Tipología y ciclo de vida de los datos

Práctica 2

José Carlos García Pérez

Contents

Descripción del dataset

Se van a usar datos de un proyecto conocido como SkillCraft (http://skillcraft.ca). La idea es aprender cómo desarrollan las personas habilidades complejas estudiando su desempeño en juegos de estrategia en tiempo real, donde hay que gestionar recursos y tomar decisiones estratégicas.

Estos datos corresponden a jugadores de Starcraft 2, un juego donde los jugadores compiten entre sí. Cada jugador elige una raza de entre tres posibles y comienza con seis trabajadores, que pueden usarse para construir fábricas, edificios militares, más trabajadores, etc.

Es un juego que mezcla crecimiento económico y estrategia militar en tiempo real.

En este estudio vamos a analizar qué variables contribuyen más a determinar en qué liga juega un jugador, en función de ciertas medidas de desempeño y, si es posible, trataremos de generar un modelo capaz de ubicar a un jugador en la liga correspondiente en función de estas variables.

Los datos están disponibles en https://archive.ics.uci.edu/ml/datasets/SkillCraft1+Master+Table+Dataset

El dataset consta de los siguientes atributos:

Variable	Descripción
GameID	Identificador único de la partida (integer)
LeagueIndex	Índice de la liga: Bronze, Silver, Gold, Platinum, Diamond, Master, GrandMaster y Professional (valores del 1 al 8)
Age	Edad del jugador (integer)
HoursPerWeek	Horas de juego a la semana (integer)
TotalHours	Horas jugadas en total (integer)
APM	Acciones por minuto (variable continua)
SelectByHotkeys	Número de selecciones de unidades o edificios por hotkeys por unidad de tiempo (variable continua)

Variable	Descripción
AssignToHotkeys	Número de unidades o edificios asignados a hotkeys por unidad de tiempo (variable continua)
UniqueHotkeys	Número de hotkeys únicas usadas por unidad de tiempo (variable continua)
MinimapAttacks	Número de acciones de ataque en minimap por unidad de tiempo (variable continua)
MinimapRightClicks	Número de clics con el botón derecho en el minimap por unidad de tiempo (variable continua)
NumberOfPACs	Número de PACs (Perception Action Cycle) por unidad de tiempo (variable continua)
GapBetweenPACs	Duración media en milisegundos entre PACs (variable continua)
ActionLatency	Latencia media desde el inicio de una PAC hasta su primera acción en milisegundos (variable continua)
ActionsInPAC	Número medio de acciones dentro de cada PAC (variable continua)
TotalMapExplored	El número de cuadrantes de 24x24 examinados por el jugador por unidad de tiempo (variable continua)
WorkersMade	Número de trabajadores entrenados por unidad de tiempo (variable continua)
UniqueUnitsMade	Unidades únicas creadas por unidad de tiempo (variable continua)
ComplexUnitsMade	Número de unidades complejas entrenadas por unidad de tiempo (variable continua)
ComplexAbilitiesUsed	Habilidades que requieren instrucciones específicas usadas por unidad de tiempo (variable continua)

La variable de salida es la variable *LeagueIndex*, que toma los valores de 1 a 8, correspondientes a cada una de las ligas.

Integración y selección de los datos de interés a analizar

Como primer paso y para que los resultados sean reproducibles, inicializaremos la semilla de números aleatorios.

```
set.seed(100)
```

A continuación cargaremos el dataset. Hay que tener en cuenta que el archivo csv debe estar en el directorio de trabajo. En caso contrario hay que especificar la ruta absoluta al archivo.

```
# Leemos los datos
skillData <- read.csv("SkillCraft1_Dataset.csv")
```

Examinaremos si el dataset se ha cargado correctamente, según la especificación de los atributos del apartado anterior.

```
# Echamos un vistazo a los datos
str(skillData)
```

```
## 'data.frame':    3395 obs. of  20 variables:
##  $ GameID              : int  52 55 56 57 58 60 61 72 77 81 ...
##  $ LeagueIndex         : int  5 5 4 3 3 2 1 7 4 4 ...
##  $ Age                 : Factor w/ 29 levels "?","16","17",..: 13 9 16 5 18 13 7 3 6 4 ...
##  $ HoursPerWeek        : Factor w/ 33 levels "?","0","10","112",..: 3 3 3 12 3 24 28 20 6 13 ...
##  $ TotalHours          : Factor w/ 238 levels "?","10","100",..: 123 164 76 146 163 198 97 5 112 2
##  $ APM                 : num  144 129 70 108 123 ...
##  $ SelectByHotkeys     : num  0.00352 0.0033 0.0011 0.00103 0.00114 ...
##  $ AssignToHotkeys     : num  0.00022 0.000259 0.000336 0.000213 0.000327 ...
##  $ UniqueHotkeys       : int  7 4 4 1 2 2 6 6 2 8 ...
##  $ MinimapAttacks      : num  1.10e-04 2.94e-04 2.94e-04 5.33e-05 0.00 ...
```

```
##  $ MinimapRightClicks  : num  0.000392 0.000432 0.000461 0.000543 0.001329 ...
##  $ NumberOfPACs        : num  0.00485 0.00431 0.00293 0.00378 0.00237 ...
##  $ GapBetweenPACs      : num  32.7 32.9 44.6 29.2 22.7 ...
##  $ ActionLatency       : num  40.9 42.3 75.4 53.7 62.1 ...
##  $ ActionsInPAC        : num  4.75 4.84 4.04 4.92 9.37 ...
##  $ TotalMapExplored    : int  28 22 22 19 15 16 15 45 29 27 ...
##  $ WorkersMade         : num  0.001397 0.001193 0.000745 0.000426 0.001174 ...
##  $ UniqueUnitsMade     : int  6 5 6 7 4 6 5 9 7 6 ...
##  $ ComplexUnitsMade    : num  0 0 0 0 0 ...
##  $ ComplexAbilitiesUsed: num  0.00 2.08e-04 1.89e-04 3.84e-04 1.93e-05 ...
```

