proyectoTID

Alejandro Casado Quijada y Gustavo Rivas Gervillas

Introducción

Descripción del dataset

Este dataset contiene datos recogidos de la aplicacion *PokemonGo*, esta aplicación es un juego de realidad aumeta que emplea el GPS del móvil para principalmente localizar y capturar pokemon en el mundo real. El dataset contiene 296021 muestras cada una de las cuales dispone de los siguientes campos:

- **pokemonId**: el identificador del pokemon, denota su clase.
- **latitude**: latitud de la posición donde se ha localizado el pokemon.
- **longitude**: longitud de la posición donde se ha localizado el pokemon.
- **appearedLocalTime**: momento exacto en el que se encontró el pokemon, con el formato yyyy-mm-ddThh-mm-ss.ms.
- **X__id**: la ficha del dataset no proporciona información sobre qué representa este dato, de hecho hemos preguntado en la propia web pero no hemos obtenido respuesta. No obstante viendo el dataset no parece ser más que una código identificador de la muestra.
- **cellId 90-5850m**: la localización goegráfica del pokemon proyectada en una celda S2.
- **appearedTimeOfDay**: momento del día en el que apareció el pokemon (night, evening, afternoon, morning).
- **appearedHour**: hora local de una observación del pokemon.
- **appearedMinute**: minuto local de una observación del pokemon.
- **appearedDayOfWeek**: día de la semana en la que se produjo el avistamiento (Monday, Tuesday, Wednesday, Thursday, Friday, Saturdy, Sunday).
- **appearedDay**: día del avistamiento.
- **appearedMonth**: mes del avistamiento.
- **appearedYear**: año del avistamiento.
- **terrainType**: tipo del terreno donde se avistó el pokemon. Este dato viene dado por un valor número según una tabla de tipos de terreno.
- **closeToWater**: si está el pokemon a 100m del agua o no.
- **city**: ciudad donde se ha visto el pokemon.
- **continent**: continente donde se ha avistado el pokemon.
- **weather**: un string indicando el tiempo que hacía en el momento del avistamiento.
- **temperature**: temperatura en grados Celsius en el momento del avistamiento.
- **windSpeed**: velocidad del viento en el momento del avistamiento km/h.
- **windBearing**: dirección del viento entre 0 y 360 grados.
- **pressure**: presión en el momento del avistamiento en bares.
- **weatherIcon**: el tiempo atmosférico en el momento del avistamiento clasificado según un sistema de categorías más simple que el empleado en *weather* (fog, clear-night, partly-cloudy-night, partly-cloudy-day, cloudy, clear-day, rain, wind).
- **sunriseMinutesMidnight**: tiempo de la aparición relativo al amanecer.
- **sunsetMinutesBefore**: tiempo de la aparición relativo a la puesta de sol.
- **population density**: densidad de población por km^2 en un avistamiento.
- **urbal-rural**: cómo de urbana es la localización donde apareció el pokemon relativa a la *population density* (<200 rural, >= 200 && < 400 midUrban, >= 400 && < 800 subUrban, >800 urban).
- **gymDistanceKm**: distancia al gimnasio más cercano al punto de aparición del pokemon.
- **pokestopDistanceKm**: distancia a la pokestop más cercana al punto de aparicion del pokemon.
- **gymIn100m - pokestopIn5000m**: son atributos booleanos que indican si hay un gimnasio o una pokestop a 100m/250m/500m/1000m/2500m/5000m de la localización donde se avistó el pokemon.

- **cooc1** - **cooc151**: booleano que indica si el avistamiento de un pokemon coincidió con el de otro (de una clase entre 1 y 151) en un radio de 100m y en un rango de tiempo de 24 horas.
- **class** dice qué pokemon se trata, y en la página del dataset indica que es el atributo a predecir. ## Preprocesamiento

En primer lugar vamos a ver cuántas muestras y atributos tiene nuestro dataset. Además veremos si las clases están balanceadas, para echo emplearemos el comando `xtab`:

```
ds <- read.csv("300k.csv")
```

```
cat("Hay un total de " , nrow(ds) , " muestras.\n")
```

```
## Hay un total de  296021  muestras.
```

```
cat("Cada muestra tiene " , ncol(ds) , " atributos.\n")
```

```
## Cada muestra tiene  208  atributos.
```

