Cómo ganar al League of Legends

Assingment Brief de la asignatura Machine Learning

May 17th, 2023

Análisis en R de qué variables son más importantes a la hora de ganar una partida del aclamado título de *Riot Games*.

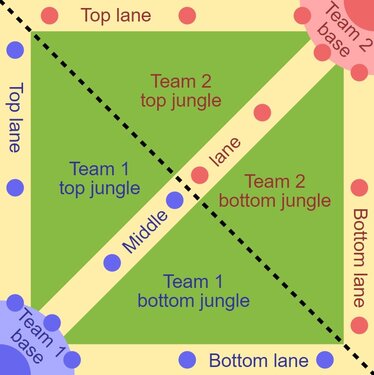
# Introducción

*League of Legends* es un juego multijugador de arena de batalla en línea (*MOBA*), desarrollado y publicado por *Riot Games* en 2009. En cada partida, dos equipos de cinco integrantes cada uno, lucharán por derribar el “Nexo” enemigo, el cual es una estructura colocada en el lado opuesto de donde comienza cada equipo. En el modo clásico, cada uno de los integrantes controlará un personaje distinto denominado “campeón”, con un set de habilidades único y diferentes estilos de juego.

El anterior mencionado “Nexo” está protegido por distintas estructuras y “súbditos”, unos enemigos que aparecen periódicamente desde cada lado del mapa para ayudar a los “campeones” en su propósito de abrirse paso hacia la base enemiga. Esos súbditos dan “oro” y “experiencia”, lo cual es necesario para mejorar a tu “campeón” a base de objetos (que se compran con “oro”) y “niveles” (los cuales mejoran tus “habilidades”).

La arena se conoce como “La Grieta del Invocador” y toma esta estructura (simplificada):

* Hay 3 “líneas” o calles: por ellas caminarán los “súbditos” y están protegidas por 3 “torres” por cada equipo en cada calle antes de llegar a la “base” rival. Las “líneas” son *Top, Mid* y *Bot.*
* El resto del mapa que no es ni una “base” ni una “calle” (en verde oscuro en la imagen) se conoce como “jungla”, en la cual hay monstruos neutrales (los cuales se comportan como los “súbditos” mencionados anteriormente) y objetivos importantes para acabar la partida como los “Dragones”, “Heraldos” y “Barón Nashor”. Estos objetivos proporcionan mejoras al equipo que los reclame.



Grieta del Invocador (simplificada)

[Original PNG version by Raizin, SVG rework by Sameboat](https://commons.wikimedia.org/w/index.php?curid=29443207)

Los “campeones” por defecto no pueden ver todas las zonas del mapa (y algunos “campeones” basan su identidad en no ser vistos, p. ej. *Shaco* o *Evelynn*), por lo que es necesario comprar “guardianes de visión” para colocar en las zonas de “jungla” y posteriormente en las zonas del mapa derribadas durante el transcurso de la partida.

La mayoría de eventos generan “oro”, p. ej., la primera baja que suceda en la partida (*first Blood*) da 400 de “oro” al “campeón” que la reclame, una baja habitual da 300, romper una torre da 50 de oro a cada miembro del equipo y 100 al que la tire, un tótem da 20, etcétera.

En *League of Legends* es posible rendirse si al menos 4/5 de los integrantes del equipo están dispuestos a ello en el minuto 20 de partida, existiendo una ocasión extraordinaria a los 15 minutos si dicha rendición es **unánime.**

Este título se ha coronado como uno de los videjuegos con más jugadores concurrentes, con una cifra de [152 millones](https://activeplayer.io/league-of-legends/) de jugadores a día de hoy. Estas cifras han hecho que este juego tenga mucha actividad a su alrededor, y mucha actividad económica se haya desarrollado en su ecosistema, como merchandising no-oficial, o aplicaciones de terceros.

[*Mobalytics*](https://mobalytics.gg/) es una empresa que desarrolla *software* a modo de *overlay* (*software* que se ejecuta mientras juegas para añadirte información) sobre distintos títulos de *Riot Games*, incluido el *League of Legends*. Esta empresa recopila datos sobre sus usuarios y el desempeño en sus partidas de cara a proporcionar mejores guías e indicadores para que sus usuarios puedan mejorar en el videojuego.

Este *overlay* necesita conocer qué datos son más determinantes de cara a predecir si la partida se puede ganar o no, de cara a mostrar a sus usuarios si votar rendirse porque la partida está **inevitablemente perdida**, o si por el contrario todavía hay un tanto por ciento suficiente de posibilidades de victoria como para que merezca la pena seguir jugando.

# Descripción del problema

Con esta introducción simplificada del funcionamiento del videojuego, se ilustra que existen muchas variables que dan ventaja a un equipo u otro. Aunque el objetivo final sea inequívoco para ganar la partida (derribar el nexo enemigo), hay muchas otras variables no tan directas como “Nexo destruido: Sí o No” que hay que tener en cuenta para dictaminar *a priori* qué va a pasar en los minutos siguientes, y poder predecir qué resultado va a tener la partida antes de derribar el nexo.

La amplia popularidad de este videjouego hace que sea muy factible extraer datos de este, ya que dispone de API pública, por lo que tendremos a disposición las variables cualesquiera que necesitemos. El objetivo en este informe es escoger un modelo de *machine learning* que prediga qué equipo va a ganar una partida en función de qué haya sucedido en la misma, asimismo como decidir cuáles son las variables más determinantes en dicho modelo.

# Análisis en R

## Datos de origen

En principio se pretendía obtener datos reales al acceder a la [API Oficial](https://developer.riotgames.com/) de Riot Games, pero se demora entre 2 y 4 semanas de espera y tienes acceso limitado de entradas por consulta, por lo que se acabó descartando por falta de tiempo. Es planteable en un futuro reescribir este trabajo con datos de mi propia cosecha.

El *dataset* ha sido obtenido de forma pública y se puede consultar en [Kaggle](https://www.kaggle.com/datasets/bobbyscience/league-of-legends-diamond-ranked-games-10-min). El *dataset* contiene datos sobre partidas de [*elo*](https://en.wikipedia.org/wiki/Elo_rating_system) alto (de diamante alto a master, se adjunta una imagen para ilustrar que son el [top 2% de jugadores](https://www.leagueofgraphs.com/rankings/rank-distribution)), en sus primeros 10 min de duración.

## Librerías y semilla

A continuación se descargan e instalan las librerías de R y se establece una semilla para garantizar que se puede repetir el ejercicio.

library(caret)  
library(caretEnsemble)  
library(corrr)  
library(corrplot)  
library(dplyr)  
library(e1071)  
library(factoextra)  
library(GGally)  
library(ggcorrplot)  
library(ggplot2)  
library(ISLR)  
library(Metrics)  
library(pls)  
library(pscl)  
library(pROC)  
library(psych)  
library(psychTools)  
library(rattle)  
library(readr)  
library(reshape2)  
library(rpart)  
library(rpart.plot)  
library(tidyverse)  
  
set.seed(123)

## Carga de datos:

Los datos de origen vienen en un CSV, y RStudio permite hacer una carga sin mayores transformaciones gracias a la librería [*readr*](https://cran.r-project.org/web/packages/readr/readme/README.html).

df <- read\_csv("data/high\_diamond\_ranked\_10min.csv")

## Rows: 9879 Columns: 40  
## ── Column specification ────────────────────────────────────  
## Delimiter: ","  
## dbl (40): gameId, blueWins, blueWardsPlaced, blueWardsDe...  
##   
## ℹ Use `spec()` to retrieve the full column specification for this data.  
## ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

# Exploración y limpieza de los datos

## Primer contacto

Se muestran las primeras líneas del *dataset*.

head(df)

## # A tibble: 6 × 40  
## gameId blueW…¹ blueW…² blueW…³ blueF…⁴ blueK…⁵ blueD…⁶  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 4519157822 0 28 2 1 9 6  
## 2 4523371949 0 12 1 0 5 5  
## 3 4521474530 0 15 0 0 7 11  
## 4 4524384067 0 43 1 0 4 5  
## 5 4436033771 0 75 4 0 6 6  
## 6 4475365709 1 18 0 0 5 3  
## # … with 33 more variables: blueAssists <dbl>,  
## # blueEliteMonsters <dbl>, blueDragons <dbl>,  
## # blueHeralds <dbl>, blueTowersDestroyed <dbl>,  
## # blueTotalGold <dbl>, blueAvgLevel <dbl>,  
## # blueTotalExperience <dbl>,  
## # blueTotalMinionsKilled <dbl>,  
## # blueTotalJungleMinionsKilled <dbl>, …

Se observa que hay 40 variables en el *dataset*, y a simple vista todas son numéricas. No obstante, se usa *str()* para obtener una descripción precisa sobre los tipos de variables.

str(df)