Como puede observarse, el dataset consta de **3395 observaciones de 20 variables**. las variables *Age*, *HoursPerWeek* y *TotalHours* se han cargado como factores, lo que nos hace sospechar que podría haber valores extraños. Examinemos en detalle estas variables.

```
# Niveles de las variables sospechosas
levels(skillData$Age)
```

```
##  [1] "?"  "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28"
## [15] "29" "30" "31" "32" "33" "34" "35" "36" "37" "38" "39" "40" "41" "43"
## [29] "44"
```

```
levels(skillData$HoursPerWeek)
```

```
##  [1] "?"   "0"   "10"  "112" "12"  "14"  "140" "16"  "168" "18"  "2"
## [12] "20"  "24"  "28"  "30"  "32"  "36"  "4"   "40"  "42"  "48"  "50"
## [23] "56"  "6"   "60"  "70"  "72"  "8"   "80"  "84"  "90"  "96"  "98"
```

```
levels(skillData$TotalHours)
```

```
##   [1] "?"       "10"      "100"     "1000"    "10000"   "1000000" "1008"
##   [8] "1024"    "10260"   "1030"    "105"     "1050"    "1080"    "1095"
##  [15] "1098"    "110"     "1100"    "1145"    "1148"    "1150"    "1155"
##  [22] "1163"    "1170"    "1196"    "12"      "120"     "1200"    "123"
##  [29] "125"     "1250"    "1256"    "1260"    "1270"    "1274"    "128"
##  [36] "130"     "1300"    "1320"    "1344"    "1350"    "138"     "140"
##  [43] "1400"    "144"     "1440"    "1460"    "1466"    "148"     "150"
##  [50] "1500"    "153"     "1560"    "16"      "160"     "1600"    "165"
##  [57] "1650"    "170"     "1700"    "174"     "175"     "176"     "1782"
##  [64] "180"     "1800"    "18000"   "1825"    "184"     "1850"    "1870"
##  [71] "1880"    "190"     "1940"    "196"     "20"      "200"     "2000"
##  [78] "20000"   "2016"    "208"     "21"      "2100"    "216"     "2160"
##  [85] "2190"    "220"     "2200"    "224"     "225"     "2250"    "230"
##  [92] "2300"    "233"     "235"     "238"     "24"      "240"     "2400"
##  [99] "243"     "25"      "250"     "2500"    "25000"   "2520"    "255"
## [106] "256"     "26"      "2600"    "2671"    "270"     "2700"    "2708"
## [113] "275"     "280"     "2800"    "288"     "2880"    "2920"    "2942"
## [120] "3"       "30"      "300"     "3000"    "303"     "310"     "312"
## [127] "315"     "320"     "325"     "3257"    "330"     "333"     "334"
## [134] "336"     "340"     "35"      "350"     "36"      "360"     "364"
## [141] "365"     "366"     "370"     "380"     "40"      "400"     "4000"
## [148] "410"     "415"     "416"     "420"     "425"     "430"     "433"
## [155] "45"      "450"     "460"     "465"     "470"     "477"     "480"
## [162] "50"      "500"     "5000"    "520"     "525"     "530"     "540"
## [169] "546"     "548"     "550"     "551"     "553"     "56"      "560"
## [176] "575"     "577"     "580"     "582"     "60"      "600"     "6000"
## [183] "610"     "612"     "620"     "624"     "625"     "630"     "639"
## [190] "640"     "65"      "650"     "666"     "670"     "690"     "699"
## [197] "7"       "70"      "700"     "711"     "72"      "720"     "727"
```

```
## [204] "730"     "731"     "740"     "745"     "75"      "750"     "755"
## [211] "77"      "770"     "780"     "80"      "800"     "820"     "830"
## [218] "840"     "848"     "85"      "850"     "854"     "865"     "88"
## [225] "898"     "90"      "900"     "9000"    "92"      "936"     "94"
## [232] "95"      "950"     "96"      "960"     "980"     "990"     "999"
```

Las tres columnas *sospechosas* incluyen el carácter **?** para indicar valores vacíos, y son los responsables de que R tome estas variables como si fueran factores. Volveremos a cargar los datos, pero esta vez haciendo la conversión del carácter **?** al valor **NA**.

```
skillData <- read.csv("SkillCraft1_Dataset.csv", na.strings = "?")
```

Examinemos de nuevo las columnas:

```
str(skillData)
```

```
## 'data.frame':    3395 obs. of  20 variables:
##  $ GameID              : int  52 55 56 57 58 60 61 72 77 81 ...
##  $ LeagueIndex         : int  5 5 4 3 3 2 1 7 4 4 ...
##  $ Age                 : int  27 23 30 19 32 27 21 17 20 18 ...
##  $ HoursPerWeek        : int  10 10 10 20 10 6 8 42 14 24 ...
##  $ TotalHours          : int  3000 5000 200 400 500 70 240 10000 2708 800 ...
##  $ APM                 : num  144 129 70 108 123 ...
##  $ SelectByHotkeys     : num  0.00352 0.0033 0.0011 0.00103 0.00114 ...
##  $ AssignToHotkeys     : num  0.00022 0.000259 0.000336 0.000213 0.000327 ...
##  $ UniqueHotkeys       : int  7 4 4 1 2 2 6 6 2 8 ...
##  $ MinimapAttacks      : num  1.10e-04 2.94e-04 2.94e-04 5.33e-05 0.00 ...
##  $ MinimapRightClicks  : num  0.000392 0.000432 0.000461 0.000543 0.001329 ...
##  $ NumberOfPACs        : num  0.00485 0.00431 0.00293 0.00378 0.00237 ...
##  $ GapBetweenPACs      : num  32.7 32.9 44.6 29.2 22.7 ...
##  $ ActionLatency       : num  40.9 42.3 75.4 53.7 62.1 ...
##  $ ActionsInPAC        : num  4.75 4.84 4.04 4.92 9.37 ...
##  $ TotalMapExplored    : int  28 22 22 19 15 16 15 45 29 27 ...
##  $ WorkersMade         : num  0.001397 0.001193 0.000745 0.000426 0.001174 ...
##  $ UniqueUnitsMade     : int  6 5 6 7 4 6 5 9 7 6 ...
##  $ ComplexUnitsMade    : num  0 0 0 0 0 ...
##  $ ComplexAbilitiesUsed: num  0.00 2.08e-04 1.89e-04 3.84e-04 1.93e-05 ...
```

En esta ocasión R sí cargó correctamente los datos, veamos el resumen estadístico de las variables anteriores:

```
summary(subset(skillData, select = c(Age, HoursPerWeek, TotalHours)))
```

```
##       Age          HoursPerWeek      TotalHours
##  Min.   :16.00   Min.   :  0.00   Min.   :     3.0
##  1st Qu.:19.00   1st Qu.:  8.00   1st Qu.:   300.0
##  Median :21.00   Median : 12.00   Median :   500.0
##  Mean   :21.65   Mean   : 15.91   Mean   :   960.4
##  3rd Qu.:24.00   3rd Qu.: 20.00   3rd Qu.:   800.0
##  Max.   :44.00   Max.   :168.00   Max.   :1000000.0
##  NA's   :55      NA's   :56       NA's   :57
```

Efectivamente, esta vez sí que tenemos los valores vacíos como NA.

Si prestamos atención a los datos, hay una variable que no nos servirá en nuestro análisis, y es la variable *GameID*, ya que esta variable es un identificador único y es distinta por cada fila.

La eliminaremos antes de pasar al siguiente punto.

```
# Eliminamos el id de la partida
skillData$GameID <- NULL
```

Limpieza de los datos

Como vimos en el apartado anterior el dataset cuenta con valores vacíos:

- Age: 55
- HoursPerWeek: 56
- TotalHours: 57

El número de observaciones con valores vacíos es muy pequeño en relación con el total de observaciones, pero aún así se van a imputar sus valores usando la técnica de los k vecinos más próximos. Para ello, se usará la implementación del paquete *VIM*.

```
library(VIM)
# La función kNN genera una nueva columna lógica que
# indica si se han imputado valores o no
skillData <- kNN(skillData, variable = c('Age', 'HoursPerWeek', 'TotalHours'))
```

De hecho, ya no existe ningún valor vacío. Han sido imputados usando los valores de otras observaciones similares, usando la distancia de Gower.

