoEntrenamiento, ], class.ind( data.ml[conjuntoEntrenamiento, 2] ) , size=1, MaxNWts=10000 )  
   
 #Una vez que lo tengo entrenado, lo que voy a hacer es calcular el error tanto en el entrenamiento como en el test de cada uno  
   
 partidas.prediccion.1neu <- predict( partidas.1neu, data.ml.stats.centscal[conjuntoEntrenamiento, ], type="raw" )  
 head(partidas.prediccion.1neu) # Vemos las probabilidades de pertenencia de cada valor  
   
 # Ahora que los tengo todos entrenados, Determinamos cual es la máxima, es decir, la clase a la que hay que asignar los objetos  
   
 partidas.prediccion.1neu.class <- apply( partidas.prediccion.1neu, MARGIN=1, FUN='which.is.max')  
   
 #Calculo el acierto  
   
 acierto\_ent\_1neu<- sum( diag( table( partidas.prediccion.1neu.class, data.ml[conjuntoEntrenamiento, 2] ) ) )/length(conjuntoEntrenamiento)  
   
 ###### TEST  
   
   
 partidas.prediccion.test.1neu <- predict( partidas.1neu, data.ml.stats.centscal[-conjuntoEntrenamiento, ], type="raw" )  
  
 partidas.prediccion.test.1neu.class <- apply( partidas.prediccion.test.1neu, MARGIN=1, FUN='which.is.max')  
  
 table( partidas.prediccion.test.1neu.class , data.ml[-conjuntoEntrenamiento, 2] )  
   
 acierto\_test\_1neu <- sum( diag( table( partidas.prediccion.test.1neu.class, data.ml[-conjuntoEntrenamiento, 2] ) ) )/(nrow(data.ml.stats.centscal) - length(conjuntoEntrenamiento))  
  
   
 dataframe.pasada <- data.frame(Ent\_1neu = acierto\_ent\_1neu,  
 Test\_1neu= acierto\_test\_1neu)  
   
 dataframe.resultados.1neu <- rbind(dataframe.resultados.1neu, dataframe.pasada)  
   
}

## # weights: 22  
## initial value 22624.058812   
## final value 6181.000000   
## converged  
## # weights: 22  
## initial value 24270.958565   
## final value 6890.000000   
## converged  
## # weights: 22  
## initial value 23532.782814   
## iter 10 value 16588.367589  
## iter 20 value 4608.407544  
## iter 30 value 3136.465372  
## iter 40 value 2893.760131  
## iter 50 value 2777.601658  
## iter 60 value 2764.962892  
## iter 70 value 2756.787408  
## iter 80 value 2732.915189  
## iter 90 value 2606.878365  
## iter 100 value 2536.009578  
## final value 2536.009578   
## stopped after 100 iterations  
## # weights: 22  
## initial value 24916.275908   
## final value 7010.000000   
## converged  
## # weights: 22  
## initial value 24712.259295   
## final value 8022.000000   
## converged  
## # weights: 22  
## initial value 23357.287747   
## final value 6740.000000   
## converged  
## # weights: 22  
## initial value 25238.498469   
## iter 10 value 20844.990310  
## iter 20 value 14239.853032  
## iter 30 value 11736.114364  
## iter 40 value 11735.929529  
## final value 11735.925198   
## converged  
## # weights: 22  
## initial value 23843.508920   
## iter 10 value 21222.773317  
## iter 20 value 19094.370629  
## iter 30 value 4045.096017  
## iter 40 value 2908.033698  
## iter 50 value 2768.815312  
## iter 60 value 2610.484101  
## iter 70 value 2443.713085  
## iter 80 value 2395.373141  
## iter 90 value 2393.512079  
## iter 100 value 2388.370119  
## final value 2388.370119   
## stopped after 100 iterations  
## # weights: 22  
## initial value 26752.269442   
## iter 10 value 17557.094154  
## iter 20 value 10372.644981  
## iter 30 value 3616.429771  
## iter 40 value 2864.522622  
## iter 50 value 2776.189433  
## iter 60 value 2752.020118  
## iter 70 value 2553.863206  
## iter 80 value 2429.271913  
## iter 90 value 2400.099475  
## iter 100 value 2397.277640  
## final value 2397.277640   
## stopped after 100 iterations  
## # weights: 22  
## initial value 22273.335621   
## iter 10 value 4796.182643  
## iter 20 value 3975.519101  
## iter 30 value 2945.847171  
## iter 40 value 2884.143936  
## iter 50 value 2853.314269  
## iter 60 value 2817.775676  
## iter 70 value 2612.629596  
## iter 80 value 2454.271142  
## iter 90 value 2394.048463  
## iter 100 value 2393.540179  
## final value 2393.540179   
## stopped after 100 iterations

Ahora le vamos a añadir softmax, a ver si mejoramos…

CON SOFTMAX #############################

dataframe.resultados.1neu.soft <- data.frame(Ent\_1neu\_soft=numeric(),  
 Test\_1neu\_soft=numeric())  
  
for (i in 1:10)  
{  
   
 partidas.1neu.soft <- nnet( data.ml.stats.centscal[conjuntoEntrenamiento, ], class.ind( data.ml[conjuntoEntrenamiento, 2] ) , size=1, MaxNWts=10000, softmax = T )  
   
 #Una vez que lo tengo entrenado, lo que voy a hacer es calcular el error tanto en el entrenamiento como en el test de cada uno  
   
 partidas.prediccion.1neu.soft <- predict( partidas.1neu.soft, data.ml.stats.centscal[conjuntoEntrenamiento, ], type="raw" )  
 head(partidas.prediccion.1neu.soft) # Vemos las probabilidades de pertenencia de cada valor  
   
 # Ahora que los tengo todos entrenados, Determinamos cual es la máxima, es decir, la clase a la que hay que asignar los objetos  
   
 partidas.prediccion.1neu.class.soft <- apply( partidas.prediccion.1neu.soft, MARGIN=1, FUN='which.is.max')  
   
 #Calculo el acierto  
   
 acierto\_ent\_1neu.soft <- sum( diag( table( partidas.prediccion.1neu.class.soft, data.ml[conjuntoEntrenamiento, 2] ) ) )/length(conjuntoEntrenamiento)  
   