## spc\_tbl\_ [9,879 × 40] (S3: spec\_tbl\_df/tbl\_df/tbl/data.frame)  
## $ gameId : num [1:9879] 4.52e+09 4.52e+09 4.52e+09 4.52e+09 4.44e+09 ...  
## $ blueWins : num [1:9879] 0 0 0 0 0 1 1 0 0 1 ...  
## $ blueWardsPlaced : num [1:9879] 28 12 15 43 75 18 18 16 16 13 ...  
## $ blueWardsDestroyed : num [1:9879] 2 1 0 1 4 0 3 2 3 1 ...  
## $ blueFirstBlood : num [1:9879] 1 0 0 0 0 0 1 0 0 1 ...  
## $ blueKills : num [1:9879] 9 5 7 4 6 5 7 5 7 4 ...  
## $ blueDeaths : num [1:9879] 6 5 11 5 6 3 6 13 7 5 ...  
## $ blueAssists : num [1:9879] 11 5 4 5 6 6 7 3 8 5 ...  
## $ blueEliteMonsters : num [1:9879] 0 0 1 1 0 1 1 0 0 1 ...  
## $ blueDragons : num [1:9879] 0 0 1 0 0 1 1 0 0 1 ...  
## $ blueHeralds : num [1:9879] 0 0 0 1 0 0 0 0 0 0 ...  
## $ blueTowersDestroyed : num [1:9879] 0 0 0 0 0 0 0 0 0 0 ...  
## $ blueTotalGold : num [1:9879] 17210 14712 16113 15157 16400 ...  
## $ blueAvgLevel : num [1:9879] 6.6 6.6 6.4 7 7 7 6.8 6.4 7.2 6.8 ...  
## $ blueTotalExperience : num [1:9879] 17039 16265 16221 17954 18543 ...  
## $ blueTotalMinionsKilled : num [1:9879] 195 174 186 201 210 225 225 209 189 220 ...  
## $ blueTotalJungleMinionsKilled: num [1:9879] 36 43 46 55 57 42 53 48 61 39 ...  
## $ blueGoldDiff : num [1:9879] 643 -2908 -1172 -1321 -1004 ...  
## $ blueExperienceDiff : num [1:9879] -8 -1173 -1033 -7 230 ...  
## $ blueCSPerMin : num [1:9879] 19.5 17.4 18.6 20.1 21 22.5 22.5 20.9 18.9 22 ...  
## $ blueGoldPerMin : num [1:9879] 1721 1471 1611 1516 1640 ...  
## $ redWardsPlaced : num [1:9879] 15 12 15 15 17 36 57 15 15 16 ...  
## $ redWardsDestroyed : num [1:9879] 6 1 3 2 2 5 1 0 2 2 ...  
## $ redFirstBlood : num [1:9879] 0 1 1 1 1 1 0 1 1 0 ...  
## $ redKills : num [1:9879] 6 5 11 5 6 3 6 13 7 5 ...  
## $ redDeaths : num [1:9879] 9 5 7 4 6 5 7 5 7 4 ...  
## $ redAssists : num [1:9879] 8 2 14 10 7 2 9 11 5 4 ...  
## $ redEliteMonsters : num [1:9879] 0 2 0 0 1 0 0 1 2 0 ...  
## $ redDragons : num [1:9879] 0 1 0 0 1 0 0 1 1 0 ...  
## $ redHeralds : num [1:9879] 0 1 0 0 0 0 0 0 1 0 ...  
## $ redTowersDestroyed : num [1:9879] 0 1 0 0 0 0 0 0 0 0 ...  
## $ redTotalGold : num [1:9879] 16567 17620 17285 16478 17404 ...  
## $ redAvgLevel : num [1:9879] 6.8 6.8 6.8 7 7 7 6.4 6.6 7.2 6.8 ...  
## $ redTotalExperience : num [1:9879] 17047 17438 17254 17961 18313 ...  
## $ redTotalMinionsKilled : num [1:9879] 197 240 203 235 225 221 164 157 240 247 ...  
## $ redTotalJungleMinionsKilled : num [1:9879] 55 52 28 47 67 59 35 54 53 43 ...  
## $ redGoldDiff : num [1:9879] -643 2908 1172 1321 1004 ...  
## $ redExperienceDiff : num [1:9879] 8 1173 1033 7 -230 ...  
## $ redCSPerMin : num [1:9879] 19.7 24 20.3 23.5 22.5 22.1 16.4 15.7 24 24.7 ...  
## $ redGoldPerMin : num [1:9879] 1657 1762 1728 1648 1740 ...  
## - attr(\*, "spec")=  
## .. cols(  
## .. gameId = col\_double(),  
## .. blueWins = col\_double(),  
## .. blueWardsPlaced = col\_double(),  
## .. blueWardsDestroyed = col\_double(),  
## .. blueFirstBlood = col\_double(),  
## .. blueKills = col\_double(),  
## .. blueDeaths = col\_double(),  
## .. blueAssists = col\_double(),  
## .. blueEliteMonsters = col\_double(),  
## .. blueDragons = col\_double(),  
## .. blueHeralds = col\_double(),  
## .. blueTowersDestroyed = col\_double(),  
## .. blueTotalGold = col\_double(),  
## .. blueAvgLevel = col\_double(),  
## .. blueTotalExperience = col\_double(),  
## .. blueTotalMinionsKilled = col\_double(),  
## .. blueTotalJungleMinionsKilled = col\_double(),  
## .. blueGoldDiff = col\_double(),  
## .. blueExperienceDiff = col\_double(),  
## .. blueCSPerMin = col\_double(),  
## .. blueGoldPerMin = col\_double(),  
## .. redWardsPlaced = col\_double(),  
## .. redWardsDestroyed = col\_double(),  
## .. redFirstBlood = col\_double(),  
## .. redKills = col\_double(),  
## .. redDeaths = col\_double(),  
## .. redAssists = col\_double(),  
## .. redEliteMonsters = col\_double(),  
## .. redDragons = col\_double(),  
## .. redHeralds = col\_double(),  
## .. redTowersDestroyed = col\_double(),  
## .. redTotalGold = col\_double(),  
## .. redAvgLevel = col\_double(),  
## .. redTotalExperience = col\_double(),  
## .. redTotalMinionsKilled = col\_double(),  
## .. redTotalJungleMinionsKilled = col\_double(),  
## .. redGoldDiff = col\_double(),  
## .. redExperienceDiff = col\_double(),  
## .. redCSPerMin = col\_double(),  
## .. redGoldPerMin = col\_double()  
## .. )  
## - attr(\*, "problems")=<externalptr>

Se aprecia que son todas de tipo ‘num’. Para algunas funcionalidades de R será necesario tranformarlo a tipo factor pero se abordará sólo donde sea necesario. Se prosigue comprobando si hay elementos vacíos en el *dataset*.

colSums(is.na(df))

## gameId blueWins   
## 0 0   
## blueWardsPlaced blueWardsDestroyed   
## 0 0   
## blueFirstBlood blueKills   
## 0 0   
## blueDeaths blueAssists   
## 0 0   
## blueEliteMonsters blueDragons   
## 0 0   
## blueHeralds blueTowersDestroyed   
## 0 0   
## blueTotalGold blueAvgLevel   
## 0 0   
## blueTotalExperience blueTotalMinionsKilled   
## 0 0   
## blueTotalJungleMinionsKilled blueGoldDiff   
## 0 0   
## blueExperienceDiff blueCSPerMin   
## 0 0   
## blueGoldPerMin redWardsPlaced   
## 0 0   
## redWardsDestroyed redFirstBlood   
## 0 0   
## redKills redDeaths   
## 0 0   
## redAssists redEliteMonsters   
## 0 0   
## redDragons redHeralds   
## 0 0   
## redTowersDestroyed redTotalGold   
## 0 0   
## redAvgLevel redTotalExperience   
## 0 0   
## redTotalMinionsKilled redTotalJungleMinionsKilled   
## 0 0   
## redGoldDiff redExperienceDiff   
## 0 0   
## redCSPerMin redGoldPerMin   
## 0 0

No hay datos faltantes, por lo que el dataset es íntegro para seguir trabajando y no es necesario hacer tratamiento de faltantes.

Miramos propiedades generales de las variables con *describe()* del paquete [*psych*](https://www.rdocumentation.org/packages/psych/versions/2.3.3/topics/describe).

describe(df)