- Age: 0
- HoursPerWeek: 0
- TotalHours: 0

Para detectar la presencia de valores atípicos examinaremos primero el resumen de los cinco números de Tukey, donde podremos obtener un pequeño análisis descriptivo de los datos. Si la media y la mediana están muy separadas, examinaremos más en detalle las variables para detectar los valores extremos.

```
# Eliminamos las columnas incluidas por la función kNN, que
# indican dónde se imputaron valores
skillData <- subset(skillData, select = c(-Age_imp, -HoursPerWeek_imp, -TotalHours_imp))

summary(skillData)
```

```
##   LeagueIndex          Age          HoursPerWeek      TotalHours
##  Min.   :1.000   Min.   :16.00   Min.   :  0.00   Min.   :      3.0
##  1st Qu.:3.000   1st Qu.:18.00   1st Qu.:  8.00   1st Qu.:    300.0
##  Median :4.000   Median :21.00   Median : 12.00   Median :    500.0
##  Mean   :4.184   Mean   :21.59   Mean   : 16.03   Mean   :    959.9
##  3rd Qu.:5.000   3rd Qu.:24.00   3rd Qu.: 20.00   3rd Qu.:    800.0
##  Max.   :8.000   Max.   :44.00   Max.   :168.00   Max.   :1000000.0
##       APM          SelectByHotkeys    AssignToHotkeys     UniqueHotkeys
##  Min.   : 22.06   Min.   :0.000000   Min.   :0.0000000   Min.   : 0.000
##  1st Qu.: 79.90   1st Qu.:0.001258   1st Qu.:0.0002042   1st Qu.: 3.000
##  Median :108.01   Median :0.002500   Median :0.0003526   Median : 4.000
##  Mean   :117.05   Mean   :0.004299   Mean   :0.0003736   Mean   : 4.365
##  3rd Qu.:142.79   3rd Qu.:0.005133   3rd Qu.:0.0004988   3rd Qu.: 6.000
##  Max.   :389.83   Max.   :0.043088   Max.   :0.0017522   Max.   :10.000
##  MinimapAttacks      MinimapRightClicks   NumberOfPACs
##  Min.   :0.000e+00   Min.   :0.0000000   Min.   :0.000679
##  1st Qu.:0.000e+00   1st Qu.:0.0001401   1st Qu.:0.002754
##  Median :3.993e-05   Median :0.0002815   Median :0.003395
##  Mean   :9.831e-05   Mean   :0.0003874   Mean   :0.003463
##  3rd Qu.:1.189e-04   3rd Qu.:0.0005141   3rd Qu.:0.004027
##  Max.   :3.019e-03   Max.   :0.0040408   Max.   :0.007971
##  GapBetweenPACs    ActionLatency     ActionsInPAC    TotalMapExplored
##  Min.   :  6.667   Min.   : 24.09   Min.   : 2.039   Min.   : 5.00
##  1st Qu.: 28.958   1st Qu.: 50.45   1st Qu.: 4.273   1st Qu.:17.00
##  Median : 36.724   Median : 60.93   Median : 5.096   Median :22.00
##  Mean   : 40.362   Mean   : 63.74   Mean   : 5.273   Mean   :22.13
```

```
##  3rd Qu.: 48.291   3rd Qu.: 73.68   3rd Qu.: 6.034   3rd Qu.:27.00
##  Max.   :237.143   Max.   :176.37   Max.   :18.558   Max.   :58.00
##   WorkersMade        UniqueUnitsMade  ComplexUnitsMade
##  Min.   :7.698e-05   Min.   : 2.000   Min.   :0.000e+00
##  1st Qu.:6.830e-04   1st Qu.: 5.000   1st Qu.:0.000e+00
##  Median :9.052e-04   Median : 6.000   Median :0.000e+00
##  Mean   :1.032e-03   Mean   : 6.534   Mean   :5.943e-05
##  3rd Qu.:1.259e-03   3rd Qu.: 8.000   3rd Qu.:8.554e-05
##  Max.   :5.149e-03   Max.   :13.000   Max.   :9.023e-04
##  ComplexAbilitiesUsed
##  Min.   :0.000e+00
##  1st Qu.:0.000e+00
##  Median :2.028e-05
##  Mean   :1.419e-04
##  3rd Qu.:1.814e-04
##  Max.   :3.084e-03
```

Parece que hay muchas variables que llaman la atención, como por ejemplo *TotalHours*, donde la mediana es 500 horas y sin embargo la media es 959.8860088 horas. Veamos cuáles son estos valores extremos, y si podría deberse a un error o son valores perfectamente válidos.

Los ceros que observamos en las columnas *HoursPerWeek*, *SelectByHotkeys*, *AssignToHotkeys*, *UniqueHotkeys*, *MinimapAttacks*, *MinimapRightClicks*, *ComplexUnitsMade* y *ComplexAbilitiesUsed* son valores válidos.

```
# La fila out indica el número de valores atípicos por columma
sapply(skillData, boxplot.stats)
```

```
##       LeagueIndex Age        HoursPerWeek TotalHours  APM
## stats Numeric,5   Numeric,5  Numeric,5    Numeric,5   Numeric,5
## n     3395        3395       3395         3395        3395
## conf  Numeric,2   Numeric,2  Numeric,2    Numeric,2   Numeric,2
## out   Integer,0   Integer,62 Integer,171  Integer,170 Numeric,104
##       SelectByHotkeys AssignToHotkeys UniqueHotkeys MinimapAttacks
## stats Numeric,5       Numeric,5       Numeric,5     Numeric,5
## n     3395            3395            3395          3395
## conf  Numeric,2       Numeric,2       Numeric,2     Numeric,2
## out   Numeric,287     Numeric,62      Integer,0     Numeric,295
##       MinimapRightClicks NumberOfPACs GapBetweenPACs ActionLatency
## stats Numeric,5          Numeric,5    Numeric,5      Numeric,5
## n     3395               3395         3395           3395
## conf  Numeric,2          Numeric,2    Numeric,2      Numeric,2
## out   Numeric,187        Numeric,58   Numeric,108    Numeric,89
##       ActionsInPAC TotalMapExplored WorkersMade UniqueUnitsMade
## stats Numeric,5    Numeric,5        Numeric,5   Numeric,5
## n     3395         3395             3395        3395
## conf  Numeric,2    Numeric,2        Numeric,2   Numeric,2
## out   Numeric,85   Integer,45       Numeric,145 Integer,4
##       ComplexUnitsMade ComplexAbilitiesUsed
## stats Numeric,5        Numeric,5
## n     3395             3395
## conf  Numeric,2        Numeric,2
## out   Numeric,361      Numeric,303
```

Las columnas *Age*, *HoursPerWeek*, *TotalHours*, *APM*, *SelectByHotkeys*, *AssignToHotkeys*, *MinimapAttacks*, *MinimapRightClicks*, *NumberOfPACs*, *GapBetweenPACs*, *ActionLatency*, *ActionsInPAC*, *TotalMapExplored*, *WorkersMade*, *UniqueUnitsMade*, *ComplexUnitsMade* y *ComplexAbilitiesUsed* presentan valores extremos (columna *out*). En principio, salvo en los casos de *HoursPerWeek* y *TotalHours*, consideraremos que los demás valores son correctos, ya que lógicamente hay jugadores que tienen más

experiencia que otros.