 ###### TEST  
   
   
 partidas.prediccion.test.1neu.soft <- predict( partidas.1neu.soft, data.ml.stats.centscal[-conjuntoEntrenamiento, ], type="raw" )  
  
 partidas.prediccion.test.1neu.class.soft <- apply( partidas.prediccion.test.1neu.soft, MARGIN=1, FUN='which.is.max')  
  
 table( partidas.prediccion.test.1neu.class.soft , data.ml[-conjuntoEntrenamiento, 2] )  
   
 acierto\_test\_1neu.soft <- sum( diag( table( partidas.prediccion.test.1neu.class.soft, data.ml[-conjuntoEntrenamiento, 2] ) ) )/(nrow(data.ml.stats.centscal) - length(conjuntoEntrenamiento))  
  
   
 dataframe.pasada <- data.frame(Ent\_1neu\_soft = acierto\_ent\_1neu.soft,  
 Test\_1neu\_soft= acierto\_test\_1neu.soft)  
   
 dataframe.resultados.1neu.soft <- rbind(dataframe.resultados.1neu.soft, dataframe.pasada)  
   
}

## # weights: 22  
## initial value 31816.726187   
## iter 10 value 14475.967039  
## iter 20 value 10690.367838  
## iter 30 value 6393.831649  
## iter 40 value 4405.682130  
## iter 50 value 4155.222079  
## iter 60 value 4080.862234  
## iter 70 value 4019.847012  
## iter 80 value 4008.207516  
## iter 90 value 4006.209295  
## iter 100 value 4006.006107  
## final value 4006.006107   
## stopped after 100 iterations  
## # weights: 22  
## initial value 33822.029691   
## iter 10 value 10268.302538  
## iter 20 value 8085.934507  
## iter 30 value 6011.807646  
## iter 40 value 5247.854990  
## iter 50 value 4744.767807  
## iter 60 value 4615.021222  
## iter 70 value 4110.074684  
## iter 80 value 4034.643788  
## iter 90 value 4014.032889  
## iter 100 value 4012.078811  
## final value 4012.078811   
## stopped after 100 iterations  
## # weights: 22  
## initial value 36113.082520   
## iter 10 value 12881.482872  
## iter 20 value 10857.149555  
## iter 30 value 4136.930195  
## iter 40 value 4066.831087  
## iter 50 value 4018.930820  
## iter 60 value 4016.842523  
## iter 70 value 4008.232813  
## iter 80 value 4005.348458  
## iter 90 value 4004.963521  
## iter 100 value 4004.937929  
## final value 4004.937929   
## stopped after 100 iterations  
## # weights: 22  
## initial value 32588.945650   
## iter 10 value 10056.351410  
## iter 20 value 8016.455666  
## iter 30 value 5892.037237  
## iter 40 value 5803.419400  
## iter 50 value 5760.425997  
## iter 60 value 5601.676456  
## iter 70 value 5132.525548  
## iter 80 value 4298.783069  
## iter 90 value 4057.524728  
## iter 100 value 4043.319847  
## final value 4043.319847   
## stopped after 100 iterations  
## # weights: 22  
## initial value 31381.334309   
## iter 10 value 13524.383510  
## iter 20 value 9672.134108  
## iter 30 value 6467.355767  
## iter 40 value 5501.950444  
## iter 50 value 5267.530332  
## iter 60 value 4970.251349  
## iter 70 value 4389.745900  
## iter 80 value 4060.968240  
## iter 90 value 4016.709834  
## iter 100 value 4014.390713  
## final value 4014.390713   
## stopped after 100 iterations  
## # weights: 22  
## initial value 31001.651193   
## iter 10 value 10667.094105  
## iter 20 value 6733.643477  
## iter 30 value 6229.668642  
## iter 40 value 5652.452048  
## iter 50 value 5404.264403  
## iter 60 value 4693.037584  
## iter 70 value 4182.563306  
## iter 80 value 4039.298820  
## iter 90 value 4035.888167  
## iter 100 value 4016.224163  
## final value 4016.224163   
## stopped after 100 iterations  
## # weights: 22  
## initial value 31792.927376   
## iter 10 value 24634.772383  
## iter 20 value 16258.448683  
## iter 30 value 6115.386545  
## iter 40 value 5491.258644  
## iter 50 value 5133.954578  
## iter 60 value 4329.046186  
## iter 70 value 4078.394846  
## iter 80 value 4045.244304  
## iter 90 value 4037.229347  
## iter 100 value 4010.007029  
## final value 4010.007029   
## stopped after 100 iterations  
## # weights: 22  
## initial value 30324.597367   
## iter 10 value 7947.166016  
## iter 20 value 6042.995199  
## iter 30 value 4261.063204  
## iter 40 value 4132.137119  
## iter 50 value 4051.628936  
## iter 60 value 4040.288816  
## iter 70 value 4009.016590  
## iter 80 value 4005.828952  
## iter 90 value 4005.253005  
## iter 100 value 4005.217664  
## final value 4005.217664   
## stopped after 100 iterations  
## # weights: 22  
## initial value 36972.728209   
## iter 10 value 10874.138232  
## iter 20 value 7731.297850  
## iter 30 value 7190.948647  
## iter 40 value 7133.630057  
## iter 50 value 7128.365499  
## iter 60 value 7128.260841  
## iter 70 value 7128.203107  
## iter 80 value 7127.725915  
## iter 90 value 7127.624189  
## iter 100 value 6196.886284  
## final value 6196.886284   
## stopped after 100 iterations  
## # weights: 22  
## initial value 44488.986254   
## iter 10 value 26886.948792  
## iter 20 value 15124.538771  
## iter 30 value 13105.825508  
## iter 40 value 13039.503327  
## iter 50 value 13015.929130  
## iter 60 value 13013.954374  
## final value 13013.570178   
## converged

2 NEURONAS ################################################### ###############################################################

SIN SOFTMAX #############################

dataframe.resultados.2neu <- data.frame(Ent\_2neu=numeric(),  
 Test\_2neu=numeric())  
  
  
for (i in 1:10)  
{  
   
 partidas.2neu <- nnet( data.ml.stats.centscal[conjuntoEntrenamiento, ], class.ind( data.ml[conjuntoEntrenamiento, 2] ) , size=2, MaxNWts=10000 )  
   
 #Una vez que lo tengo entrenado, lo que voy a hacer es calcular el error tanto en el entrenamiento como en el test de cada uno  
   
 partidas.prediccion.2neu <- predict( partidas.2neu, data.ml.stats.centscal[conjuntoEntrenamiento, ], type="raw" )  
 head(partidas.prediccion.2neu) # Vemos las probabilidades de pertenencia de cada valor  
   
 # Ahora que los tengo todos entrenados, Determinamos cual es la máxima, es decir, la clase a la que hay que asignar los objetos  
   
 partidas.prediccion.2neu.class <- apply( partidas.prediccion.2neu, MARGIN=1, FUN='which.is.max')  
   
 #Calculo el acierto  
   
 acierto\_ent\_2neu<- sum( diag( table( partidas.prediccion.2neu.class, data.ml[conjuntoEntrenamiento, 2] ) ) )/length(conjuntoEntrenamiento)  
   