## vars n mean  
## gameId 1 9879 4500084044.85  
## blueWins 2 9879 0.50  
## blueWardsPlaced 3 9879 22.29  
## blueWardsDestroyed 4 9879 2.82  
## blueFirstBlood 5 9879 0.50  
## blueKills 6 9879 6.18  
## blueDeaths 7 9879 6.14  
## blueAssists 8 9879 6.65  
## blueEliteMonsters 9 9879 0.55  
## blueDragons 10 9879 0.36  
## blueHeralds 11 9879 0.19  
## blueTowersDestroyed 12 9879 0.05  
## blueTotalGold 13 9879 16503.46  
## blueAvgLevel 14 9879 6.92  
## blueTotalExperience 15 9879 17928.11  
## blueTotalMinionsKilled 16 9879 216.70  
## blueTotalJungleMinionsKilled 17 9879 50.51  
## blueGoldDiff 18 9879 14.41  
## blueExperienceDiff 19 9879 -33.62  
## blueCSPerMin 20 9879 21.67  
## blueGoldPerMin 21 9879 1650.35  
## redWardsPlaced 22 9879 22.37  
## redWardsDestroyed 23 9879 2.72  
## redFirstBlood 24 9879 0.50  
## redKills 25 9879 6.14  
## redDeaths 26 9879 6.18  
## redAssists 27 9879 6.66  
## redEliteMonsters 28 9879 0.57  
## redDragons 29 9879 0.41  
## redHeralds 30 9879 0.16  
## redTowersDestroyed 31 9879 0.04  
## redTotalGold 32 9879 16489.04  
## redAvgLevel 33 9879 6.93  
## redTotalExperience 34 9879 17961.73  
## redTotalMinionsKilled 35 9879 217.35  
## redTotalJungleMinionsKilled 36 9879 51.31  
## redGoldDiff 37 9879 -14.41  
## redExperienceDiff 38 9879 33.62  
## redCSPerMin 39 9879 21.73  
## redGoldPerMin 40 9879 1648.90  
## sd median  
## gameId 27573278.49 4510920346.0  
## blueWins 0.50 0.0  
## blueWardsPlaced 18.02 16.0  
## blueWardsDestroyed 2.17 3.0  
## blueFirstBlood 0.50 1.0  
## blueKills 3.01 6.0  
## blueDeaths 2.93 6.0  
## blueAssists 4.06 6.0  
## blueEliteMonsters 0.63 0.0  
## blueDragons 0.48 0.0  
## blueHeralds 0.39 0.0  
## blueTowersDestroyed 0.24 0.0  
## blueTotalGold 1535.45 16398.0  
## blueAvgLevel 0.31 7.0  
## blueTotalExperience 1200.52 17951.0  
## blueTotalMinionsKilled 21.86 218.0  
## blueTotalJungleMinionsKilled 9.90 50.0  
## blueGoldDiff 2453.35 14.0  
## blueExperienceDiff 1920.37 -28.0  
## blueCSPerMin 2.19 21.8  
## blueGoldPerMin 153.54 1639.8  
## redWardsPlaced 18.46 16.0  
## redWardsDestroyed 2.14 2.0  
## redFirstBlood 0.50 0.0  
## redKills 2.93 6.0  
## redDeaths 3.01 6.0  
## redAssists 4.06 6.0  
## redEliteMonsters 0.63 0.0  
## redDragons 0.49 0.0  
## redHeralds 0.37 0.0  
## redTowersDestroyed 0.22 0.0  
## redTotalGold 1490.89 16378.0  
## redAvgLevel 0.31 7.0  
## redTotalExperience 1198.58 17974.0  
## redTotalMinionsKilled 21.91 218.0  
## redTotalJungleMinionsKilled 10.03 51.0  
## redGoldDiff 2453.35 -14.0  
## redExperienceDiff 1920.37 28.0  
## redCSPerMin 2.19 21.8  
## redGoldPerMin 149.09 1637.8  
## trimmed mad  
## gameId 4504103897.37 19856848.76  
## blueWins 0.50 0.00  
## blueWardsPlaced 18.29 2.97  
## blueWardsDestroyed 2.61 1.48  
## blueFirstBlood 0.51 0.00  
## blueKills 6.04 2.97  
## blueDeaths 6.00 2.97  
## blueAssists 6.31 4.45  
## blueEliteMonsters 0.47 0.00  
## blueDragons 0.33 0.00  
## blueHeralds 0.11 0.00  
## blueTowersDestroyed 0.00 0.00  
## blueTotalGold 16439.35 1506.32  
## blueAvgLevel 6.92 0.30  
## blueTotalExperience 17946.56 1154.95  
## blueTotalMinionsKilled 217.21 22.24  
## blueTotalJungleMinionsKilled 50.36 8.90  
## blueGoldDiff 11.84 2360.30  
## blueExperienceDiff -35.52 1856.22  
## blueCSPerMin 21.72 2.22  
## blueGoldPerMin 1643.93 150.63  
## redWardsPlaced 18.37 2.97  
## redWardsDestroyed 2.51 1.48  
## redFirstBlood 0.49 0.00  
## redKills 6.00 2.97  
## redDeaths 6.04 2.97  
## redAssists 6.33 4.45  
## redEliteMonsters 0.50 0.00  
## redDragons 0.39 0.00  
## redHeralds 0.08 0.00  
## redTowersDestroyed 0.00 0.00  
## redTotalGold 16428.95 1466.29  
## redAvgLevel 6.93 0.30  
## redTotalExperience 17982.25 1150.50  
## redTotalMinionsKilled 217.89 22.24  
## redTotalJungleMinionsKilled 51.06 10.38  
## redGoldDiff -11.84 2360.30  
## redExperienceDiff 35.52 1856.22  
## redCSPerMin 21.79 2.22  
## redGoldPerMin 1642.90 146.63  
## min max  
## gameId 4295358071.0 4527990640.0  
## blueWins 0.0 1.0  
## blueWardsPlaced 5.0 250.0  
## blueWardsDestroyed 0.0 27.0  
## blueFirstBlood 0.0 1.0  
## blueKills 0.0 22.0  
## blueDeaths 0.0 22.0  
## blueAssists 0.0 29.0  
## blueEliteMonsters 0.0 2.0  
## blueDragons 0.0 1.0  
## blueHeralds 0.0 1.0  
## blueTowersDestroyed 0.0 4.0  
## blueTotalGold 10730.0 23701.0  
## blueAvgLevel 4.6 8.0  
## blueTotalExperience 10098.0 22224.0  
## blueTotalMinionsKilled 90.0 283.0  
## blueTotalJungleMinionsKilled 0.0 92.0  
## blueGoldDiff -10830.0 11467.0  
## blueExperienceDiff -9333.0 8348.0  
## blueCSPerMin 9.0 28.3  
## blueGoldPerMin 1073.0 2370.1  
## redWardsPlaced 6.0 276.0  
## redWardsDestroyed 0.0 24.0  
## redFirstBlood 0.0 1.0  
## redKills 0.0 22.0  
## redDeaths 0.0 22.0  
## redAssists 0.0 28.0  
## redEliteMonsters 0.0 2.0  
## redDragons 0.0 1.0  
## redHeralds 0.0 1.0  
## redTowersDestroyed 0.0 2.0  
## redTotalGold 11212.0 22732.0  
## redAvgLevel 4.8 8.2  
## redTotalExperience 10465.0 22269.0  
## redTotalMinionsKilled 107.0 289.0  
## redTotalJungleMinionsKilled 4.0 92.0  
## redGoldDiff -11467.0 10830.0  
## redExperienceDiff -8348.0 9333.0  
## redCSPerMin 10.7 28.9  
## redGoldPerMin 1121.2 2273.2  
## range skew kurtosis  
## gameId 232632569.0 -1.46 3.33  
## blueWins 1.0 0.00 -2.00  
## blueWardsPlaced 245.0 4.14 23.42  
## blueWardsDestroyed 27.0 2.85 17.18  
## blueFirstBlood 1.0 -0.02 -2.00  
## blueKills 22.0 0.54 0.26  
## blueDeaths 22.0 0.51 0.21  
## blueAssists 29.0 0.89 1.16  
## blueEliteMonsters 2.0 0.69 -0.50  
## blueDragons 1.0 0.57 -1.67  
## blueHeralds 1.0 1.60 0.55  
## blueTowersDestroyed 4.0 5.59 39.83  
## blueTotalGold 12971.0 0.47 0.48  
## blueAvgLevel 3.4 -0.34 1.11  
## blueTotalExperience 12126.0 -0.25 0.68  
## blueTotalMinionsKilled 193.0 -0.27 0.17  
## blueTotalJungleMinionsKilled 92.0 0.12 0.38  
## blueGoldDiff 22297.0 0.03 0.30  
## blueExperienceDiff 17681.0 0.02 0.36  
## blueCSPerMin 19.3 -0.27 0.17  
## blueGoldPerMin 1297.1 0.47 0.48  
## redWardsPlaced 270.0 4.56 30.45  
## redWardsDestroyed 24.0 2.95 18.22  
## redFirstBlood 1.0 0.02 -2.00  
## redKills 22.0 0.51 0.21  
## redDeaths 22.0 0.54 0.26  
## redAssists 28.0 0.82 0.78  
## redEliteMonsters 2.0 0.62 -0.57  
## redDragons 1.0 0.35 -1.88  
## redHeralds 1.0 1.85 1.44  
## redTowersDestroyed 2.0 5.34 30.53  
## redTotalGold 11520.0 0.41 0.22  
## redAvgLevel 3.4 -0.40 1.23  
## redTotalExperience 11804.0 -0.28 0.82  
## redTotalMinionsKilled 182.0 -0.29 0.23  
## redTotalJungleMinionsKilled 88.0 0.23 0.41  
## redGoldDiff 22297.0 -0.03 0.30  
## redExperienceDiff 17681.0 -0.02 0.36  
## redCSPerMin 18.2 -0.29 0.23  
## redGoldPerMin 1152.0 0.41 0.22  
## se  
## gameId 277416.26  
## blueWins 0.01  
## blueWardsPlaced 0.18  
## blueWardsDestroyed 0.02  
## blueFirstBlood 0.01  
## blueKills 0.03  
## blueDeaths 0.03  
## blueAssists 0.04  
## blueEliteMonsters 0.01  
## blueDragons 0.00  
## blueHeralds 0.00  
## blueTowersDestroyed 0.00  
## blueTotalGold 15.45  
## blueAvgLevel 0.00  
## blueTotalExperience 12.08  
## blueTotalMinionsKilled 0.22  
## blueTotalJungleMinionsKilled 0.10  
## blueGoldDiff 24.68  
## blueExperienceDiff 19.32  
## blueCSPerMin 0.02  
## blueGoldPerMin 1.54  
## redWardsPlaced 0.19  
## redWardsDestroyed 0.02  
## redFirstBlood 0.01  
## redKills 0.03  
## redDeaths 0.03  
## redAssists 0.04  
## redEliteMonsters 0.01  
## redDragons 0.00  
## redHeralds 0.00  
## redTowersDestroyed 0.00  
## redTotalGold 15.00  
## redAvgLevel 0.00  
## redTotalExperience 12.06  
## redTotalMinionsKilled 0.22  
## redTotalJungleMinionsKilled 0.10  
## redGoldDiff 24.68  
## redExperienceDiff 19.32  
## redCSPerMin 0.02  
## redGoldPerMin 1.50

## Variables correlacionadas por la naturaleza del problema

En base a mi conocimiento del funcionamiento del juego, hay variables que a simple vistazo van a tener gran correlación, por cómo se estructura el *dataset* y las reglas intrínsecas del propio *League of Legends*.