Vamos a proceder a eliminar el atributo **appearedLocalTime**, la eliminamos en primer lugar por la dificultad de trabajar con este dado, el cual podríamos transformar en una serie de variables que desglosasen su contenido, no obstante tenemos otros atributos que ya lo hacen, como son la hora, el día, el mes y el año del avistamiento. Por otro lado vamos a eleminar también el atributo **X__id** que como ya hemos comentado no sabemos qué representa. Además teniendo en cuenta que la aplicación se lanzó el día 6 de julio de 2016, es claro que no el año del avistamiento no aporta ninguna información, con lo que también eleminaremos el atributo **appearedYear**.

```
ds <- subset(ds, select = -c(appearedLocalTime, X_id, appearedYear))
```

Viendo el dataset nos hemos dado cuenta de que para los atributos booleanos que nos indican si hay un gimnasio o una pokeparada a una distancia determinada del lugar de avistamiento del pokemon, siguen un patrón, y es que, al parecer, estas variables lo que indican es si hay un gimnasio o una pokeparada **en un radio** de una determinada longitud, con lo cual en cuanto el atributo que indica si hay un gimansio a una distancia es cierto, el resto de atributos que indican si hay un gimnasio a una distancia mayor también lo son. Lo mismo sucede con las paradas. Entonces una vez hayamos confirmado esto, como la existencia de un gimnasio o una pokeparada en un radio determinado implica la existencia en un radio mayor, podremos eliminar, sin pérdida de información, todos estos atributos y quedarnos únicamente con los atributos **gymDistanceKm** y **pokestopDistanceKm**, que resumirían la información contenida en los otros atributos. Para tratar de comprobar esto vamos a hacer uso de las reglas de asociación con el paquete *arules*:

```
booleansGyms = subset(ds, select = c(gymIn100m, gymIn250m, gymIn500m, gymIn1000m, gymIn2500m, gymIn5000
reglas = apriori(booleansGyms, parameter = list(support = 0.0, confidence = 0.8, minlen = 2, maxlen = 2
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5       0      2
##  maxlen target   ext
##       2  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 0
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[12 item(s), 296021 transaction(s)] done [0.04s].
```

```
## sorting and recoding items ... [12 item(s)] done [0.01s].
## creating transaction tree ... done [0.07s].
## checking subsets of size 1 2
```

```
## Warning in apriori(booleansGyms, parameter = list(support = 0, confidence
## = 0.8, : Mining stopped (maxlen reached). Only patterns up to a length of 2
## returned!
```

```
##  done [0.00s].
## writing ... [44 rule(s)] done [0.00s].
## creating S4 object  ... done [0.03s].
```

```
inspect(subset(reglas, confidence == 1.0))
```

```
##      lhs                    rhs                 support    confidence
## [1]  {gymIn5000m=false} => {gymIn2500m=false} 0.03438270 1
## [2]  {gymIn5000m=false} => {gymIn1000m=false} 0.03438270 1
## [3]  {gymIn5000m=false} => {gymIn500m=false}  0.03438270 1
## [4]  {gymIn5000m=false} => {gymIn250m=false}  0.03438270 1
## [5]  {gymIn5000m=false} => {gymIn100m=false}  0.03438270 1
## [6]  {gymIn2500m=false} => {gymIn1000m=false} 0.09148337 1
## [7]  {gymIn2500m=false} => {gymIn500m=false}  0.09148337 1
## [8]  {gymIn2500m=false} => {gymIn250m=false}  0.09148337 1
## [9]  {gymIn2500m=false} => {gymIn100m=false}  0.09148337 1
## [10] {gymIn100m=true}   => {gymIn250m=true}   0.10508038 1
## [11] {gymIn100m=true}   => {gymIn500m=true}   0.10508038 1
## [12] {gymIn100m=true}   => {gymIn1000m=true}  0.10508038 1
## [13] {gymIn100m=true}   => {gymIn2500m=true}  0.10508038 1
## [14] {gymIn100m=true}   => {gymIn5000m=true}  0.10508038 1
## [15] {gymIn1000m=false} => {gymIn500m=false}  0.16140747 1
## [16] {gymIn1000m=false} => {gymIn250m=false}  0.16140747 1
## [17] {gymIn1000m=false} => {gymIn100m=false}  0.16140747 1
## [18] {gymIn500m=false}  => {gymIn250m=false}  0.32052794 1
## [19] {gymIn500m=false}  => {gymIn100m=false}  0.32052794 1
## [20] {gymIn250m=true}   => {gymIn500m=true}   0.36822388 1
## [21] {gymIn250m=true}   => {gymIn1000m=true}  0.36822388 1
## [22] {gymIn250m=true}   => {gymIn2500m=true}  0.36822388 1
## [23] {gymIn250m=true}   => {gymIn5000m=true}  0.36822388 1
## [24] {gymIn250m=false}  => {gymIn100m=false}  0.63177612 1
## [25] {gymIn500m=true}   => {gymIn1000m=true}  0.67947206 1
## [26] {gymIn500m=true}   => {gymIn2500m=true}  0.67947206 1
## [27] {gymIn500m=true}   => {gymIn5000m=true}  0.67947206 1
## [28] {gymIn1000m=true}  => {gymIn2500m=true}  0.83859253 1
## [29] {gymIn1000m=true}  => {gymIn5000m=true}  0.83859253 1
## [30] {gymIn2500m=true}  => {gymIn5000m=true}  0.90851663 1
##      lift
## [1]  10.930948
## [2]   6.195500
## [3]   3.119853
## [4]   1.582839
## [5]   1.117419
## [6]   6.195500
## [7]   3.119853
## [8]   1.582839
## [9]   1.117419
## [10]  2.715739
```