 ###### TEST  
   
   
 partidas.prediccion.test.2neu <- predict( partidas.2neu, data.ml.stats.centscal[-conjuntoEntrenamiento, ], type="raw" )  
  
 partidas.prediccion.test.2neu.class <- apply( partidas.prediccion.test.2neu, MARGIN=1, FUN='which.is.max')  
  
 table( partidas.prediccion.test.2neu.class , data.ml[-conjuntoEntrenamiento, 2] )  
   
 acierto\_test\_2neu <- sum( diag( table( partidas.prediccion.test.2neu.class, data.ml[-conjuntoEntrenamiento, 2] ) ) )/(nrow(data.ml.stats.centscal) - length(conjuntoEntrenamiento))  
  
   
 dataframe.pasada <- data.frame(Ent\_2neu = acierto\_ent\_2neu,  
 Test\_2neu= acierto\_test\_2neu)  
   
 dataframe.resultados.2neu <- rbind(dataframe.resultados.2neu, dataframe.pasada)  
   
}

## # weights: 42  
## initial value 25840.155275   
## iter 10 value 3462.236440  
## iter 20 value 2971.302975  
## iter 30 value 2840.223535  
## iter 40 value 2623.760746  
## iter 50 value 2431.622084  
## iter 60 value 2399.762494  
## iter 70 value 2332.511987  
## iter 80 value 2327.238905  
## iter 90 value 2325.564707  
## iter 100 value 2272.367680  
## final value 2272.367680   
## stopped after 100 iterations  
## # weights: 42  
## initial value 24410.882799   
## final value 6953.000000   
## converged  
## # weights: 42  
## initial value 23305.646010   
## final value 6428.000000   
## converged  
## # weights: 42  
## initial value 24814.238623   
## iter 10 value 3417.619475  
## iter 20 value 3099.374072  
## iter 30 value 2915.994053  
## iter 40 value 2775.721252  
## iter 50 value 2666.024718  
## iter 60 value 2618.403962  
## iter 70 value 2558.330571  
## iter 80 value 2447.234402  
## iter 90 value 2217.092527  
## iter 100 value 2153.080556  
## final value 2153.080556   
## stopped after 100 iterations  
## # weights: 42  
## initial value 25995.675747   
## final value 6734.000000   
## converged  
## # weights: 42  
## initial value 24295.624589   
## final value 5626.000000   
## converged  
## # weights: 42  
## initial value 25130.035474   
## final value 7086.000000   
## converged  
## # weights: 42  
## initial value 24346.296602   
## final value 8398.000000   
## converged  
## # weights: 42  
## initial value 22369.187385   
## final value 6094.000000   
## converged  
## # weights: 42  
## initial value 22358.481312   
## final value 7580.000000   
## converged

Ahora le vamos a añadir softmax, a ver si mejoramos…

CON SOFTMAX #############################

dataframe.resultados.2neu.soft <- data.frame(Ent\_2neu\_soft=numeric(),  
 Test\_2neu\_soft=numeric())  
  
for (i in 1:10)  
{  
   
 partidas.2neu.soft <- nnet( data.ml.stats.centscal[conjuntoEntrenamiento, ], class.ind( data.ml[conjuntoEntrenamiento, 2] ) , size=2, MaxNWts=10000, softmax = T )  
   
 #Una vez que lo tengo entrenado, lo que voy a hacer es calcular el error tanto en el entrenamiento como en el test de cada uno  
   
 partidas.prediccion.2neu.soft <- predict( partidas.2neu.soft, data.ml.stats.centscal[conjuntoEntrenamiento, ], type="raw" )  
 head(partidas.prediccion.2neu.soft) # Vemos las probabilidades de pertenencia de cada valor  
   
 # Ahora que los tengo todos entrenados, Determinamos cual es la máxima, es decir, la clase a la que hay que asignar los objetos  
   
 partidas.prediccion.2neu.class.soft <- apply( partidas.prediccion.2neu.soft, MARGIN=1, FUN='which.is.max')  
   
 #Calculo el acierto  
   
 acierto\_ent\_2neu.soft <- sum( diag( table( partidas.prediccion.2neu.class.soft, data.ml[conjuntoEntrenamiento, 2] ) ) )/length(conjuntoEntrenamiento)  
   
 ###### TEST  
   
   
 partidas.prediccion.test.2neu.soft <- predict( partidas.2neu.soft, data.ml.stats.centscal[-conjuntoEntrenamiento, ], type="raw" )  
  
 partidas.prediccion.test.2neu.class.soft <- apply( partidas.prediccion.test.2neu.soft, MARGIN=1, FUN='which.is.max')  
  
 table( partidas.prediccion.test.2neu.class.soft , data.ml[-conjuntoEntrenamiento, 2] )  
   
 acierto\_test\_2neu.soft <- sum( diag( table( partidas.prediccion.test.2neu.class.soft, data.ml[-conjuntoEntrenamiento, 2] ) ) )/(nrow(data.ml.stats.centscal) - length(conjuntoEntrenamiento))  
  
   
 dataframe.pasada <- data.frame(Ent\_2neu\_soft = acierto\_ent\_2neu.soft,  
 Test\_2neu\_soft= acierto\_test\_2neu.soft)  
   
 dataframe.resultados.2neu.soft <- rbind(dataframe.resultados.2neu.soft, dataframe.pasada)  
   