La columna *gameId* no aporta información puesto que no guarda nada mas allá que para cuantificar las partidas en los servidores de *Riot Games*, se puede quitar sin mayor impacto.

df <- df[,-1]

Por cómo funciona el juego, en este *dataset* hay muchas columnas con información redundante, puesto que si un equipo se hace una baja (*blueKill*), se corresponde con que el equipo rojo obtiene una muerte (*redDeaths*), por lo que esas columnas siempre van a contener información complementaria (*Discalimer:* soy plenamente consciente que es posible ejecutarse en el juego, pero son un caso ínfimo y que no varía el oro de la partida, por lo que al quitar las columnas de las muertes no alteramos ninguna de las otras).

df <- df[,!names(df) %in% c("blueDeaths", "redDeaths")]

De forma análoga, por las características del *dataset*, los datos están limitados a los primeros 10 minutos de la partida, por lo que también se pueden eliminar las columnas *PerMin* puesto que van a estar directamente correladas con columnas de su misma métrica. Las columnas que muestran un diferencial de oro o experiencia también se pueden suprimir en favor de los valores totales (*blueTotalGold*, y *redTotalGold*).

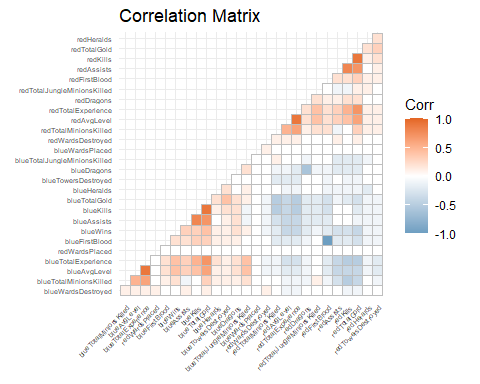
df <- df[,!names(df) %in% c("blueCSPerMin", "redCSPerMin",  
 "blueGoldPerMin", "redGoldPerMin",  
 "blueGoldDiff", "redGoldDiff",  
 "blueExperienceDiff","redExperienceDiff"  
 )]

Por último, este dataset agrupaba los [Monstruos Épicos](https://leagueoflegends.fandom.com/wiki/Category:Epic_monsters) en una misma columna, se ha decidido conservar con los datos por separado (*blueDragons, blueHeralds, redDragons, redHeralds*) para valorar si priorizar un objetivo u otro es determinante a la hora de ganar o perder.

df <- df[,!names(df) %in% c("blueEliteMonsters", "redEliteMonsters")]

Se buscan correlaciones entre las variables que quedan:

corr\_first <- round(cor(df), 1)  
ggcorrplot( corr\_first,  
 hc.order = TRUE,  
 type = "lower",  
 outline.col = "gray",  
 tl.cex = 5,  
 colors = c("#6D9EC1", "white", "#E46726")  
 ) + ggtitle("Correlation Matrix")



Se puede observar que *blueAVGlevel* tiene una correlación muy alta con *blueTotalExperience*, y sus contrapartidas del equipo rojo *redAVGLevel* y *redTotalExperience*, una vez más una está en función de la otra (*TotalExperience* = 5 \* *AVGLevel* \* experiencia necesaria para subir de nivel). Entonces, se elige una de las 2 para representar este dato, en esta caso los valores totales, quitando los promedios.

df <- df[,!names(df) %in% c("blueAVGLevel", "redAVGLevel")]

La columna *redFirstBlood* es la contraparte de *blueFirstBlood* (puesto que si no consigue un equipo la primera sangre la consigue el otro), se suprime la variable del equipo rojo para seguir la misma estructura en todo el *dataset* al tener sólo *blueWins*, que es la variable a predecir (no existe *redWins* en el *dataset* porque precisamente sería la contraparte).

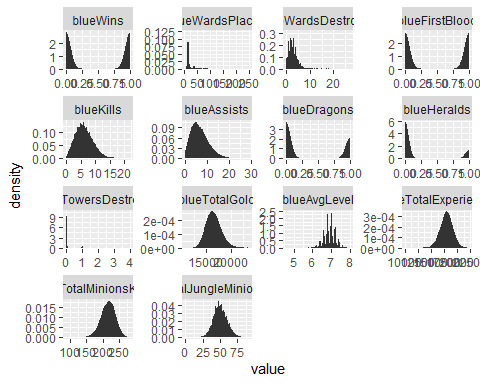
df <- df[,!names(df) %in% c("redFirstBlood")]

## Variables correlacionadas restantes

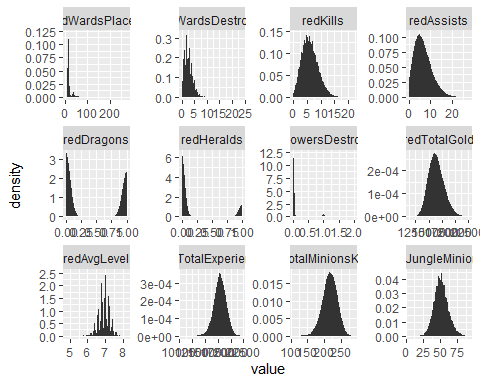
Cabe destacar una correlación natural entre *blueKills* y *blueTotalGold* (y sus contrapartidas *redKills* y *redTotalGold*), puesto que cada *kill* da 300 de oro, pero al haber más formas de conseguir oro en el juego, no es pertinente eliminar ninguna de las dos.

Se analiza la distribución, separando entre equipo rojo y equipo azul por claridad a la hora de mostrar los gráficos.

df\_sideBlue <- melt(df[1:14])  
df\_sideRed <- melt(df[15:26])  
  
ggplot(data = df\_sideBlue, aes(x = value))+  
 stat\_density()+  
 facet\_wrap(~variable, scales = "free")



ggplot(data=df\_sideRed, aes(x = value))+  
 stat\_density()+  
 facet\_wrap(~variable, scales = "free")

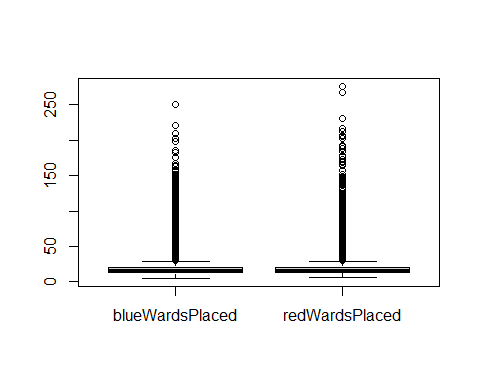


Se puede ver que las variables tienen una distribución normal cuando se trata de variables no dicotómicas, mientras las dicotómicas (*blueWins*, *blueFirstBlood*, *blueHerlads*, *blueDragons* y sus contrapartes en *red*) cumplen con su definición. Las variables *blueTowersDestroyed*, *blueWardsPlaced* y *blueWardsDestroyed* (y sus contrapartidas en *red*) siguen una distribución logarítmica normal, puesto que los sucesos alejados del eje *Y* son muy improbables.

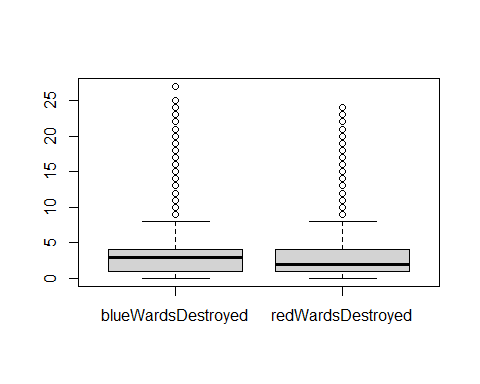
Los “Dragones” y “Heraldos” son dicotómicas por las características del *dataset*, puesto que no pueden suceder más de 1 vez antes de la marca temporal de los 10min.

Se hacen *boxplots* de los casos anómalos para encontrar *outliers* y decidir si se deben quitar.

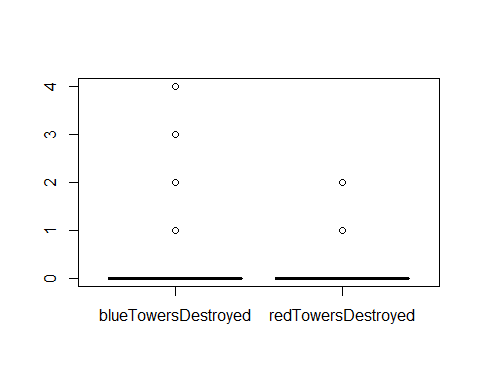
boxplot(df[c("blueWardsPlaced","redWardsPlaced")])



boxplot(df[c("blueWardsDestroyed","redWardsDestroyed")])



boxplot(df[c("blueTowersDestroyed","redTowersDestroyed")])



Se comprueba que son datos plausibles dentro de cómo funciona *League of Legends*, se podría valorar quitarlos si posteriormente perjudicasen al modelo. El caso más preocupante serían las partidas donde se tiran más de 2 torres al minuto 10, pero de momento no se van a sacar del *dataset*.