En el caso de *HoursPerWeek*, el valor máximo es 168, que justamente es el número de horas de una semana, por lo que consideraremos este valor incorrecto, ya que implicaría que el jugador está activo las 24 horas del día, los 7 días de la semana. De hecho, el 99% de los datos están por debajo de 56 horas, que será el umbral que usaremos para seleccionar las observaciones que consideraremos correctas para este estudio.

```
ninetyNinePercentile <-quantile(skillData$HoursPerWeek, 0.99)

cleanSkillData <- skillData[skillData$HoursPerWeek <= ninetyNinePercentile[[1]], ]
```

En el caso de *TotalHours* seguiremos una estrategia similar.

```
ninetyNinePercentile <-quantile(skillData$TotalHours, 0.99)

cleanSkillData <- skillData[skillData$TotalHours <= ninetyNinePercentile[[1]], ]
```

El dataset contiene ahora 3362 filas. Generaremos un nuevo csv con este conjunto.

```
write.csv(cleanSkillData, "cleanSkillCraftData.csv", row.names = FALSE)
```

Análisis de los datos

En nuestro caso, el objetivo es detectar las variables que más contribuyen a explicar la liga en la que está ubicado el jugador y, si es posible, generar un modelo predictivo y/o de clasificación para ubicar a un jugador en una liga según las variables de su desempeño en el juego. Para ello, dividiremos el dataset en dos conjuntos, uno para entrenamiento y generación del modelo, y otro para su evaluación. Se usará el paquete **caret** para seleccionar el 80% de los datos para entrenamiento y el restante 20% para evaluación.

```
library(caret)

index <- createDataPartition(cleanSkillData$LeagueIndex, p = 0.8, list = FALSE)

trainSet <- cleanSkillData[index,]
testSet <- cleanSkillData[-index,]
```

Se ha dividido el dataset en dos subconjuntos. Por un lado, un conjunto de entrenamiento de 2691 observaciones y por otro, un conjunto de evaluación con 671 observaciones.

Normalidad y homogeneidad de la varianza

A continuación estudiaremos la normalidad y homogeneidad de la varianza en nuestro conjunto. Para ello se usará el test de normalidad de Anderson-Darling, que básicamente realiza el siguiente contraste de hipótesis:

- H0: No hay diferencias observables entre los datos y la distribución normal
- H1: Existen diferencias observables entre los datos y la distribución normal

```
library(nortest)

sapply(skillData, ad.test)
```

```
##           LeagueIndex
## statistic 72.41094
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
```

```
## data.name "X[[i]]"
##           Age
## statistic 55.75697
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           HoursPerWeek
## statistic 121.4881
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           TotalHours
## statistic 1227.766
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           APM
## statistic 48.41665
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           SelectByHotkeys
## statistic 309.2307
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           AssignToHotkeys
## statistic 26.75282
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           UniqueHotkeys
## statistic 29.11114
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           MinimapAttacks
## statistic 365.415
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           MinimapRightClicks
## statistic 164.2161
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           NumberOfPACs
## statistic 11.76259
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           GapBetweenPACs
## statistic 66.37304
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
```

```
##           ActionLatency
## statistic 37.12531
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           ActionsInPAC
## statistic 33.60604
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           TotalMapExplored
## statistic 12.92975
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           WorkersMade
## statistic 92.37054
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           UniqueUnitsMade
## statistic 42.48036
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           ComplexUnitsMade
## statistic 541.1089
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
##           ComplexAbilitiesUsed
## statistic 428.8086
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"
```

Si nos fijamos en los p-valores obtenidos por cada columna, son mucho menores que el nivel de significación por defecto, 0.05, por lo que no aceptaríamos la hipótesis nula y aceptaríamos que las variables no siguen una distribución normal.

Adicionalmente, aplicaremos el test de normalidad de Shapiro-Wilk.

```
sapply(skillData, shapiro.test)
```

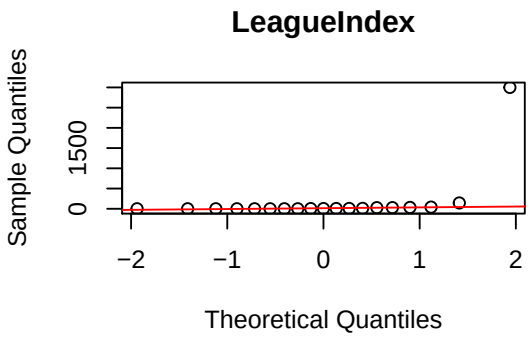
```
##           LeagueIndex                   Age
## statistic 0.9468336                     0.9196004
## p.value   2.247775e-33                  3.085481e-39
## method    "Shapiro-Wilk normality test" "Shapiro-Wilk normality test"
## data.name "X[[i]]"                      "X[[i]]"
##           HoursPerWeek                  TotalHours
## statistic 0.8107521                     0.009286868
## p.value   6.008563e-53                  1.69065e-85
## method    "Shapiro-Wilk normality test" "Shapiro-Wilk normality test"
## data.name "X[[i]]"                      "X[[i]]"
##           APM                           SelectByHotkeys
## statistic 0.9291169                     0.6779926
## p.value   2.125816e-37                  1.567685e-62
## method    "Shapiro-Wilk normality test" "Shapiro-Wilk normality test"
```

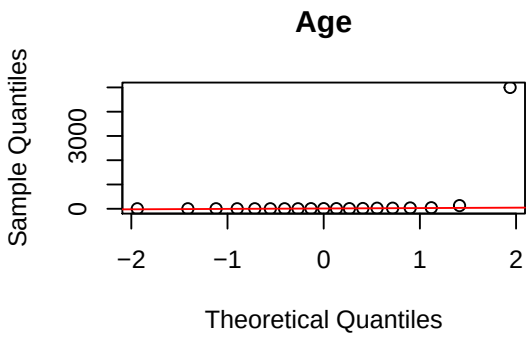
```
## data.name "X[[i]]"                      "X[[i]]"
##           AssignToHotkeys               UniqueHotkeys
## statistic 0.9379782                     0.9710874
## p.value   1.682394e-35                  1.213896e-25
## method    "Shapiro-Wilk normality test" "Shapiro-Wilk normality test"
## data.name "X[[i]]"                      "X[[i]]"
##           MinimapAttacks                MinimapRightClicks
## statistic 0.5863291                     0.7851274
## p.value   2.477916e-67                  3.662189e-55
## method    "Shapiro-Wilk normality test" "Shapiro-Wilk normality test"
## data.name "X[[i]]"                      "X[[i]]"
##           NumberOfPACs                  GapBetweenPACs
## statistic 0.9836737                     0.8869495
## p.value   2.309147e-19                  1.929383e-44
## method    "Shapiro-Wilk normality test" "Shapiro-Wilk normality test"
## data.name "X[[i]]"                      "X[[i]]"
##           ActionLatency                 ActionsInPAC
## statistic 0.9405905                     0.9168067
## p.value   6.708599e-35                  9.615285e-40
## method    "Shapiro-Wilk normality test" "Shapiro-Wilk normality test"
## data.name "X[[i]]"                      "X[[i]]"
##           TotalMapExplored              WorkersMade
## statistic 0.9762768                     0.8829352
## p.value   2.314579e-23                  5.415569e-45
## method    "Shapiro-Wilk normality test" "Shapiro-Wilk normality test"
## data.name "X[[i]]"                      "X[[i]]"
##           UniqueUnitsMade               ComplexUnitsMade
## statistic 0.9715586                     0.6129533
## p.value   1.896182e-25                  4.863375e-66
## method    "Shapiro-Wilk normality test" "Shapiro-Wilk normality test"
## data.name "X[[i]]"                      "X[[i]]"
##           ComplexAbilitiesUsed
## statistic 0.5810706
## p.value   1.403299e-67
## method    "Shapiro-Wilk normality test"
## data.name "X[[i]]"
```