```
## [11]  1.471731
## [12]  1.192474
## [13]  1.100695
## [14]  1.035607
## [15]  3.119853
## [16]  1.582839
## [17]  1.117419
## [18]  1.582839
## [19]  1.117419
## [20]  1.471731
## [21]  1.192474
## [22]  1.100695
## [23]  1.035607
## [24]  1.117419
## [25]  1.192474
## [26]  1.100695
## [27]  1.035607
## [28]  1.100695
## [29]  1.035607
## [30]  1.035607
```

```
booleansPokestops = subset(ds, select = c(pokestopIn100m, pokestopIn250m, pokestopIn500m, pokestopIn100
reglas = apriori(booleansPokestops, parameter = list(support = 0.0, confidence = 0.8, minlen = 2, maxle
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5       0      2
##  maxlen target   ext
##       2  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 0
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[12 item(s), 296021 transaction(s)] done [0.04s].
## sorting and recoding items ... [12 item(s)] done [0.01s].
## creating transaction tree ... done [0.07s].
## checking subsets of size 1 2
```

```
## Warning in apriori(booleansPokestops, parameter = list(support = 0,
## confidence = 0.8, : Mining stopped (maxlen reached). Only patterns up to a
## length of 2 returned!
```

```
##  done [0.00s].
## writing ... [44 rule(s)] done [0.00s].
## creating S4 object  ... done [0.03s].
```

```
inspect(subset(reglas, confidence == 1.0))
```

```
##      lhs                          rhs                        support
## [1]  {pokestopIn5000m=false} => {pokestopIn2500m=false} 0.01465099
```

```
## [2]  {pokestopIn5000m=false} => {pokestopIn1000m=false} 0.01465099
## [3]  {pokestopIn5000m=false} => {pokestopIn500m=false}  0.01465099
## [4]  {pokestopIn5000m=false} => {pokestopIn250m=false}  0.01465099
## [5]  {pokestopIn5000m=false} => {pokestopIn100m=false}  0.01465099
## [6]  {pokestopIn2500m=false} => {pokestopIn1000m=false} 0.03284902
## [7]  {pokestopIn2500m=false} => {pokestopIn500m=false}  0.03284902
## [8]  {pokestopIn2500m=false} => {pokestopIn250m=false}  0.03284902
## [9]  {pokestopIn2500m=false} => {pokestopIn100m=false}  0.03284902
## [10] {pokestopIn1000m=false} => {pokestopIn500m=false}  0.07603515
## [11] {pokestopIn1000m=false} => {pokestopIn250m=false}  0.07603515
## [12] {pokestopIn1000m=false} => {pokestopIn100m=false}  0.07603515
## [13] {pokestopIn500m=false}  => {pokestopIn250m=false}  0.17450789
## [14] {pokestopIn500m=false}  => {pokestopIn100m=false}  0.17450789
## [15] {pokestopIn100m=true}   => {pokestopIn250m=true}   0.34113458
## [16] {pokestopIn100m=true}   => {pokestopIn500m=true}   0.34113458
## [17] {pokestopIn100m=true}   => {pokestopIn1000m=true}  0.34113458
## [18] {pokestopIn100m=true}   => {pokestopIn2500m=true}  0.34113458
## [19] {pokestopIn100m=true}   => {pokestopIn5000m=true}  0.34113458
## [20] {pokestopIn250m=false}  => {pokestopIn100m=false}  0.35807257
## [21] {pokestopIn250m=true}   => {pokestopIn500m=true}   0.64192743
## [22] {pokestopIn250m=true}   => {pokestopIn1000m=true}  0.64192743
## [23] {pokestopIn250m=true}   => {pokestopIn2500m=true}  0.64192743
## [24] {pokestopIn250m=true}   => {pokestopIn5000m=true}  0.64192743
## [25] {pokestopIn500m=true}   => {pokestopIn1000m=true}  0.82549211
## [26] {pokestopIn500m=true}   => {pokestopIn2500m=true}  0.82549211
## [27] {pokestopIn500m=true}   => {pokestopIn5000m=true}  0.82549211
## [28] {pokestopIn1000m=true}  => {pokestopIn2500m=true}  0.92396485
## [29] {pokestopIn1000m=true}  => {pokestopIn5000m=true}  0.92396485
## [30] {pokestopIn2500m=true}  => {pokestopIn5000m=true}  0.96715098
##      confidence lift
## [1]  1          30.442308
## [2]  1          13.151813
## [3]  1           5.730400
## [4]  1           2.792730
## [5]  1           1.517761
## [6]  1          13.151813
## [7]  1           5.730400
## [8]  1           2.792730
## [9]  1           1.517761
## [10] 1           5.730400
## [11] 1           2.792730
## [12] 1           1.517761
## [13] 1           2.792730
## [14] 1           1.517761
## [15] 1           1.557808
## [16] 1           1.211399
## [17] 1           1.082292
## [18] 1           1.033965
## [19] 1           1.014869
## [20] 1           1.517761
## [21] 1           1.211399
## [22] 1           1.082292
## [23] 1           1.033965
## [24] 1           1.014869
```

```
## [25] 1           1.082292
## [26] 1           1.033965
## [27] 1           1.014869
## [28] 1           1.033965
## [29] 1           1.014869
## [30] 1           1.014869
```