## Análisis de Componentes Principales

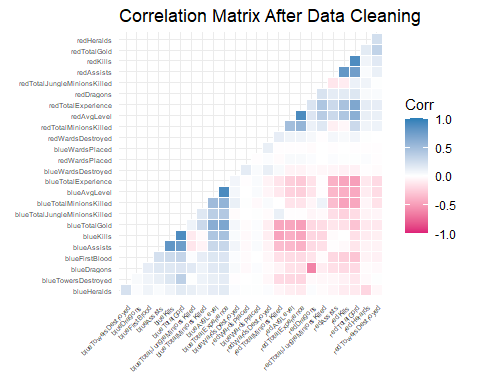
Como continuación del proceso de limpieza y exploración de los datos, se realiza un Análisis de Componentes Principales (PCA). En caso de que fuese necesario, se usará un *dataset* con el PCA en un modelo para compararlo con el modelo escogido.

Para hacer el análisis, se elimina la variable a predecir (*blueWins*).

df\_noWins <- df[,-1]  
df\_Wins <-df$blueWins

A continuación se buscan nuevamente correlaciones después de la limpieza de datos

corr\_post <- cor(df\_noWins)  
ggcorrplot( corr\_post,  
 hc.order = TRUE,  
 type = "lower",  
 outline.col = "white",  
 tl.cex = 5,  
 colors = c("#dd1c77", "white", "#2c7fb8")  
 ) + ggtitle("Correlation Matrix After Data Cleaning")



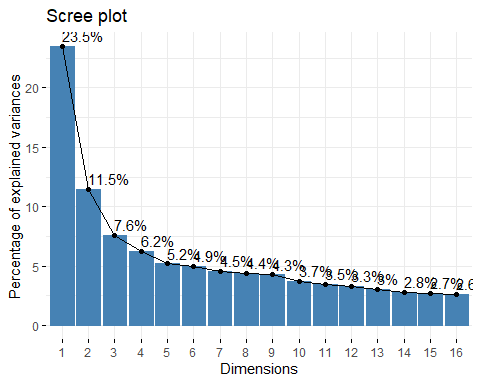
Se observa una tabla con el comportamiento que se esperaba, con los problemas de colinealidad descritos en [el apartado anterior](#variables-correlacionadas-restantes). A continuación, se hace un análisis de componentes principales del *dataset*, normalizándolo.

df\_PCA <- prcomp(df\_noWins, scale = TRUE)  
summary(df\_PCA)

## Importance of components:  
## PC1 PC2 PC3 PC4  
## Standard deviation 2.4229 1.6922 1.37776 1.24824  
## Proportion of Variance 0.2348 0.1145 0.07593 0.06232  
## Cumulative Proportion 0.2348 0.3494 0.42529 0.48762  
## PC5 PC6 PC7 PC8  
## Standard deviation 1.13916 1.11173 1.0654 1.04926  
## Proportion of Variance 0.05191 0.04944 0.0454 0.04404  
## Cumulative Proportion 0.53952 0.58896 0.6344 0.67840  
## PC9 PC10 PC11 PC12  
## Standard deviation 1.03163 0.96373 0.93028 0.90699  
## Proportion of Variance 0.04257 0.03715 0.03462 0.03291  
## Cumulative Proportion 0.72097 0.75812 0.79274 0.82565  
## PC13 PC14 PC15 PC16  
## Standard deviation 0.87277 0.83705 0.82399 0.80664  
## Proportion of Variance 0.03047 0.02803 0.02716 0.02603  
## Cumulative Proportion 0.85612 0.88414 0.91130 0.93733  
## PC17 PC18 PC19 PC20  
## Standard deviation 0.63606 0.57714 0.53651 0.43543  
## Proportion of Variance 0.01618 0.01332 0.01151 0.00758  
## Cumulative Proportion 0.95351 0.96683 0.97835 0.98593  
## PC21 PC22 PC23 PC24  
## Standard deviation 0.37592 0.27932 0.25695 0.18787  
## Proportion of Variance 0.00565 0.00312 0.00264 0.00141  
## Cumulative Proportion 0.99158 0.99470 0.99734 0.99876  
## PC25  
## Standard deviation 0.17631  
## Proportion of Variance 0.00124  
## Cumulative Proportion 1.00000

Se puede contemplar en la columna *‘Cummulative Proportion’* que se necesitan muchos componentes para explicar la varianza de los datos, y que a partir del decimotercer componente el porcentaje acumulado es ínfimo. Con la gráfica a continuación, gracias a la función [*fviz\_eig()*](https://www.rdocumentation.org/packages/factoextra/versions/1.0.7/topics/eigenvalue) del paquete *factoextra*, se ilustra lo que se acaba de mencionar con las proporciones de varianza acumuladas.

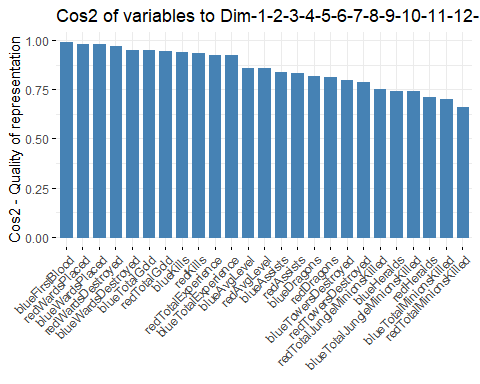
fviz\_eig(df\_PCA, addlabels = TRUE, ncp = 16)



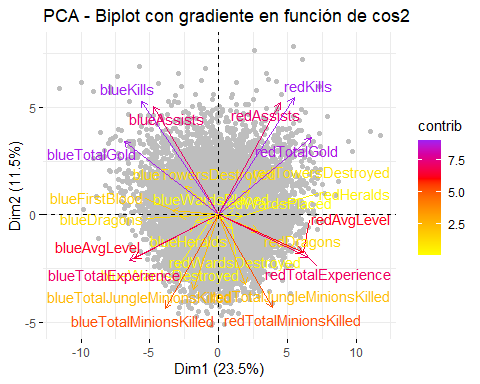
Con los 2 primeros componentes se podría explicar casi todo el *dataset*, pero se eligen los primeros 8 para tener un alto porcentaje (84.8%) de explicación de la variabilidad.

Con el gráfico a continuación se ilustra lo que aportan cada variable a las 13 dimensiones que se han elegido.

fviz\_cos2(df\_PCA, choice = "var", axes = 1:13)

 Se ilustra un biplot (con la función [*fviz\_pca\_biplot()*](https://www.rdocumentation.org/packages/factoextra/versions/1.0.7/topics/fviz_pca) en específico, la cual mejora en legibilidad) para ver con más claridad la relación entre cos2 con las 2 primeras dimensiones del PCA.

fviz\_pca\_biplot(df\_PCA,  
 label = "var",  
 repel = TRUE,  
 col.ind = "grey",  
 col.var = "contrib",  
 habillage = "none",  
 gradient.cols = c("yellow","orange","red","purple"),  
 title = "PCA - Biplot con gradiente en función de cos2"  
 )

 En este gráfico se pueden ver correlaciones ya descritas en la matriz, y con el gradiente las más oscuras son las que más contribuyen a las primeras 2 dimensiones.

Se prosigue con una muestra de los *eigenvalues* de cada variable contra cada componente principal. Se ha [invesigado](https://stats.stackexchange.com/a/88882) al respecto y R calcula los vectores y valores con el signo “opuesto” al intuitivo, mas cambiar un vector de signo sólo cambia su orientación y no su magnitud.

head(df\_PCA$x)

## PC1 PC2 PC3 PC4  
## [1,] -0.3053428 2.1567567 1.1178173 0.21444693  
## [2,] 2.7261154 0.2878242 1.8944058 -0.02591277  
## [3,] 1.6662611 3.0462708 2.0622460 -1.45422246  
## [4,] 0.7831661 -0.5541783 0.1346166 0.11302023  
## [5,] 0.7727639 -0.6926944 -1.2514918 1.03189628  
## [6,] -0.4832874 -1.5580756 0.6146685 -1.07015268  
## PC5 PC6 PC7 PC8  
## [1,] -0.5359094 -0.92323691 1.3831211 -1.24194915  
## [2,] -3.2192523 0.46505646 -3.2894528 -0.13815343  
## [3,] 1.5285756 1.32991290 0.1037891 1.14285837  
## [4,] 1.8597867 1.21407836 -0.2067669 -0.29879760  
## [5,] 0.1436694 -0.79728131 0.5527243 -0.08764263  
## [6,] -0.2734012 -0.08874767 0.6300268 -0.31863224  
## PC9 PC10 PC11 PC12  
## [1,] 0.7026742 0.3780208 -0.04857475 1.0054745  
## [2,] 1.1038654 0.6920393 1.83972743 -1.2751722  
## [3,] 0.2936312 0.1174155 -0.68855662 0.7206025  
## [4,] 0.9745045 -0.6603998 0.51442022 -0.8106494  
## [5,] 1.9127393 -2.0003779 0.03174135 -1.6959924  
## [6,] -0.1560849 -0.4021418 -1.26219503 0.6134183  
## PC13 PC14 PC15 PC16  
## [1,] -0.6222564 -0.25539111 -0.05188531 0.50081734  
## [2,] -1.5797719 -0.12523895 0.32271519 0.14189715  
## [3,] -0.2012029 -0.07697889 0.26308547 -0.22657499  
## [4,] -0.5986431 0.16534940 -0.20516803 -1.35355286  
## [5,] 0.2298255 0.08855118 -0.20179754 0.83033744  
## [6,] -2.1998252 -0.63147397 0.36609852 0.04866242  
## PC17 PC18 PC19 PC20  
## [1,] -0.3951641 -0.9990485 0.03643916 0.1374354  
## [2,] -0.2218126 0.4623759 -0.52339518 -0.1948706  
## [3,] 0.1106044 0.6733130 0.02366237 -0.2780962  
## [4,] -0.3289388 -1.0977424 -0.51104993 0.9686751  
## [5,] 0.4593451 -0.1305312 -0.38918008 -0.2354442  
## [6,] -0.1298036 0.4897135 -0.03444657 0.2732107  
## PC21 PC22 PC23 PC24  
## [1,] 0.343231502 0.018795933 -0.16326678 -0.12771061  
## [2,] -0.041366531 -0.251105801 -0.09882677 -0.11551519  
## [3,] 0.696613942 -0.003986194 0.17070557 0.34186355  
## [4,] 0.231952651 0.036859652 0.02523758 -0.06925737  
## [5,] 0.156473083 0.013415886 -0.17193011 -0.21145800  
## [6,] -0.007949892 0.013109423 -0.02668905 0.07445551  
## PC25  
## [1,] -0.18524586  
## [2,] -0.04308494  
## [3,] 0.08988234  
## [4,] 0.04959591  
## [5,] -0.13175387  
## [6,] -0.03801920

Se puede comprobar como en el primer componente todos los valores del equipo rojo (el contrario a la variable a predecir, *blueWins*) tienen un valor negativo, puesto que cuantos más recursos obtenga el equipo contrario, más dificil será ganar una partida.