}

## # weights: 42  
## initial value 35604.385568   
## iter 10 value 7797.312783  
## iter 20 value 5489.874705  
## iter 30 value 4600.353133  
## iter 40 value 3858.241601  
## iter 50 value 3718.072566  
## iter 60 value 3601.936007  
## iter 70 value 3419.934162  
## iter 80 value 3351.017232  
## iter 90 value 3304.805955  
## iter 100 value 3264.055832  
## final value 3264.055832   
## stopped after 100 iterations  
## # weights: 42  
## initial value 40089.329036   
## iter 10 value 7464.682074  
## iter 20 value 4899.671253  
## iter 30 value 3817.568606  
## iter 40 value 3694.252237  
## iter 50 value 3641.872475  
## iter 60 value 3624.033958  
## iter 70 value 3608.045789  
## iter 80 value 3580.369062  
## iter 90 value 3523.513787  
## iter 100 value 3501.173999  
## final value 3501.173999   
## stopped after 100 iterations  
## # weights: 42  
## initial value 33335.693682   
## iter 10 value 10351.140745  
## iter 20 value 7091.200000  
## iter 30 value 4335.461190  
## iter 40 value 3787.079278  
## iter 50 value 3540.812141  
## iter 60 value 3407.405179  
## iter 70 value 3241.111270  
## iter 80 value 3201.161583  
## iter 90 value 3185.168782  
## iter 100 value 3150.300766  
## final value 3150.300766   
## stopped after 100 iterations  
## # weights: 42  
## initial value 30535.698127   
## iter 10 value 5520.475183  
## iter 20 value 4221.312998  
## iter 30 value 3960.053049  
## iter 40 value 3826.261310  
## iter 50 value 3501.797552  
## iter 60 value 3325.549701  
## iter 70 value 3214.095206  
## iter 80 value 3190.431762  
## iter 90 value 3160.363801  
## iter 100 value 3157.437492  
## final value 3157.437492   
## stopped after 100 iterations  
## # weights: 42  
## initial value 32916.433268   
## iter 10 value 5909.153652  
## iter 20 value 5435.668208  
## iter 30 value 4919.805738  
## iter 40 value 3971.840421  
## iter 50 value 3490.495592  
## iter 60 value 3336.246843  
## iter 70 value 3303.083843  
## iter 80 value 3298.597854  
## iter 90 value 3297.486332  
## iter 100 value 3294.903849  
## final value 3294.903849   
## stopped after 100 iterations  
## # weights: 42  
## initial value 30538.171685   
## iter 10 value 6985.931956  
## iter 20 value 5861.761104  
## iter 30 value 5123.162615  
## iter 40 value 4434.213353  
## iter 50 value 4014.854854  
## iter 60 value 3871.153838  
## iter 70 value 3762.017066  
## iter 80 value 3596.754797  
## iter 90 value 3408.249413  
## iter 100 value 3400.812859  
## final value 3400.812859   
## stopped after 100 iterations  
## # weights: 42  
## initial value 35823.723028   
## iter 10 value 7485.770023  
## iter 20 value 6279.273730  
## iter 30 value 5671.718186  
## iter 40 value 5515.298405  
## iter 50 value 5039.290622  
## iter 60 value 4041.590538  
## iter 70 value 3917.048914  
## iter 80 value 3735.395832  
## iter 90 value 3634.636187  
## iter 100 value 3585.316572  
## final value 3585.316572   
## stopped after 100 iterations  
## # weights: 42  
## initial value 41867.338103   
## iter 10 value 5394.414477  
## iter 20 value 5021.610778  
## iter 30 value 4386.463230  
## iter 40 value 4150.124830  
## iter 50 value 3735.517136  
## iter 60 value 3296.987658  
## iter 70 value 3237.298387  
## iter 80 value 3225.560874  
## iter 90 value 3221.499831  
## iter 100 value 3219.043576  
## final value 3219.043576   
## stopped after 100 iterations  
## # weights: 42  
## initial value 33021.086483   
## iter 10 value 11951.794569  
## iter 20 value 5951.778205  
## iter 30 value 4289.164513  
## iter 40 value 3888.187929  
## iter 50 value 3630.558637  
## iter 60 value 3588.242435  
## iter 70 value 3520.729023  
## iter 80 value 3345.856865  
## iter 90 value 3237.972547  
## iter 100 value 3203.517510  
## final value 3203.517510   
## stopped after 100 iterations  
## # weights: 42  
## initial value 29993.757351   
## iter 10 value 5089.251255  
## iter 20 value 4638.809249  
## iter 30 value 4012.342598  
## iter 40 value 3581.140725  
## iter 50 value 3255.324719  
## iter 60 value 3219.136036  
## iter 70 value 3213.213024  
## iter 80 value 3200.152059  
## iter 90 value 3166.323502  
## iter 100 value 3161.552976  
## final value 3161.552976   
## stopped after 100 iterations

3 NEURONAS ################################################### ###############################################################

SIN SOFTMAX #############################

dataframe.resultados.3neu <- data.frame(Ent\_3neu=numeric(),  
 Test\_3neu=numeric())  
  
  
for (i in 1:10)  
{  
   
 partidas.3neu <- nnet( data.ml.stats.centscal[conjuntoEntrenamiento, ], class.ind( data.ml[conjuntoEntrenamiento, 2] ) , size=3, MaxNWts=10000 )  
   
 #Una vez que lo tengo entrenado, lo que voy a hacer es calcular el error tanto en el entrenamiento como en el test de cada uno  
   
 partidas.prediccion.3neu <- predict( partidas.2neu, data.ml.stats.centscal[conjuntoEntrenamiento, ], type="raw" )  
 head(partidas.prediccion.3neu) # Vemos las probabilidades de pertenencia de cada valor  
   
 # Ahora que los tengo todos entrenados, Determinamos cual es la máxima, es decir, la clase a la que hay que asignar los objetos  
   
 partidas.prediccion.3neu.class <- apply( partidas.prediccion.3neu, MARGIN=1, FUN='which.is.max')  
   
 #Calculo el acierto  
   
 acierto\_ent\_3neu<- sum( diag( table( partidas.prediccion.3neu.class, data.ml[conjuntoEntrenamiento, 2] ) ) )/length(conjuntoEntrenamiento)  
   
 ###### TEST  
   
   
 partidas.prediccion.test.3neu <- predict( partidas.3neu, data.ml.stats.centscal[-conjuntoEntrenamiento, ], type="raw" )  
  
 partidas.prediccion.test.3neu.class <- apply( partidas.prediccion.test.3neu, MARGIN=1, FUN='which.is.max')  
  
 table( partidas.prediccion.test.3neu.class , data.ml[-conjuntoEntrenamiento, 2] )  
   
 acierto\_test\_3neu <- sum( diag( table( partidas.prediccion.test.3neu.class, data.ml[-conjuntoEntrenamiento, 2] ) ) )/(nrow(data.ml.stats.centscal) - length(conjuntoEntrenamiento))  
  
   
 dataframe.pasada <- data.frame(Ent\_3neu = acierto\_ent\_3neu,  
 Test\_3neu= acierto\_test\_3neu)  
   
 dataframe.resultados.3neu <- rbind(dataframe.resultados.3neu, dataframe.pasada)  
   
}

## # weights: 62  
## initial value 26828.271214   
## final value 6561.000000   
## converged  
## # weights: 62  
## initial value 22710.511207   
## final value 6332.000000   
## converged  
## # weights: 62  
## initial value 24379.179027   
## final value 8827.000000   
## converged  
## # weights: 62  
## initial value 25541.481209   
## iter 10 value 4619.181129  
## iter 20 value 3174.003482  
## iter 30 value 2917.952079  
## iter 40 value 2838.391520  
## iter 50 value 2768.978140  
## iter 60 value 2738.194913  
## iter 70 value 2725.334110  
## iter 80 value 2714.512227  
## iter 90 value 2703.141973  
## iter 100 value 2655.979968  
## final value 2655.979968   
## stopped after 100 iterations  
## # weights: 62  
## initial value 22170.705849   
## final value 8972.000000   
## converged  
## # weights: 62  
## initial value 26709.960284   
## final value 6654.000000   
## converged  
## # weights: 62  
## initial value 24253.079436   
## iter 10 value 3320.514031  
## iter 20 value 2579.013343  
## iter 30 value 2360.935981  
## iter 40 value 2190.891766  
## iter 50 value 2050.066856  
## iter 60 value 1912.612185  
## iter 70 value 1788.640152  
## iter 80 value 1779.507364  
## iter 90 value 1766.816060  
## iter 100 value 1740.566980  
## final value 1740.566980   
## stopped after 100 iterations  
## # weights: 62  
## initial value 26042.886570   
## final value 7092.000000   
## converged  
## # weights: 62  
## initial value 23050.440340   
## final value 7348.998780   
## converged  
## # weights: 62  
## initial value 21612.762762   
## final value 8415.000000   
## converged