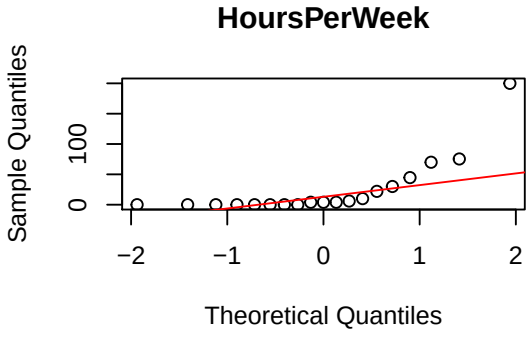
Al igual que con el test de Anderson-Darling, podemos confirmar que las variables no tienen una distribución normal.

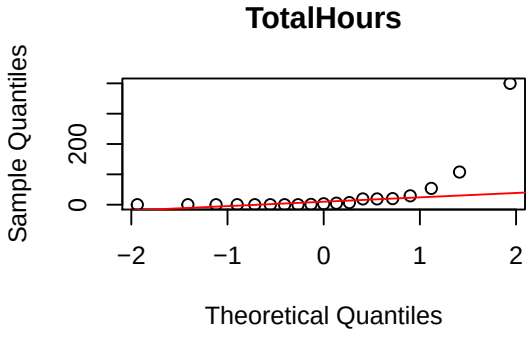
Gráficamente, podemos usar las gráficas *quantile-quantile*.

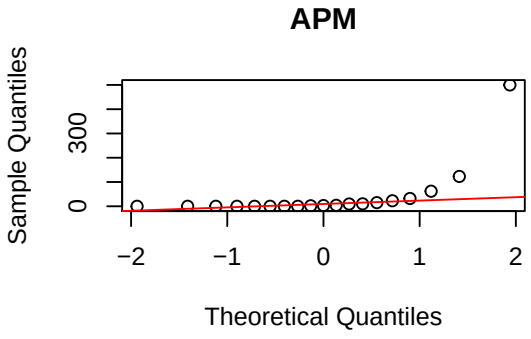
```r
par(mfrow=c(2,2))

for (col in 1:ncol(skillData)) {
  qqnorm(skillData[col, ], main = colnames(skillData)[col])
  qqline(skillData[col, ], col = "red")
}
```