## Separación de los datos

Se escoge repartir los datos de forma aleatoria en 2 subsets, uno de entrenamiento y otro para comprobar posteriormente los modelos frente a los datos originales. Dividimos también el *dataset* sin victorias. El *split* se hace un 80-20 para cada uno de los *datasets*. Tranformamos en factores la variable a predecir para garantizar el buen comportamiento de los paquetes de R.

mark <- sample(c ( TRUE , FALSE ), nrow (df), replace = TRUE , prob = c (0.8, 0.2))  
train <- df[mark, ]  
test <- df[!mark, ]  
  
train\_noWins <- df\_noWins[mark, ]  
test\_noWins <- df\_noWins[!mark, ]  
  
train\_asFactor <- train  
train\_asFactor$blueWins <- as.factor(train\_asFactor$blueWins)  
  
test\_asFactor <- test  
test\_asFactor$blueWins <- as.factor(test\_asFactor$blueWins)

# Modelos escogidos

En este trabajo se tratan 3 modelos distintos de *Machine Learning*:

1. [Regresión Logística](#regresión-logística)
2. [Naive Bayes](#naive-bayes)
3. [Árbol de Decisión](#árbol-de-decisión)

Posteriormente en el apartado de [Métricas](#métricas) se decidirá cuál es el más apto para la resolución del problema.

## Regresión Logística

### Modelo GLM

Se usa la función *glm()* (modelo lineal general) y especificamos *family = “binomial”* para usar un modelo de regresión logística al *dataset*. La variable a predecir es *blueWins*.

model\_GLM <- glm( blueWins ~ .,family = "binomial", data = train\_asFactor)  
summary(model\_GLM)

##   
## Call:  
## glm(formula = blueWins ~ ., family = "binomial", data = train\_asFactor)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.6969 -0.8696 0.1199 0.8658 2.7934   
##   
## Coefficients:  
## Estimate Std. Error z value  
## (Intercept) -1.741e-01 1.381e+00 -0.126  
## blueWardsPlaced -1.206e-03 1.499e-03 -0.805  
## blueWardsDestroyed -2.575e-03 1.297e-02 -0.199  
## blueFirstBlood 5.170e-02 5.878e-02 0.880  
## blueKills -3.398e-02 3.438e-02 -0.988  
## blueAssists -1.256e-02 1.300e-02 -0.966  
## blueDragons 3.943e-01 7.319e-02 5.388  
## blueHeralds 1.867e-02 7.295e-02 0.256  
## blueTowersDestroyed -2.846e-01 1.619e-01 -1.758  
## blueTotalGold 5.274e-04 7.804e-05 6.758  
## blueAvgLevel -1.792e-01 2.017e-01 -0.888  
## blueTotalExperience 2.495e-04 6.657e-05 3.748  
## blueTotalMinionsKilled -2.913e-03 2.169e-03 -1.343  
## blueTotalJungleMinionsKilled 3.182e-03 3.708e-03 0.858  
## redWardsPlaced -1.529e-03 1.446e-03 -1.057  
## redWardsDestroyed -1.350e-03 1.292e-02 -0.104  
## redKills -4.033e-04 3.415e-02 -0.012  
## redAssists 1.741e-02 1.264e-02 1.377  
## redDragons -2.826e-01 7.187e-02 -3.933  
## redHeralds -8.872e-02 7.727e-02 -1.148  
## redTowersDestroyed 3.666e-01 1.680e-01 2.182  
## redTotalGold -4.602e-04 7.716e-05 -5.964  
## redAvgLevel 1.049e-02 2.023e-01 0.052  
## redTotalExperience -2.650e-04 6.720e-05 -3.943  
## redTotalMinionsKilled 3.958e-03 2.139e-03 1.850  
## redTotalJungleMinionsKilled 6.679e-03 3.711e-03 1.800  
## Pr(>|z|)   
## (Intercept) 0.899662   
## blueWardsPlaced 0.420917   
## blueWardsDestroyed 0.842637   
## blueFirstBlood 0.379070   
## blueKills 0.323034   
## blueAssists 0.334001   
## blueDragons 7.13e-08 \*\*\*  
## blueHeralds 0.798049   
## blueTowersDestroyed 0.078706 .   
## blueTotalGold 1.40e-11 \*\*\*  
## blueAvgLevel 0.374401   
## blueTotalExperience 0.000178 \*\*\*  
## blueTotalMinionsKilled 0.179355   
## blueTotalJungleMinionsKilled 0.390833   
## redWardsPlaced 0.290360   
## redWardsDestroyed 0.916806   
## redKills 0.990578   
## redAssists 0.168414   
## redDragons 8.39e-05 \*\*\*  
## redHeralds 0.250842   
## redTowersDestroyed 0.029108 \*   
## redTotalGold 2.46e-09 \*\*\*  
## redAvgLevel 0.958637   
## redTotalExperience 8.06e-05 \*\*\*  
## redTotalMinionsKilled 0.064291 .   
## redTotalJungleMinionsKilled 0.071849 .   
## ---  
## Signif. codes:   
## 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 11034.9 on 7959 degrees of freedom  
## Residual deviance: 8382.6 on 7934 degrees of freedom  
## AIC: 8434.6  
##   
## Number of Fisher Scoring iterations: 4

Se puede observar que las variables con mayores p-valor son aquellas que son más importantes a la hora de ganar una partida, representadas con asteriscos en el resumen del modelo.

### Ajuste del Modelo R2

Para comprobar cúanto se ajusta el modelo a los datos, se usa R2 de McFadden:

pscl::pR2(model\_GLM)["McFadden"]

## fitting null model for pseudo-r2

## McFadden   
## 0.2403507

El R2 de McFadden de valores entre 0.2 y 0.4 indican que el modelo se ajusta [muy bien a los datos](https://eml.berkeley.edu/~mcfadden/travel.html), puesto que inferior a 0.2 es que el modelo está muy alejado, y más de 0.4 es que el modelo se ajusta de más, dando lugar a *overfitting*.

### Importancia de las variables y VIF

A continuación, sacamos una tabla con la importancia de las variables:

glm\_VarImp <- as.data.frame(caret::varImp(model\_GLM))  
glm\_VarImp <- data.frame(VarImp = glm\_VarImp$Overall,  
 names = rownames(glm\_VarImp))  
glm\_VarImp <- glm\_VarImp[order(glm\_VarImp$VarImp,decreasing = T),]  
rownames(glm\_VarImp) <- NULL  
glm\_VarImp

## VarImp names  
## 1 6.75804583 blueTotalGold  
## 2 5.96398995 redTotalGold  
## 3 5.38786020 blueDragons  
## 4 3.94261160 redTotalExperience  
## 5 3.93290681 redDragons  
## 6 3.74834217 blueTotalExperience  
## 7 2.18202304 redTowersDestroyed  
## 8 1.85015571 redTotalMinionsKilled  
## 9 1.80007647 redTotalJungleMinionsKilled  
## 10 1.75824302 blueTowersDestroyed  
## 11 1.37731870 redAssists  
## 12 1.34274259 blueTotalMinionsKilled  
## 13 1.14830573 redHeralds  
## 14 1.05733308 redWardsPlaced  
## 15 0.98824217 blueKills  
## 16 0.96608619 blueAssists  
## 17 0.88826045 blueAvgLevel  
## 18 0.87961095 blueFirstBlood  
## 19 0.85810711 blueTotalJungleMinionsKilled  
## 20 0.80483090 blueWardsPlaced  
## 21 0.25587311 blueHeralds  
## 22 0.19852218 blueWardsDestroyed  
## 23 0.10445804 redWardsDestroyed  
## 24 0.05186443 redAvgLevel  
## 25 0.01180887 redKills

Para comprobar si existen problemas de multicolinealidad, se usan los Factores de Inflación de Varianza (VIF).

sort(car::vif(model\_GLM), decreasing = TRUE)

## blueTotalGold redTotalGold   
## 13.886220 12.854941   
## blueKills redKills   
## 12.264234 11.199482   
## blueTotalExperience redTotalExperience   
## 6.461874 6.436347   
## blueAvgLevel redAvgLevel   
## 4.086301 3.969657   
## blueAssists redAssists   
## 3.354773 3.209163   
## blueTotalMinionsKilled redTotalMinionsKilled   
## 2.855013 2.805526   
## redTotalJungleMinionsKilled blueTotalJungleMinionsKilled   
## 1.883279 1.811416   
## redDragons blueDragons   
## 1.764463 1.716425   
## blueTowersDestroyed redTowersDestroyed   
## 1.504381 1.433555   
## blueFirstBlood blueHeralds   
## 1.213176 1.129593   
## redHeralds blueWardsDestroyed   
## 1.114912 1.069242   
## redWardsDestroyed redWardsPlaced   
## 1.055076 1.030849   
## blueWardsPlaced   
## 1.027129

Como regla general, los valores de VIF por encima de 5 indican una multicolinealidad severa. Dado que hay varias que superan el 10, es posible que se deba rehacer este modelo haciendo un tratamiento distinto, pero en principio la cadena lógica que nos lleva a elegir estas variables se mantiene, puesto que las que más problemas de colinealidad presentan son las que se han previsto en la [Exploración de Datos](#variables-correlacionadas-restantes).