Ahora le vamos a añadir softmax, a ver si mejoramos…

CON SOFTMAX #############################

dataframe.resultados.3neu.soft <- data.frame(Ent\_3neu\_soft=numeric(),  
 Test\_3neu\_soft=numeric())  
  
for (i in 1:10)  
{  
   
 partidas.3neu.soft <- nnet( data.ml.stats.centscal[conjuntoEntrenamiento, ], class.ind( data.ml[conjuntoEntrenamiento, 2] ) , size=3, MaxNWts=10000, softmax = T )  
   
 #Una vez que lo tengo entrenado, lo que voy a hacer es calcular el error tanto en el entrenamiento como en el test de cada uno  
   
 partidas.prediccion.3neu.soft <- predict( partidas.3neu.soft, data.ml.stats.centscal[conjuntoEntrenamiento, ], type="raw" )  
 head(partidas.prediccion.3neu.soft) # Vemos las probabilidades de pertenencia de cada valor  
   
 # Ahora que los tengo todos entrenados, Determinamos cual es la máxima, es decir, la clase a la que hay que asignar los objetos  
   
 partidas.prediccion.3neu.class.soft <- apply( partidas.prediccion.3neu.soft, MARGIN=1, FUN='which.is.max')  
   
 #Calculo el acierto  
   
 acierto\_ent\_3neu.soft <- sum( diag( table( partidas.prediccion.3neu.class.soft, data.ml[conjuntoEntrenamiento, 2] ) ) )/length(conjuntoEntrenamiento)  
   
 ###### TEST  
   
   
 partidas.prediccion.test.3neu.soft <- predict( partidas.3neu.soft, data.ml.stats.centscal[-conjuntoEntrenamiento, ], type="raw" )  
  
 partidas.prediccion.test.3neu.class.soft <- apply( partidas.prediccion.test.3neu.soft, MARGIN=1, FUN='which.is.max')  
  
 table( partidas.prediccion.test.3neu.class.soft , data.ml[-conjuntoEntrenamiento, 2] )  
   
 acierto\_test\_3neu.soft <- sum( diag( table( partidas.prediccion.test.3neu.class.soft, data.ml[-conjuntoEntrenamiento, 2] ) ) )/(nrow(data.ml.stats.centscal) - length(conjuntoEntrenamiento))  
  
   
 dataframe.pasada <- data.frame(Ent\_3neu\_soft = acierto\_ent\_3neu.soft,  
 Test\_3neu\_soft= acierto\_test\_3neu.soft)  
   
 dataframe.resultados.3neu.soft <- rbind(dataframe.resultados.3neu.soft, dataframe.pasada)  
   
}

## # weights: 62  
## initial value 26959.089071   
## iter 10 value 8017.722062  
## iter 20 value 5247.084761  
## iter 30 value 4480.312728  
## iter 40 value 3738.448424  
## iter 50 value 3220.183428  
## iter 60 value 3138.615727  
## iter 70 value 3114.822323  
## iter 80 value 3058.338077  
## iter 90 value 3022.726137  
## iter 100 value 3010.491971  
## final value 3010.491971   
## stopped after 100 iterations  
## # weights: 62  
## initial value 35104.216370   
## iter 10 value 6483.391417  
## iter 20 value 4890.381725  
## iter 30 value 3968.377257  
## iter 40 value 3601.383196  
## iter 50 value 3508.332126  
## iter 60 value 3475.916504  
## iter 70 value 3391.964488  
## iter 80 value 3261.791968  
## iter 90 value 3223.394021  
## iter 100 value 3205.021132  
## final value 3205.021132   
## stopped after 100 iterations  
## # weights: 62  
## initial value 38819.214037   
## iter 10 value 5697.972041  
## iter 20 value 4619.666363  
## iter 30 value 3973.983565  
## iter 40 value 3583.608368  
## iter 50 value 3436.358579  
## iter 60 value 3328.733748  
## iter 70 value 3267.598449  
## iter 80 value 3193.519942  
## iter 90 value 3108.954612  
## iter 100 value 2980.679935  
## final value 2980.679935   
## stopped after 100 iterations  
## # weights: 62  
## initial value 32039.077781   
## iter 10 value 8111.232504  
## iter 20 value 5420.802081  
## iter 30 value 4498.438231  
## iter 40 value 4148.657430  
## iter 50 value 3878.581177  
## iter 60 value 3401.912300  
## iter 70 value 3227.973492  
## iter 80 value 3204.279078  
## iter 90 value 3184.520300  
## iter 100 value 3173.132897  
## final value 3173.132897   
## stopped after 100 iterations  
## # weights: 62  
## initial value 37445.206609   
## iter 10 value 5429.092585  
## iter 20 value 3764.653607  
## iter 30 value 3526.272064  
## iter 40 value 3306.733051  
## iter 50 value 3190.179832  
## iter 60 value 3100.887719  
## iter 70 value 3046.700525  
## iter 80 value 3023.441888  
## iter 90 value 3005.822905  
## iter 100 value 2998.888772  
## final value 2998.888772   
## stopped after 100 iterations  
## # weights: 62  
## initial value 31287.984196   
## iter 10 value 6079.603646  
## iter 20 value 4570.149188  
## iter 30 value 4181.983314  
## iter 40 value 3598.110568  
## iter 50 value 3421.396713  
## iter 60 value 3286.008783  
## iter 70 value 3240.871874  
## iter 80 value 3218.348962  
## iter 90 value 3204.576101  
## iter 100 value 3175.740184  
## final value 3175.740184   
## stopped after 100 iterations  
## # weights: 62  
## initial value 26794.949255   
## iter 10 value 5598.110012  
## iter 20 value 4965.015483  
## iter 30 value 4518.894922  
## iter 40 value 3807.394313  
## iter 50 value 3378.427630  
## iter 60 value 3213.194834  
## iter 70 value 3142.592053  
## iter 80 value 3118.792102  
## iter 90 value 3086.087949  
## iter 100 value 3063.909032  
## final value 3063.909032   
## stopped after 100 iterations  
## # weights: 62  
## initial value 43440.447467   
## iter 10 value 6455.955135  
## iter 20 value 4470.867065  
## iter 30 value 3853.774463  
## iter 40 value 3486.695031  
## iter 50 value 3340.582315  
## iter 60 value 3268.582376  
## iter 70 value 3214.993962  
## iter 80 value 3172.211745  
## iter 90 value 3093.484534  
## iter 100 value 3079.523242  
## final value 3079.523242   
## stopped after 100 iterations  
## # weights: 62  
## initial value 31239.769360   
## iter 10 value 4538.710585  
## iter 20 value 4098.457931  
## iter 30 value 3893.657551  
## iter 40 value 3639.215430  
## iter 50 value 3477.540030  
## iter 60 value 3300.690772  
## iter 70 value 3242.672931  
## iter 80 value 3223.189747  
## iter 90 value 3217.711392  
## iter 100 value 3193.486925  
## final value 3193.486925   
## stopped after 100 iterations  
## # weights: 62  
## initial value 31241.236305   
## iter 10 value 6771.217635  
## iter 20 value 5290.205634  
## iter 30 value 3872.584140  
## iter 40 value 3542.170314  
## iter 50 value 3353.494019  
## iter 60 value 3184.756445  
## iter 70 value 3120.444182  
## iter 80 value 3007.669120  
## iter 90 value 2951.047729  
## iter 100 value 2936.044283  
## final value 2936.044283   
## stopped after 100 iterations