Hemos obtenido las distintas reglas de asociación sin atender al soporte ya que no estamos interesados en saber cuantas veces se da una correspondencia entre dos hechos (el soporte de dicha regla), lo que queremos saber es que cuando se da un hecho, esto implica que se den los que suponemos que deberian darse (la confianza de las reglas encontradas). Por tanto, como podemos ver, las reglas de asociación encontradas prueban nuestras suposiciones. Dado esto, procedemos a elminar los atributos. Dado que la confianza es del 100%, no afecta haber usado el conjunto entero.

```
ds <- subset(ds, select = -c(gymIn100m, gymIn250m, gymIn500m, gymIn1000m, gymIn2500m, gymIn5000m, pokes
```

Este dataset nos proporciona una serie de atributos para la localización del pokemon. Los primeros que nos encontramos son **latitude** y **longitude**, se tratan de coordenadas geográficas. Por otro lado aparecen **cellId_90m**, **cellId_180m**, **cellId_370m**, **cellId_730m**, **cellId_1460m**, **cellId_2920m**, **cellId_5850m**. Estos indican la posición geográfica usando celdas s2. Estas celdas se clasifican en niveles atendiendo a su área, desde 0 (menor área) a 30 (mayor área). Se obtienen según longitud y latitud, por lo que son la misma información representada de distinta manera. Además para métodos que dependen de una distancia como el KNN tendríamos que investigar la distancia entre las distintas celdas a través de su ID, lo que supondría una carga de trabajo extra e innecesaria. Se puede consultar más información sobre las celdas s2 en el siguiente enlace