En el apartado de [métricas](#Xd7da6b162034b5049400af9db82d28a5613540f) haremos más análisis a este modelo.

## Naive Bayes

En el caso del modelo Naive Bayes, se deben hacer algunas transformaciones a los datos

train\_NB <- train\_asFactor  
test\_NB <- test\_asFactor  
  
x\_train\_NB <- train\_NB[,-1]  
y\_train\_NB <- train\_NB$blueWins

### Modelo Naive Bayes

El modelo se basa en el teorema de Bayes, nombrado en honor de Thomas Bayes (1700). este teorema asume que todas las variables son independientes y da la misma importancia a todas las variables. Se usa la librería *caret*.

model\_NB = caret::train(x\_train\_NB, y\_train\_NB, 'naive\_bayes', trControl=trainControl(method='cv',number=10))  
model\_NB

## Naive Bayes   
##   
## 7960 samples  
## 25 predictor  
## 2 classes: '0', '1'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 7163, 7164, 7163, 7164, 7165, 7165, ...   
## Resampling results across tuning parameters:  
##   
## usekernel Accuracy Kappa   
## FALSE 0.7248796 0.4497802  
## TRUE 0.7272693 0.4544962  
##   
## Tuning parameter 'laplace' was held constant at a value  
## of 0  
## Tuning parameter 'adjust' was held constant at  
## a value of 1  
## Accuracy was used to select the optimal model using  
## the largest value.  
## The final values used for the model were laplace =  
## 0, usekernel = TRUE and adjust = 1.

En el apartado [métricas](#métricas-del-modelo-de-naive-bayes) se comparará con otros modelos.

### Importancia de las variables

nb\_VarImp <- as.data.frame(caret::filterVarImp(model\_NB$trainingData, y\_train\_NB, nonpara = TRUE))  
nb\_VarImp <- data.frame(VarImp = nb\_VarImp$X0, names = rownames(nb\_VarImp))  
nb\_VarImp <- head(nb\_VarImp, -1)  
nb\_VarImp <- nb\_VarImp[order(nb\_VarImp$VarImp, decreasing = T),]  
rownames(nb\_VarImp) <- NULL  
nb\_VarImp

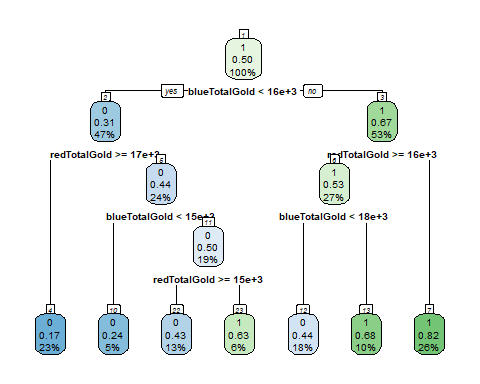
## VarImp names  
## 1 0.7417915 blueTotalGold  
## 2 0.7399140 redTotalGold  
## 3 0.7294051 blueTotalExperience  
## 4 0.7252357 redTotalExperience  
## 5 0.6996078 blueAvgLevel  
## 6 0.6989901 redAvgLevel  
## 7 0.6963795 redKills  
## 8 0.6914970 blueKills  
## 9 0.6596826 blueAssists  
## 10 0.6584104 redAssists  
## 11 0.6302503 blueTotalMinionsKilled  
## 12 0.6234871 redTotalMinionsKilled  
## 13 0.6058094 blueDragons  
## 14 0.6049817 redDragons  
## 15 0.6018770 blueFirstBlood  
## 16 0.5740091 blueTotalJungleMinionsKilled  
## 17 0.5574036 redTotalJungleMinionsKilled  
## 18 0.5399822 redWardsDestroyed  
## 19 0.5376372 redHeralds  
## 20 0.5344257 blueHeralds  
## 21 0.5302263 blueWardsDestroyed  
## 22 0.5263137 blueWardsPlaced  
## 23 0.5255946 redWardsPlaced  
## 24 0.5237321 blueTowersDestroyed  
## 25 0.5214253 redTowersDestroyed

## Árbol de decisión

### Modelo de Árbol de decisión

Se usa la librería [*rpart()*](https://cran.r-project.org/web/packages/rpart/index.html) para hacer un modelo de árbol de decisión.

model\_BT <- rpart(blueWins~.,  
 data = train,  
 method = "class")  
rpart.plot(model\_BT, nn=TRUE)



### Reglas de decisión

En la tabla a continuación se muestran las reglas que sigue el árbol a la hora de decidir.

rules\_BT <- rpart.rules(model\_BT)  
row.names(rules\_BT) <- NULL  
rules\_BT

## blueWins   
## 0.17 when blueTotalGold < 16276 & redTotalGold >= 16814  
## 0.24 when blueTotalGold < 14794 & redTotalGold < 16814   
## 0.43 when blueTotalGold is 14794 to 16276 & redTotalGold is 15474 to 16814  
## 0.44 when blueTotalGold is 16276 to 17626 & redTotalGold >= 15960  
## 0.63 when blueTotalGold is 14794 to 16276 & redTotalGold < 15474   
## 0.68 when blueTotalGold >= 17626 & redTotalGold >= 15960  
## 0.82 when blueTotalGold >= 16276 & redTotalGold < 15960

En el apartado de [métricas](#métricas-del-modelo-de-árbol-de-decisión) haremos más análisis a este modelo.

# Métricas

## Métricas del modelo de Regresión Logística

Matriz de confusión y estadísiticas.

prediction\_GLM <- predict.glm(model\_GLM, test\_asFactor, type = "response")  
absolutes\_LR <- 1\*(prediction\_GLM>=0.5)  
absolutes\_LR <- as.factor(absolutes\_LR)  
metrics\_GLM <- confusionMatrix(test\_asFactor$blueWins, absolutes\_LR)  
metrics\_GLM

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 702 275  
## 1 245 697  
##   
## Accuracy : 0.729   
## 95% CI : (0.7085, 0.7488)  
## No Information Rate : 0.5065   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.4582   
##   
## Mcnemar's Test P-Value : 0.2035   
##   
## Sensitivity : 0.7413   
## Specificity : 0.7171   
## Pos Pred Value : 0.7185   
## Neg Pred Value : 0.7399   
## Prevalence : 0.4935   
## Detection Rate : 0.3658   
## Detection Prevalence : 0.5091   
## Balanced Accuracy : 0.7292   
##   
## 'Positive' Class : 0   
##

acc\_GLM <- accuracy(test$blueWins, absolutes\_LR)  
cat("Exactitud del modelo Regresión Logística: ", acc\_GLM, "%.\n")

## Exactitud del modelo Regresión Logística: 0.7290255 %.

## Métricas del modelo de Naive Bayes

Matriz de confusión y estadísiticas.

prediction\_NB <- predict(model\_NB, newdata = test\_NB )  
metrics\_NB <- confusionMatrix(prediction\_NB, test\_NB$blueWins)  
metrics\_NB

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 665 220  
## 1 312 722  
##   
## Accuracy : 0.7228   
## 95% CI : (0.7022, 0.7427)  
## No Information Rate : 0.5091   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.4463   
##   
## Mcnemar's Test P-Value : 7.968e-05   
##   
## Sensitivity : 0.6807   
## Specificity : 0.7665   
## Pos Pred Value : 0.7514   
## Neg Pred Value : 0.6983   
## Prevalence : 0.5091   
## Detection Rate : 0.3465   
## Detection Prevalence : 0.4612   
## Balanced Accuracy : 0.7236   
##   
## 'Positive' Class : 0   
##

acc\_NB <- accuracy(test\_NB$blueWins, prediction\_NB)  
cat("Exactitud del modelo Naive Bayes: ", acc\_NB, "%.\n")

## Exactitud del modelo Naive Bayes: 0.7227723 %.

## Métricas del modelo de Árbol de decisión

Matriz de confusión y estadísiticas.

prediction\_BT <- predict(object = model\_BT, test[-1], type = "class")  
metrics\_BT <- confusionMatrix(table(test$blueWins, prediction\_BT))  
metrics\_BT

## Confusion Matrix and Statistics  
##   
## prediction\_BT  
## 0 1  
## 0 767 210  
## 1 357 585  
##   
## Accuracy : 0.7045   
## 95% CI : (0.6836, 0.7249)  
## No Information Rate : 0.5857   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.4072   
##   
## Mcnemar's Test P-Value : 8.71e-10   
##   
## Sensitivity : 0.6824   
## Specificity : 0.7358   
## Pos Pred Value : 0.7851   
## Neg Pred Value : 0.6210   
## Prevalence : 0.5857   
## Detection Rate : 0.3997   
## Detection Prevalence : 0.5091   
## Balanced Accuracy : 0.7091   
##   
## 'Positive' Class : 0   
##

acc\_BT <- accuracy(test$blueWins, prediction\_BT)  
cat("Exactitud del modelo Árbol de Decisión: ", acc\_BT, "%.\n")

## Exactitud del modelo Árbol de Decisión: 0.7045336 %.

## Comparación de las métricas

Los 3 modelos tienen una exactitud similar, pero el modelo de [Regresión Logística](#regresión-logística) tiene el valor superior al resto. A continuación una tabla que agrupa las 3 exactitudes.

metrics\_table <- matrix(c(acc\_GLM, acc\_NB, acc\_BT),ncol = 1, byrow = TRUE)  
rownames(metrics\_table) <- c("Regresión Logística", "Naive Bayes", "Binary Tree")  
colnames(metrics\_table) <- c("Exactitud (%)")  
  
metrics\_table <- as.table(metrics\_table)  
metrics\_table

## Exactitud (%)  
## Regresión Logística 0.7290255  
## Naive Bayes 0.7227723  
## Binary Tree 0.7045336

Se utiliza el PCA para hacer un modelo a expensas de encontrar mejores resultados.