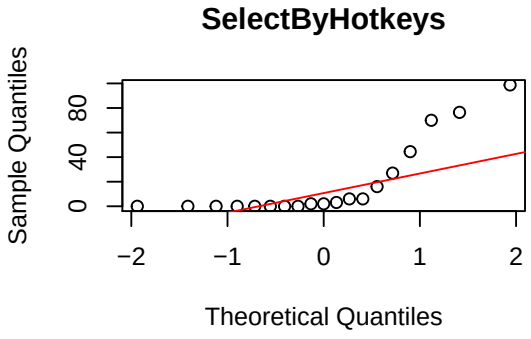

LeagueIndex
Sample Quantiles
0
1500
−2
−1
0
1
2
Theoretical Quantiles

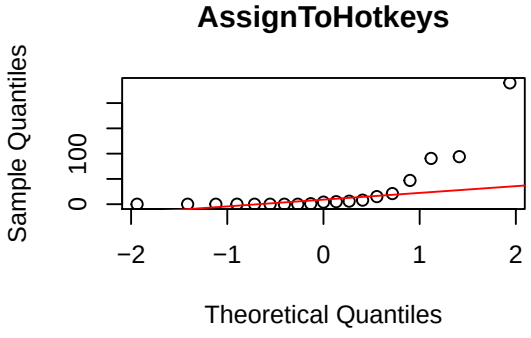

Age
Sample Quantiles
0
3000
−2
−1
0
1
2
Theoretical Quantiles

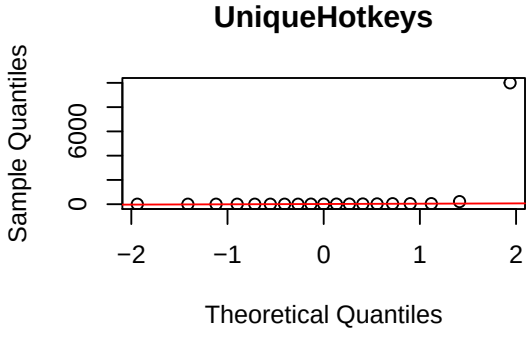

HoursPerWeek
Sample Quantiles
0
100
−2
−1
0
1
2
Theoretical Quantiles

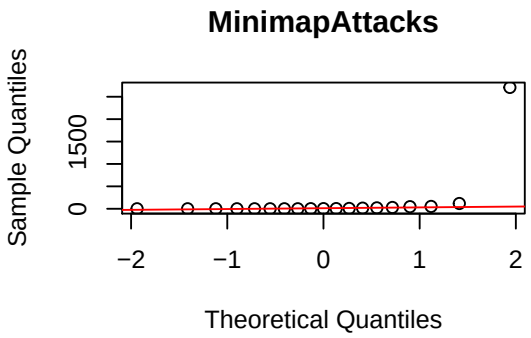

TotalHours
Sample Quantiles
0
200
−2
−1
0
1
2
Theoretical Quantiles

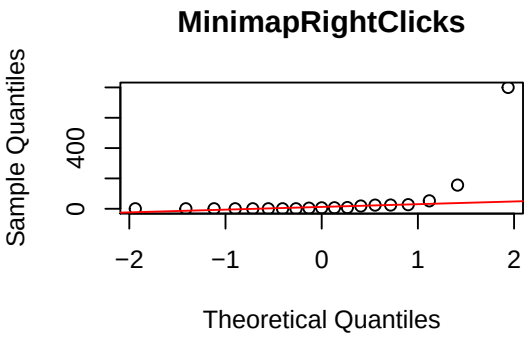

APM
Sample Quantiles
0
300
−2
−1
0
1
2
Theoretical Quantiles

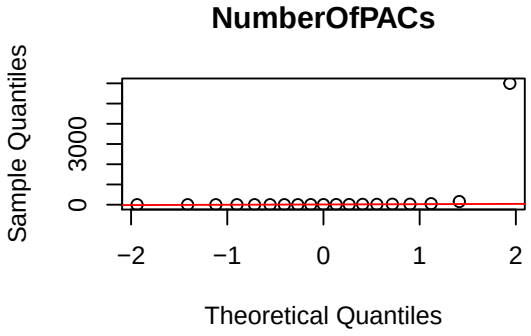

SelectByHotkeys
Sample Quantiles
0
40
80
−2
−1
0
1
2
Theoretical Quantiles

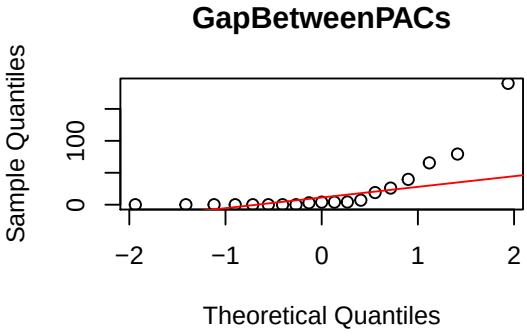

AssignToHotkeys
Sample Quantiles
0
100
−2
−1
0
1
2
Theoretical Quantiles

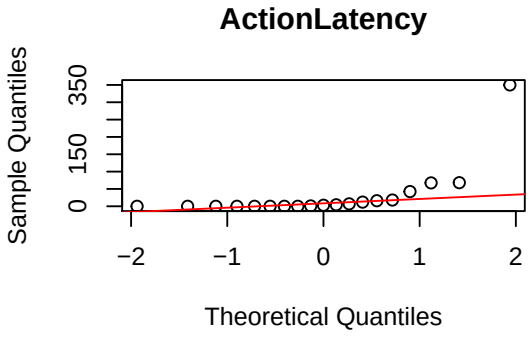

UniqueHotkeys
Sample Quantiles
0
6000
−2
−1
0
1
2
Theoretical Quantiles

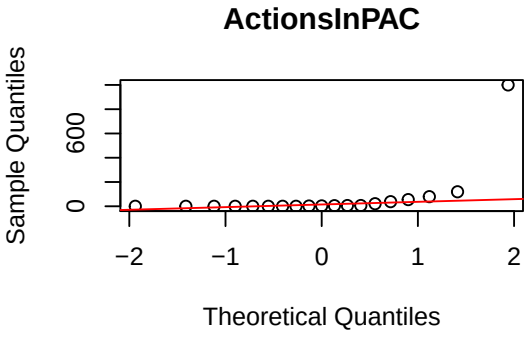

MinimapAttacks
Sample Quantiles
0
1500
−2
−1
0
1
2
Theoretical Quantiles

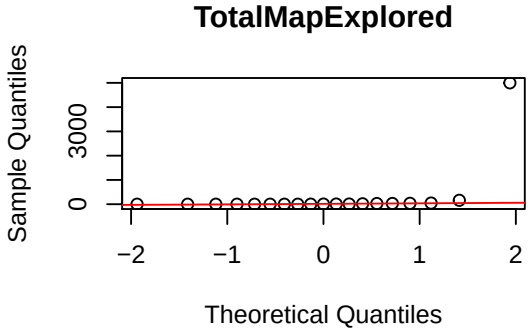

MinimapRightClicks
Sample Quantiles
0
400
−2
−1
0
1
2
Theoretical Quantiles

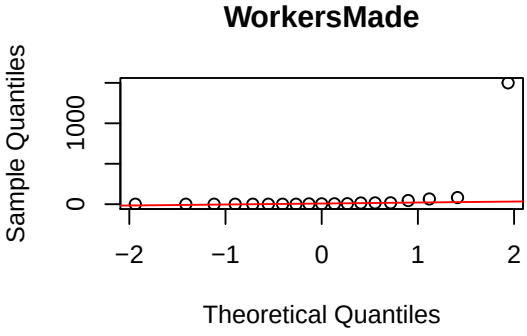

NumberOfPACs
Sample Quantiles
0
3000
−2
−1
0
1
2
Theoretical Quantiles

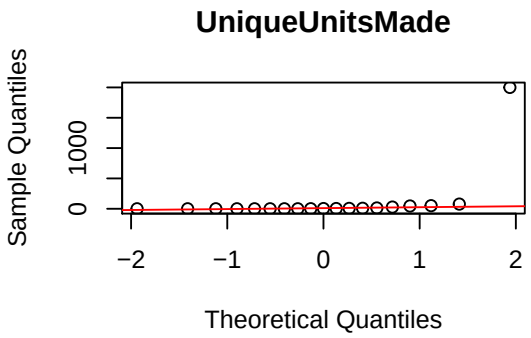

GapBetweenPACs
Sample Quantiles
0
100
−2
−1
0
1
2
Theoretical Quantiles

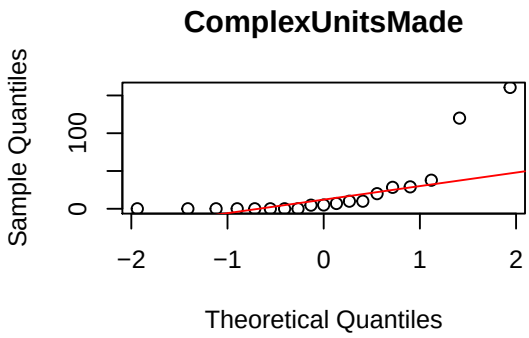

ActionLatency
Sample Quantiles
0
150
350
−2
−1
0
1
2
Theoretical Quantiles

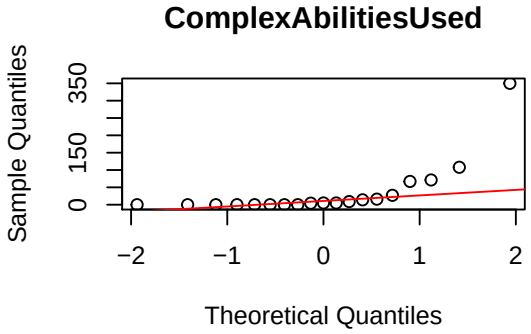

ActionsInPAC
Sample Quantiles
0
600
−2
−1
0
1
2
Theoretical Quantiles


TotalMapExplored
Sample Quantiles
0
3000
−2
−1
0
1
2
Theoretical Quantiles


WorkersMade
Sample Quantiles
0
1000
−2
−1
0
1
2
Theoretical Quantiles







En nuestro estudio, podría ser interesante estudiar la homogeneidad de las varianzas por edad. Para ello crearemos cuatro grupos y aplicaremos el test de Fligner-Killeen, que es una buena opción con datos no normales y con presencia de *outliers*.

```
skillData$AgeRange <- cut(skillData$Age, 4)

fligner.test(LeagueIndex ~ AgeRange, data = skillData)
```

```
##
##  Fligner-Killeen test of homogeneity of variances
##
## data:  LeagueIndex by AgeRange
## Fligner-Killeen:med chi-squared = 21.201, df = 3, p-value =
## 9.562e-05
```

El valor-p es muy inferior a 0.05, por lo que no aceptaríamos la hipótesis de que las varianzas son homogéneas.