Por lo comentando arriba se va a optar a eliminar los atributos correspondientes a las celdas, dejando los atributos **latitude** y **longitude** como los únicos para determinar la posición.

```
ds <- subset(ds, select = -c(cellId_90m, cellId_180m, cellId_370m, cellId_730m, cellId_1460m, cellId_29
```

Por otro lado al explorar el dataset observamos que en el atributo **appearedDayOfWeek** se tomaba el valor *dummy_day*, y al consultar los valores que toma este atributo a lo largo del dataset vimos que no aparecia el lunes cuando el resto de los dias de la semana si que aparecen, con lo cual procedemos a revisar si este valor del atributo se corresponde con el lunes o es simplemente un valor perdido:

```
unique(ds$appearedDayOfWeek)
```

```
## [1] dummy_day Sunday    Saturday  Friday    Thursday  Wednesday Tuesday
## Levels: dummy_day Friday Saturday Sunday Thursday Tuesday Wednesday
```

```
unique(subset(ds, appearedDayOfWeek == "dummy_day")$appearedMonth)
```

```
## [1] 8
```

```
unique(subset(ds, appearedDayOfWeek == "dummy_day")$appearedDay)
```

```
## [1] 8
```

Como podemos observar el único día en el que se registran observaciones en las que el atributo toma el valor *dummy_day* es el 8 de agosto que efectivamente fue un lunes. Además mientras se comprobaba este hecho hemos observado que todas las observaciones se realizaron en agosto y que además fue durante una semana de agosto, con lo cual el atributo **appearedMonth** no aporta ninguna información y además los atributos **appearedDayOfWeek** y **appearedDay** aportan la misma información, ya que al tomarse las muestras durante una sola semana hay una correspondencia biyectiva entre los valores de ambos atributos. Y como a priori no consideramos que la distancia entre días sea significativa, vamos a optar por quedarnos con el atributo categórico:

```r
#le damos un categoría más singnificativa que dummy_day al atributo
levels(ds$appearedDayOfWeek)[1] <- "Monday"

#vemos los meses y días en los que se tomaron las muestras
unique(ds$appearedMonth)
```

```
## [1] 8
```

```r
unique(ds$appearedDay)
```

```
## [1] 8 7 6 5 4 3 2
```

```r
#eliminamos las variables innecesarias
ds <- subset(ds, select = -c(appearedMonth, appearedDay))
```

```r
nrow(ds[ds$appearedDayOfWeek == "Monday", ])
```

```
## [1] 9001
```

```r
levels(ds$appearedDayOfWeek)
```

```
## [1] "Monday"    "Friday"    "Saturday"  "Sunday"    "Thursday"  "Tuesday"
## [7] "Wednesday"
```

Con lo cual como vemos el número de muestras y de atributos es muy elevado. Además el número de clases a clasificar es muy elevado por lo tanto lo que vamos a hacer es intentar clasificar los pokemon según su tipo, así que vamos a añadir dicha columna al dataset. Los tipos considerados son aquellos que se establecieron en la primera generación de Pokemon y se recogen por medio de las siguientes variables que almacenan una cadena indicando el tipo; estas variables se han añadido por comodidad a la hora de generar el vector de tipos que crearemos a continuación:

```r
P = "planta"
A = "agua"
F = "fuego"
B = "bicho"
G = "fantasma"
L = "lucha"
N = "normal"
E = "electrico"
S = "psiquico"
V = "veneno"
H = "hielo"
D = "dragon"
T = "tierra"
R = "roca"
```