# Modelo de Regresión logística con PCA

Gracias a la función pcr() del paquete [*pls*](https://cran.r-project.org/web/packages/pls/pls.pdf) podemos recrear un modelo aplicando Regresión por Componentes Principales, y configurar un modelo acorde.

model\_GLMPCA <- pcr( blueWins ~ .,  
 data = train,  
 scale = TRUE,  
 validation = "CV"  
 )  
summary(model\_GLMPCA)

## Data: X dimension: 7960 25   
## Y dimension: 7960 1  
## Fit method: svdpc  
## Number of components considered: 25  
##   
## VALIDATION: RMSEP  
## Cross-validated using 10 random segments.  
## (Intercept) 1 comps 2 comps 3 comps 4 comps  
## CV 0.5001 0.4279 0.4278 0.4278 0.4268  
## adjCV 0.5001 0.4279 0.4278 0.4278 0.4268  
## 5 comps 6 comps 7 comps 8 comps 9 comps  
## CV 0.4268 0.4268 0.4269 0.427 0.427  
## adjCV 0.4268 0.4268 0.4269 0.427 0.427  
## 10 comps 11 comps 12 comps 13 comps 14 comps  
## CV 0.4271 0.427 0.427 0.427 0.4269  
## adjCV 0.4270 0.427 0.427 0.427 0.4269  
## 15 comps 16 comps 17 comps 18 comps 19 comps  
## CV 0.4269 0.4270 0.4269 0.4270 0.4271  
## adjCV 0.4269 0.4269 0.4269 0.4269 0.4270  
## 20 comps 21 comps 22 comps 23 comps 24 comps  
## CV 0.4271 0.4258 0.4259 0.4259 0.4250  
## adjCV 0.4271 0.4258 0.4258 0.4258 0.4249  
## 25 comps  
## CV 0.4249  
## adjCV 0.4248  
##   
## TRAINING: % variance explained  
## 1 comps 2 comps 3 comps 4 comps 5 comps  
## X 23.58 35.01 42.61 48.85 54.00  
## blueWins 26.81 26.84 26.84 27.20 27.21  
## 6 comps 7 comps 8 comps 9 comps 10 comps  
## X 58.94 63.46 67.87 72.11 75.82  
## blueWins 27.22 27.22 27.22 27.22 27.23  
## 11 comps 12 comps 13 comps 14 comps 15 comps  
## X 79.28 82.57 85.62 88.39 91.12  
## blueWins 27.26 27.26 27.30 27.34 27.36  
## 16 comps 17 comps 18 comps 19 comps 20 comps  
## X 93.73 95.35 96.68 97.83 98.58  
## blueWins 27.37 27.40 27.40 27.40 27.40  
## 21 comps 22 comps 23 comps 24 comps 25 comps  
## X 99.15 99.47 99.74 99.88 100.0  
## blueWins 27.84 27.84 27.85 28.16 28.2

Tal y como se mostraba en [PCA](#análisis-de-componentes-principales), hasta el decimotercer componente no se alcanza un 85.6% de explicación de la varianza.

prediction\_GLMPCA <- predict(model\_GLMPCA, test\_asFactor, ncomp = 13, type = "response")  
absolutes\_LRPCA <- 1\*(prediction\_GLMPCA>=0.5)  
absolutes\_LRPCA <- as.factor(absolutes\_LRPCA)  
metrics\_GLMPCA <- confusionMatrix(test\_asFactor$blueWins, absolutes\_LRPCA)  
metrics\_GLMPCA

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 696 281  
## 1 245 697  
##   
## Accuracy : 0.7259   
## 95% CI : (0.7053, 0.7458)  
## No Information Rate : 0.5096   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.452   
##   
## Mcnemar's Test P-Value : 0.127   
##   
## Sensitivity : 0.7396   
## Specificity : 0.7127   
## Pos Pred Value : 0.7124   
## Neg Pred Value : 0.7399   
## Prevalence : 0.4904   
## Detection Rate : 0.3627   
## Detection Prevalence : 0.5091   
## Balanced Accuracy : 0.7262   
##   
## 'Positive' Class : 0   
##

Se puede comprobar que no es una mejora significativa respecto a ninguno de los anteriores modelos, e incluso empeora con respecto a su semejante, obteniendo menos exactitud.

acc\_GLMPCA <- accuracy(test$blueWins, absolutes\_LRPCA)  
cat("Exactitud del modelo Regresión Logística con PCA: ", acc\_GLMPCA, "%.\n")

## Exactitud del modelo Regresión Logística con PCA: 0.7258989 %.

# Modelo escogido: Regresión Logística

En comparación con el resto de los modelos:

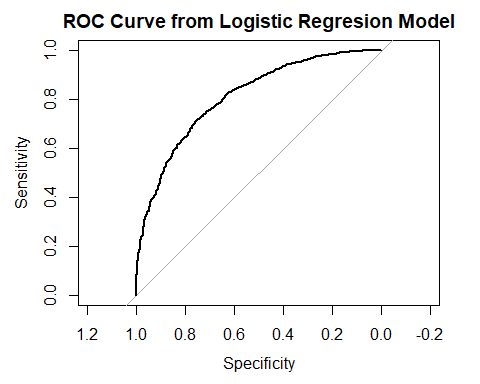
metrics\_table <- matrix(c(acc\_GLM, acc\_GLMPCA, acc\_NB, acc\_BT),ncol = 1, byrow = TRUE)  
rownames(metrics\_table) <- c("Regresión Logística","Regresión Logística con PCA", "Naive Bayes", "Binary Tree")  
colnames(metrics\_table) <- c("Exactitud (%)")  
  
metrics\_table <- as.table(metrics\_table)  
metrics\_table

## Exactitud (%)  
## Regresión Logística 0.7290255  
## Regresión Logística con PCA 0.7258989  
## Naive Bayes 0.7227723  
## Binary Tree 0.7045336

La métrica de exactitud del modelo de Regresión Logística sigue siendo la mejor, y otro motivo para elegir este modelo es también su mayor sensibilidad frente al resto de modelos. Recuperando la [descripción del problema](#descripción-del-problema), se pretendía minimizar los falsos positivos, es decir, los casos que prediga una victoria y resulte ser una derrota, puesto que jugar para acabar perdiendo o sufriendo por una partida agónica y acabar perdiendo es una sensación frustrante, añadiendo el componente del coste de oportunidad de tener tiempo a seguir jugando y suplir la derrota al rendirse.

## Curva ROC del Modelo Escogido

roc\_score = roc(test$blueWins, prediction\_GLM)  
plot(roc\_score, main = "ROC Curve from Logistic Regresion Model")



Se puede ver que es una curva visiblemente alejada de la diagonal, aunque no presente una estructura ideal, es suficiente para mostrar que el modelo es válido.

# Conclusiones

El modelo que más nos conviene sigue siendo el modelo de [Regresión Logística](#regresión-logística), y su curva ROC es buena. Es el más parsimonioso y ninguno de los modelos más complejos tiene mejora significativa. Las variables que más debería priorizar un supuesto *overlay* de *Mobalytics* serían las que vienen dadas por esta tabla de Importancia de Variables en el modelo escogido.

glm\_VarImp

## VarImp names  
## 1 6.75804583 blueTotalGold  
## 2 5.96398995 redTotalGold  
## 3 5.38786020 blueDragons  
## 4 3.94261160 redTotalExperience  
## 5 3.93290681 redDragons  
## 6 3.74834217 blueTotalExperience  
## 7 2.18202304 redTowersDestroyed  
## 8 1.85015571 redTotalMinionsKilled  
## 9 1.80007647 redTotalJungleMinionsKilled  
## 10 1.75824302 blueTowersDestroyed  
## 11 1.37731870 redAssists  
## 12 1.34274259 blueTotalMinionsKilled  
## 13 1.14830573 redHeralds  
## 14 1.05733308 redWardsPlaced  
## 15 0.98824217 blueKills  
## 16 0.96608619 blueAssists  
## 17 0.88826045 blueAvgLevel  
## 18 0.87961095 blueFirstBlood  
## 19 0.85810711 blueTotalJungleMinionsKilled  
## 20 0.80483090 blueWardsPlaced  
## 21 0.25587311 blueHeralds  
## 22 0.19852218 blueWardsDestroyed  
## 23 0.10445804 redWardsDestroyed  
## 24 0.05186443 redAvgLevel  
## 25 0.01180887 redKills

Muchas gracias por leerme.