Ahora juntamos todos los resultados:

dataframe.resultados.perceptron.stats <- cbind(dataframe.resultados.1neu,   
 dataframe.resultados.1neu.soft,   
 dataframe.resultados.2neu,  
 dataframe.resultados.2neu.soft,  
 dataframe.resultados.3neu,  
 dataframe.resultados.3neu.soft)  
  
remove(dataframe.resultados.1neu)  
remove(dataframe.resultados.1neu.soft)  
remove(dataframe.resultados.2neu)  
remove(dataframe.resultados.2neu.soft)  
remove(dataframe.resultados.3neu)  
remove(dataframe.resultados.3neu.soft)

Ahora, con estos resultados en el dataframe, vamos a ver los mejores resultados que he obtenido con cada red neuronal:

head(dataframe.resultados.perceptron.stats[order(dataframe.resultados.perceptron.stats$Test\_1neu, decreasing = TRUE), 1:2])

## Ent\_1neu Test\_1neu  
## 8 0.9634222 0.9630200  
## 9 0.9616444 0.9607088  
## 10 0.9615778 0.9605547  
## 3 0.9621111 0.9604006  
## 1 0.9313111 0.9289676  
## 2 0.9234444 0.9208012

head(dataframe.resultados.perceptron.stats[order(dataframe.resultados.perceptron.stats$Test\_1neu\_soft, decreasing = TRUE), 3:4])

## Ent\_1neu\_soft Test\_1neu\_soft  
## 1 0.9616444 0.9610169  
## 6 0.9616667 0.9610169  
## 2 0.9617333 0.9608629  
## 3 0.9615556 0.9608629  
## 5 0.9616667 0.9608629  
## 7 0.9615778 0.9608629

head(dataframe.resultados.perceptron.stats[order(dataframe.resultados.perceptron.stats$Test\_2neu, decreasing = TRUE), 5:6])

## Ent\_2neu Test\_2neu  
## 4 0.9665333 0.9627119  
## 1 0.9645556 0.9620955  
## 6 0.9374889 0.9354391  
## 9 0.9321556 0.9326656  
## 3 0.9285778 0.9246533  
## 5 0.9251778 0.9228043

head(dataframe.resultados.perceptron.stats[order(dataframe.resultados.perceptron.stats$Test\_2neu\_soft, decreasing = TRUE), 7:8])

## Ent\_2neu\_soft Test\_2neu\_soft  
## 3 0.9694444 0.9676425  
## 9 0.9684222 0.9667180  
## 4 0.9691333 0.9662558  
## 8 0.9678222 0.9656394  
## 5 0.9674000 0.9648690  
## 10 0.9668889 0.9630200

head(dataframe.resultados.perceptron.stats[order(dataframe.resultados.perceptron.stats$Test\_3neu, decreasing = TRUE), 9:10])

## Ent\_3neu Test\_3neu  
## 7 0.9157778 0.9694915  
## 4 0.9157778 0.9616333  
## 2 0.9157778 0.9280431  
## 1 0.9157778 0.9235747  
## 6 0.9157778 0.9228043  
## 8 0.9157778 0.9163328

head(dataframe.resultados.perceptron.stats[order(dataframe.resultados.perceptron.stats$Test\_3neu\_soft, decreasing = TRUE), 11:12])

## Ent\_3neu\_soft Test\_3neu\_soft  
## 10 0.9711111 0.9704160  
## 5 0.9702222 0.9687211  
## 3 0.9705778 0.9684129  
## 7 0.9692444 0.9677966  
## 4 0.9690444 0.9671803  
## 8 0.9691556 0.9670262

Como podemos ver, el mejor resultado lo obtenemos con 3 neuronas y softmax, pero no difiere prácticamente nada con 1 neurona o 2 neuronas. De este modo, ante la igualdad de condiciones, la mejor red es la más simple, y por lo tanto la de 1 neurona.

#### KNN

K - Nearest Neighbours es otro algoritmo de clasificación muy utilizado. Se basa en la posición del nodo que se está analizando con los K nodos más cercanos mediante distancia euclídea. De este modo, calcula la similitud e indentifica a un nodo como de un grupo o de otro.

La teoría dice que para K-NN, el mejor número a escoger es la raíz cuadrada del total de observaciones que tenemos. Lo calculamos:

k <- round(sqrt(nrow(data.ml.stats.centscal)), 0)

Ahora creamos los grupos:

conjuntoEntrenamiento <- data.ml.stats.centscal[1:45000, ]  
conjuntoTest <- data.ml.stats.centscal[45001 : nrow(data.ml.stats.centscal), ] # Utilizo por supuesto la matriz de centrado y escalado  
  
etiquetasEntrenamiento <- data.ml[1:45000, 2]  
etiquetasTest <- data.ml[45001:nrow(data.ml.stats.centscal), 2]

Ahora lo que vamos a hacer es calcular para diferentes K’s el resultado que obtenemos, para ver qué tal clasifica:

###### K = 227

prediccion.knn.227 <- knn(train = conjuntoEntrenamiento,   
 test = conjuntoTest,   
 cl = etiquetasEntrenamiento,  
 k = 227)   
head(prediccion.knn.227)

## [1] 2 2 1 1 1 2  
## Levels: 1 2

Sacamos crosstable:

CrossTable(x = etiquetasTest ,  
 y = prediccion.knn.227,   
 prop.chisq = FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 6490   
##   
##   
## | prediccion.knn.227   
## etiquetasTest | 1 | 2 | Row Total |   
## --------------|-----------|-----------|-----------|  
## 1 | 3098 | 156 | 3254 |   
## | 0.952 | 0.048 | 0.501 |   
## | 0.948 | 0.048 | |   
## | 0.477 | 0.024 | |   
## --------------|-----------|-----------|-----------|  
## 2 | 170 | 3066 | 3236 |   
## | 0.053 | 0.947 | 0.499 |   
## | 0.052 | 0.952 | |   
## | 0.026 | 0.472 | |   
## --------------|-----------|-----------|-----------|  
## Column Total | 3268 | 3222 | 6490 |   
## | 0.504 | 0.496 | |   
## --------------|-----------|-----------|-----------|  
##   
##

Ahora voy a hacer lo mismo con 150 y con 350, para ver las diferencias que puede haber con la predicción:

###### K = 150

prediccion.knn.150 <- knn(train = conjuntoEntrenamiento,   
 test = conjuntoTest,   
 cl = etiquetasEntrenamiento,  
 prob = TRUE,  
 k = 150)   
head(prediccion.knn.150)

## [1] 2 2 1 1 1 2  
## Levels: 1 2

Sacamos crosstable:

CrossTable(x = etiquetasTest ,  
 y = prediccion.knn.150,   
 prop.chisq = FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 6490   
##   
##   
## | prediccion.knn.150   
## etiquetasTest | 1 | 2 | Row Total |   
## --------------|-----------|-----------|-----------|  
## 1 | 3116 | 138 | 3254 |   
## | 0.958 | 0.042 | 0.501 |   
## | 0.950 | 0.043 | |   
## | 0.480 | 0.021 | |   
## --------------|-----------|-----------|-----------|  
## 2 | 163 | 3073 | 3236 |   
## | 0.050 | 0.950 | 0.499 |   
## | 0.050 | 0.957 | |   
## | 0.025 | 0.473 | |   
## --------------|-----------|-----------|-----------|  
## Column Total | 3279 | 3211 | 6490 |   
## | 0.505 | 0.495 | |   
## --------------|-----------|-----------|-----------|  
##   
##

###### K = 350

prediccion.knn.350 <- knn(train = conjuntoEntrenamiento,   
 test = conjuntoTest,   
 cl = etiquetasEntrenamiento,  
 prob = TRUE,  
 k = 350)  
head(prediccion.knn.350)

## [1] 2 2 1 1 1 2  
## Levels: 1 2

Sacamos crosstable:

CrossTable(x = etiquetasTest ,  
 y = prediccion.knn.350,   
 prop.chisq = FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 6490   
##   
##   
## | prediccion.knn.350   
## etiquetasTest | 1 | 2 | Row Total |   
## --------------|-----------|-----------|-----------|  
## 1 | 3104 | 150 | 3254 |   
## | 0.954 | 0.046 | 0.501 |   
## | 0.942 | 0.047 | |   
## | 0.478 | 0.023 | |   
## --------------|-----------|-----------|-----------|  
## 2 | 191 | 3045 | 3236 |   
## | 0.059 | 0.941 | 0.499 |   
## | 0.058 | 0.953 | |   
## | 0.029 | 0.469 | |   
## --------------|-----------|-----------|-----------|  
## Column Total | 3295 | 3195 | 6490 |   
## | 0.508 | 0.492 | |   
## --------------|-----------|-----------|-----------|  
##   
##

#### Random Forest

Random Forest es otro algoritmo supervisado que se basa en la creación de árboles de decisión, en los cuales se hacen preguntas y según la contestación obtenemos un resultado u otro. Finalmente, todos los bosques que se forman llegan a un acuerdo y se pone la decisión final.

Para el random forest no se necesita por su naturaleza un centrado y escalado anterior, de tal manera que directamente procedo con la parte de estadísticas del dataframe original:

data.ml.rf <- data.ml[, 1:18]  
data.ml.rf <- data.ml.rf[, -2] # Me quedo solo con la parte de stats y sin el ganador  
  
model <- randomForest(as.factor(data.ml[, 2]) ~ ., data = data.ml.rf, importance = TRUE, ntree = 300)  
model

##   
## Call:  
## randomForest(formula = as.factor(data.ml[, 2]) ~ ., data = data.ml.rf, importance = TRUE, ntree = 300)   
## Type of random forest: classification  
## Number of trees: 300  
## No. of variables tried at each split: 4  
##   
## OOB estimate of error rate: 2.81%  
## Confusion matrix:  
## 1 2 class.error  
## 1 25463 614 0.02354565  
## 2 835 24578 0.03285720

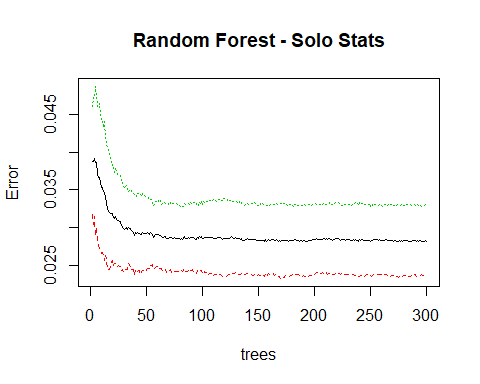
Aquí podemos ver la matriz de confusión, de la que obtenemos también el fallo por clases.

El Out-Of-Bag es un método de estimación de error que se usa en algunos algoritmos como Random Forest, y usa el modelo de Bagging para hacer muestras de submuestras usadas para el entrenamiento. El OOB es el error de predidcción medio de cada una de las muestras de entrenamiento.

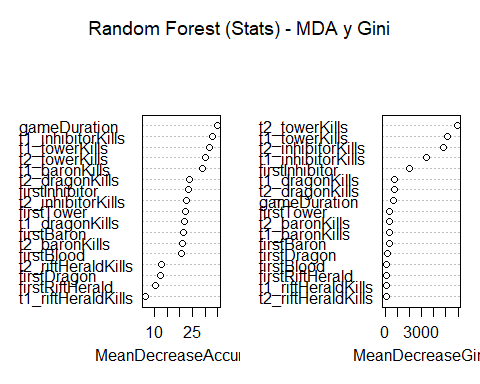
Bagging es un meta-algoritmo usado para aumentar la estabilidad y precisión de algoritmos de Machine Learning de clasificación y regresión.