Correlación con la variable dependiente

A continuación obtendremos las variables con mayor correlación con la variable dependiente. Para ello usaremos la función *findCorrelation* del paquete *caret*, que a partir de la matriz de correlación nos devolverá las variables con un coeficiente de correlación superior a 0.7.

```
correlationMatrix <- cor(cleanSkillData)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff = 0.7)

colnames(cleanSkillData)[highlyCorrelated]
```

```
## [1] "APM"           "ActionLatency"
```

Modelo de regresión lineal

Crearemos un modelo de regresión lineal con los dos predictores anteriores, usando el conjunto de entrenamiento (*train set*). Para su evaluación, se usará el *test set*.

```
linearModel <- lm(LeagueIndex ~ APM + ActionLatency, data = trainSet)

linearModelSummary <- summary(linearModel)

linearModelSummary
```

```
##
## Call:
## lm(formula = LeagueIndex ~ APM + ActionLatency, data = trainSet)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1831 -0.6896  0.0442  0.7569  2.9885
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4.9050737  0.1531579   32.03   <2e-16 ***
## APM            0.0109392  0.0005635   19.41   <2e-16 ***
## ActionLatency -0.0315805  0.0015219  -20.75   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.058 on 2688 degrees of freedom
## Multiple R-squared:  0.5204, Adjusted R-squared:   0.52
## F-statistic:  1458 on 2 and 2688 DF,  p-value: < 2.2e-16
```

Para obtener una métrica del error obtenido, se creará una función que calcule la raíz cuadrada del error cuadrático medio. Cuanto más bajo, mejor. En este caso, nos indica en promedio el número de ligas que nos estamos desviando en las predicciones.

```
rmse <- function(actual, predicted) {
    sqrt(mean((actual - predicted)^2))
}

linearModelPredictions <- predict(linearModel, testSet)
linearModelError <- rmse(testSet$LeagueIndex, linearModelPredictions)

linearModelError
```

```
## [1] 1.037184
```

Modelo de clasificación

Con el modelo anterior cometemos un error en promedio de 1.0371838 ligas. Dado que la variable objetivo es un valor discreto (aunque numérico), podría ser una buena opción abordar el problema como un problema de clasificación, en lugar de regresión. Para ello generaremos un nuevo modelo, basado en árboles de decisión.

```
library(C50)

# Se convierte la variable objetivo a factor para discretizarla
trainSet$LeagueIndex <- as.factor(trainSet$LeagueIndex)
testSet$LeagueIndex <- as.factor(testSet$LeagueIndex)
```

```r
# Se usan todas las variables predictoras
treeModel <- C5.0(LeagueIndex ~ ., data = trainSet)

treeModelPredictions <- predict(treeModel, testSet)
```

Resultados

La aproximación como problema de regresión lineal arroja un error de 1.0371838 ligas. La siguiente tabla muestra el resultado del modelo regresión lineal en el *test set* (sólo se muestran las primeras predicciones).

```r
linearModelDF <- data.frame("Valor real" = testSet$LeagueIndex,
                            "Predicción" = linearModelPredictions)

head(linearModelDF)
```

```
##    Valor.real Predicción
## 13          3   3.494142
## 20          4   4.267843
## 27          6   6.549708
## 28          5   3.447834
## 30          6   5.111546
## 31          3   3.139626
```

Al discretizar la variable objetivo, ya que sólo puede tomar valores del 1 al 8, y evaluar el modelo en *train set*, obtenemos la siguiente matriz de confusión (en el conjunto de entrenamiento):

```r
confusionMatrix(trainSet$LeagueIndex, predict(treeModel, trainSet))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3   4   5   6   7   8
##          1 119   6   2  12   1   0   0   0
##          2   7 225  19  15  12   1   0   0
##          3   7   9 395  12   9   1   0   0
##          4   1   7  34 572  23   8   0   0
##          5   6   5  11  25 559  26   1   0
##          6   3   0  11  12  19 444   2   0
##          7   0   1   1   3   2   3  14   1
##          8   0   0   0   1   0   3   1  40
##
## Overall Statistics
##
##                Accuracy : 0.88
##                  95% CI : (0.8671, 0.892)
##     No Information Rate : 0.2423
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8524
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.83217  0.88933   0.8351   0.8773   0.8944   0.9136
```

```
## Specificity           0.99176  0.97785   0.9829   0.9642   0.9642   0.9787
## Pos Pred Value        0.85000  0.80645   0.9122   0.8868   0.8831   0.9043
## Neg Pred Value        0.99059  0.98839   0.9655   0.9609   0.9679   0.9809
## Prevalence            0.05314  0.09402   0.1758   0.2423   0.2323   0.1806
## Detection Rate        0.04422  0.08361   0.1468   0.2126   0.2077   0.1650
## Detection Prevalence  0.05203  0.10368   0.1609   0.2397   0.2352   0.1825
## Balanced Accuracy     0.91196  0.93359   0.9090   0.9207   0.9293   0.9461
##                      Class: 7 Class: 8
## Sensitivity          0.777778  0.97561
## Specificity          0.995885  0.99811
## Pos Pred Value       0.560000  0.88889
## Neg Pred Value       0.998500  0.99962
## Prevalence           0.006689  0.01524
## Detection Rate       0.005203  0.01486
## Detection Prevalence 0.009290  0.01672
## Balanced Accuracy    0.886831  0.98686
```

Si comparamos con el error que obtenemos en el conjunto de evaluación **probablemente estemos ante un problema de sobreajuste**, ya que la precisión en el *test set* es muchísima peor. O podría ser que el conjunto de entrenamiento no es representativo del total, aunque la función *createDataPartition* tiene en cuenta este hecho para hacer una partición estratificada. El árbol de decisión generado es capaz de explicar los datos de entrenamiento, pero no generaliza bien:

```
confusionMatrix(testSet$LeagueIndex, treeModelPredictions)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5  6  7  8
##          1  5  9  8  5  0  0  0  0
##          2  6 14 25 17  5  1  0  0
##          3  7 11 41 41 15  2  0  0
##          4  9 12 28 63 30 19  0  0
##          5  0  4 24 45 53 31  1  0
##          6  1  1  8 19 32 59  2  1
##          7  0  0  0  0  0  3  3  1
##          8  0  0  0  0  0  4  0  6
##
## Overall Statistics
##
##                Accuracy : 0.3636
##                  95% CI : (0.3272, 0.4013)
##     No Information Rate : 0.2832
##     P-Value [Acc > NIR] : 3.762e-06
##
##                   Kappa : 0.2121
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity          0.178571  0.27451   0.3060  0.33158  0.39259  0.49580
## Specificity          0.965785  0.91290   0.8585  0.79626  0.80410  0.88406
## Pos Pred Value       0.185185  0.20588   0.3504  0.39130  0.33544  0.47967
## Neg Pred Value       0.964286  0.93864   0.8321  0.75098  0.84016  0.89051
## Prevalence           0.041729  0.07601   0.1997  0.28316  0.20119  0.17735
## Detection Rate       0.007452  0.02086   0.0611  0.09389  0.07899  0.08793
## Detection Prevalence 0.040238  0.10134   0.1744  0.23994  0.23547  0.18331
```

```
## Balanced Accuracy    0.572178  0.59371   0.5822  0.56392  0.59835  0.68993
##                      Class: 7 Class: 8
## Sensitivity          0.500000 0.750000
## Specificity          0.993985 0.993967
## Pos Pred Value       0.428571 0.600000
## Neg Pred Value       0.995482 0.996974
## Prevalence           0.008942 0.011923
## Detection Rate       0.004471 0.008942
## Detection Prevalence 0.010432 0.014903
## Balanced Accuracy    0.746992 0.871983
```