Ahora obtenemos el número de árboles que necesitamos realmente, y la importancia de las variables en este modelo:

plot(model, main="Random Forest - Solo Stats")



varImpPlot(model, main = "Random Forest (Stats) - MDA y Gini") # Gracias a importance = true



Lo exportamos a PDF…

pdf("../imagenes/RandomForest\_FT.pdf")  
plot(model, main="Random Forest - Solo Stats")  
varImpPlot(model, main = "Random Forest (Stats) - MDA y Gini")  
dev.off

## function (which = dev.cur())   
## {  
## if (which == 1)   
## stop("cannot shut down device 1 (the null device)")  
## .External(C\_devoff, as.integer(which))  
## dev.cur()  
## }  
## <bytecode: 0x00000000134d26f0>  
## <environment: namespace:grDevices>

Vamos a interpretar estos datos del modelo:

* MeanDecreaseAccuracy se refiere al decremento de la exactitud del modelo si se permutan los valores en cada característica. En otras palabras, MDA nos muestra la media de valores que se clasificarían mal si se quitara esa característica de la predicción. Los valores que nos da MDA son la media de las variables que no se clasificarían bien en caso de quitar esas variables del entrenamiento del modelo. En nuestro caso, game duration, los inhibidores y las torres son lo que más información le está aportando al modelo.
* MeanDecreaseGini se refiere a la medida de la ganancia media de pureza mediante la división de cierta variable. Cuanto más importante sea la variable, habrá un mayor descenso en el Gini. La importancia de este está íntimamente relacionada a la función de decisión local, que Random Forest usa para seleccionar cual es la mejor separación. Como vemos, las torres y los inhibidores vuelven a ser cruciales para determinar quién es el ganador de la partida.

Ahora lo voy a hacer con 10 fold X Validation:

result <- rfcv(data.ml.rf, as.factor(data.ml[, 2]), cv.fold=10)   
#head(result)

#### SVM Kernel Lineal

SVM es una técnica supervisada que es muy robusta frente a la dimensionalidad. En este caso, nuestro problema no tiene una dimensión excesiva, pero Vamos a usar esta técnica para ver qué resultados nos arroja. En este caso, con Kernel Lineal:

modelo.svm <- svm(data.ml.stats.centscal, as.factor(data.ml[, 2]), kernel = "linear") # Al poner los grupos como factor, estoy consiguiendo que no sean continuos para el modelo, sino "discretos", ya que los factor no son valores que puedan ser continuos. Con esto consigo una clasificación.  
summary(modelo.svm)

##   
## Call:  
## svm.default(x = data.ml.stats.centscal, y = as.factor(data.ml[, 2]), kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1   
## gamma: 0.05882353   
##   
## Number of Support Vectors: 4818  
##   
## ( 2407 2411 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 1 2

Ahora que tenemos creado este primer modelo, toca predecir:

prediccion <- predict(modelo.svm, data.ml.stats.centscal)  
head(prediccion)

## 1 2 3 4 5 6   
## 1 1 1 1 1 1   
## Levels: 1 2

Ahora que hemos predicho, tenemos que sacar la matriz de confusión:

matriz.conf <- table(prediccion, data.ml[, 2])  
matriz.conf

##   
## prediccion 1 2  
## 1 25038 945  
## 2 1039 24468

sum(diag(matriz.conf))/nrow(data.ml)

## [1] 0.9614682

#### SVM Kernel Radial

Ahora vamos a probar SVM pero con Kernel Radial:

modelo\_svm.radial <- svm(data.ml.stats.centscal, as.factor(data.ml[, 2]), kernel="radial")  
summary(modelo\_svm.radial)

##   
## Call:  
## svm.default(x = data.ml.stats.centscal, y = as.factor(data.ml[, 2]), kernel = "radial")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 1   
## gamma: 0.05882353   
##   
## Number of Support Vectors: 4273  
##   
## ( 2135 2138 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 1 2

Aquí tenemos una C-Classification (necesaria para clasificar), con Kernel esta vez radial.

Ahora que tenemos creado este segundo modelo, toca predecir:

prediccion.radial <- predict(modelo\_svm.radial, data.ml.stats.centscal)  
head(prediccion.radial)

## 1 2 3 4 5 6   
## 1 1 1 1 1 1   
## Levels: 1 2

Ahora que hemos predicho, tenemos que sacar la matriz de confusión:

matriz.conf.radial <- table(prediccion.radial, data.ml[, 2])  
matriz.conf.radial

##   
## prediccion.radial 1 2  
## 1 25527 798  
## 2 550 24615

sum(diag(matriz.conf.radial))/nrow(data.ml)

## [1] 0.9738202

#### Epílogo Modelos Supervisados

Como hemos visto, solo con la parte de estadísticas podemos obtener una predicción de los resultados bastante acertada. Los otros dos dataframes que he creado (champions y total) tendríamos que probarlos ante los mismos métodos, pero el problema es que son demasiado grandes como para ejecutarlos en un ordenador de una manera medianamente rápida (según mis cálculos, en mi portátil son más de 2 días de computación contínua). De este modo, queda aquí indicado que la prueba de ellos (especialmente el de sólo campeones) sería una buena muestra de ver si se puede predecir el resultado de la partida sin las estadísticas principales.

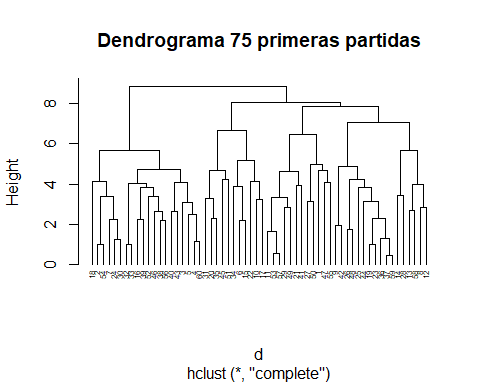
### Algoritmos no supervisados

Como podemos ver, los datos de ninguna manera son perfectamente separables, sino que las partidas que gana el equipo 1 se centran más a la izquierda en esta componente discriminante que forma x, y las del dos se centran más a la derecha. Como podemos ver, está bastante difuminada la frontera.

#### Cluster Jerárquico

En este caso, el cluster lo haré con un número reducido de partidas, ya que si no el cluster no se verá demasiado bien:

#Usamos la matriz de centrado y escalado para que cada coordenada represente el mismo grado de diferencia  
  
d <- dist(data.ml.stats.centscal[1:60, ], method = "euclidean")  
  
cluster.jerarquico <- hclust(d, method = "complete" )  
  
plot(cluster.jerarquico, cex = 0.5, hang = -1, main="Dendrograma 75 primeras partidas")



Exportamos a PDF…

pdf("../imagenes/clusterJerárquico.pdf")  
  
plot(cluster.jerarquico, cex = 0.5, hang = -1, main="Dendrograma 60 primeras partidas")  
  
dev.off()

## png   
## 2