Por último, trataremos de mejorar el resultado con *random forests*, una técnica más compleja basada en árboles de decisión.

```
library(randomForest)

rfModel <- randomForest(LeagueIndex ~ ., data = trainSet)

rfModelPredictions <- predict(rfModel, testSet, type = "class")
```

Examinemos el resultado evaluando el modelo basado en *random forests*:

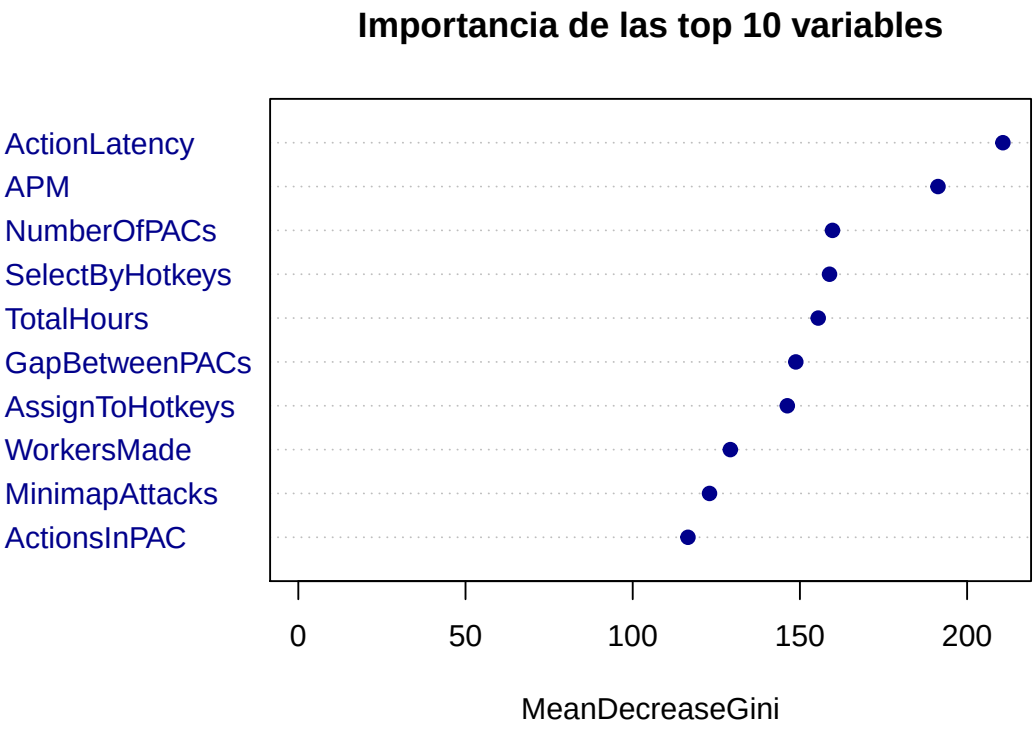
```
confusionMatrix(testSet$LeagueIndex, rfModelPredictions)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5  6  7  8
##          1  6 12  7  2  0  0  0  0
##          2  3 16 32 17  0  0  0  0
##          3  3 10 31 58 15  0  0  0
##          4  1  7 27 74 40 12  0  0
##          5  0  1  6 48 76 27  0  0
##          6  0  0  2 13 44 63  0  1
##          7  0  0  0  0  0  7  0  0
##          8  0  0  0  0  0  3  0  7
##
## Overall Statistics
##
##                Accuracy : 0.4069
##                  95% CI : (0.3694, 0.4451)
##     No Information Rate : 0.3159
##     P-Value [Acc > NIR] : 4.304e-07
##
##                   Kappa : 0.2558
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity          0.461538  0.34783   0.2952   0.3491   0.4343  0.56250
## Specificity          0.968085  0.91680   0.8481   0.8105   0.8347  0.89267
## Pos Pred Value       0.222222  0.23529   0.2650   0.4596   0.4810  0.51220
## Neg Pred Value       0.989130  0.95025   0.8664   0.7294   0.8070  0.91058
## Prevalence           0.019374  0.06855   0.1565   0.3159   0.2608  0.16692
## Detection Rate       0.008942  0.02385   0.0462   0.1103   0.1133  0.09389
## Detection Prevalence 0.040238  0.10134   0.1744   0.2399   0.2355  0.18331
## Balanced Accuracy    0.714812  0.63231   0.5716   0.5798   0.6345  0.72758
```

```
##                      Class: 7 Class: 8
## Sensitivity               NA  0.87500
## Specificity           0.98957  0.99548
## Pos Pred Value            NA  0.70000
## Neg Pred Value            NA  0.99849
## Prevalence            0.00000  0.01192
## Detection Rate        0.00000  0.01043
## Detection Prevalence  0.01043  0.01490
## Balanced Accuracy         NA  0.93524
```

Aunque supone una mejora con respecto al modelo anterior, detecta muy pocos verdaderos positivos, salvo quizá para la liga 8. Según este modelo, las variables más relevantes son las siguientes:

```
varImpPlot(rfModel, sort = TRUE, n.var = 10, main="Importancia de las top 10 variables",
           col="dark blue", pch=19)
```



Conclusiones

El estudio anterior nos lleva a pensar que es posible averiguar qué variables son las que más contribuyen a ubicar a un jugador en una liga u otra. De hecho, podríamos afirmar que *APM* y *ActionLatency* son los mejores predictores, como puede observarse tanto en el estudio de correlación como en el resultado del *varImpPlot* del punto anterior.

A pesar de que el resultado con árboles de decisión no ha sido muy bueno, probablemente se pueda mejorar considerablemente si discretizáramos otras variables, como por ejemplo la edad del jugador o las horas de juego. En cualquier caso, un desvío en promedio de 1.0371838 ligas en las predicciones con el modelo lineal nos lleva a pensar que trabajando un poco más el dataset podamos llegar a obtener un modelo lineal más preciso, y mucho más fácil de interpretar que, por ejemplo, el modelo basado en *random forests*.

Referencias

1. Normality tests for continuous data
2. Homogeneity of variance