A continuación formamos un array considerando el tipo primario de cada pokemon de la primera generación, dando lugar a un array de 151 elementos. Pese que en la app hay algunas especies de pokemon que no aparecen por motivos de indexación, ya que las muestras dan la clase según el número original del pokemon, introducimos el tipo de los 151 pokemon:

```r
tipos = c(P,P,P,F,F,F,A,A,A,B,B,B,
  B,B,B,N,N,N,N,N,N,N,V,V,
  E,E,T,T,V,V,V,V,V,V,N,N,
  F,F,N,N,V,V,P,P,P,B,B,B,
  B,T,T,N,N,A,A,L,L,F,F,A,
  A,A,S,S,S,L,L,L,P,P,P,A,
  A,R,R,R,F,F,A,A,E,E,N,N,
```

```
  N,A,A,V,V,A,A,G,F,F,R,S,
  S,A,A,E,E,P,P,T,T,L,L,N,
  V,V,T,T,N,P,N,A,A,A,A,A,
  A,S,B,H,E,F,B,N,A,A,A,N,
  N,A,E,F,N,R,R,R,R,R,N,H,
  E,F,D,D,D,S,S)
```

Ahora vamos a proceder a añadir la columna de tipos al dataset que hemos cargado:

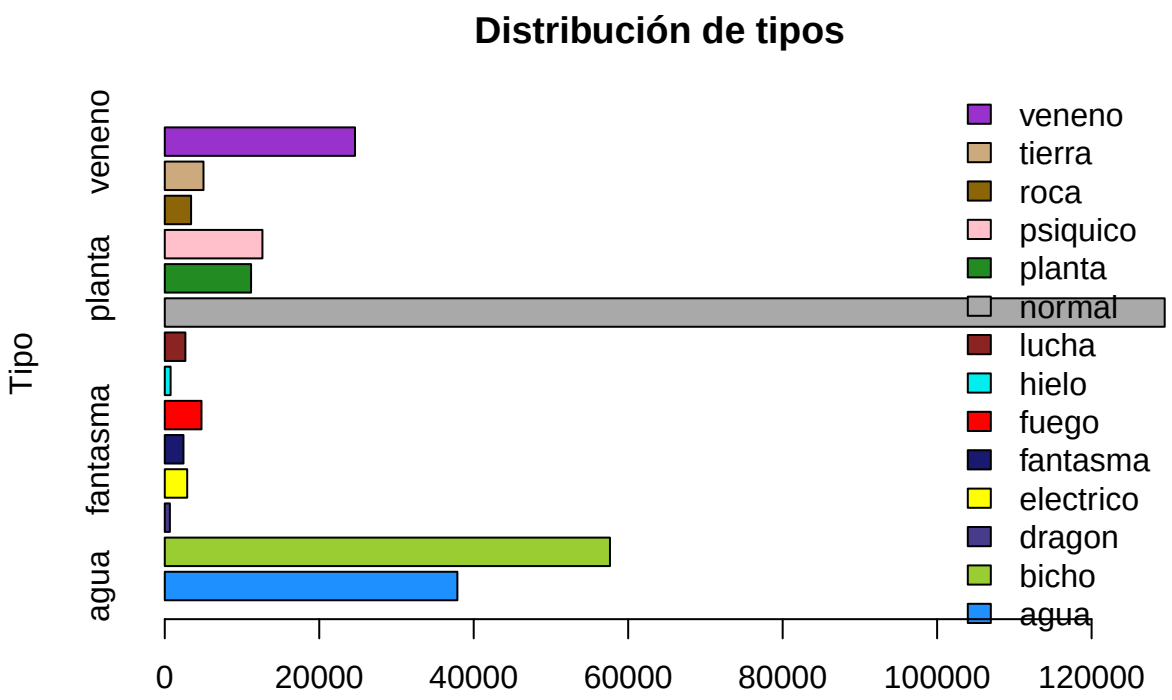
```
ds["tipo"] <- tipos[ds$class]
```

Veamos entonces cómo se distribuyen las muestran que tenemos según el tipo de pokemon avistado:

```
T <- xtabs(~ tipo, ds)
etiquetas = c("agua", "bicho", "dragon", "electrico", "fantasma", "fuego", "hielo", "lucha", "normal",
print(T)
```

```
## tipo
##      agua     bicho    dragon electrico  fantasma     fuego     hielo
##     37887     57641       664      2903      2419      4755       755
##     lucha    normal    planta  psiquico      roca    tierra    veneno
##      2670    129463     11171     12642      3408      5012     24631
```

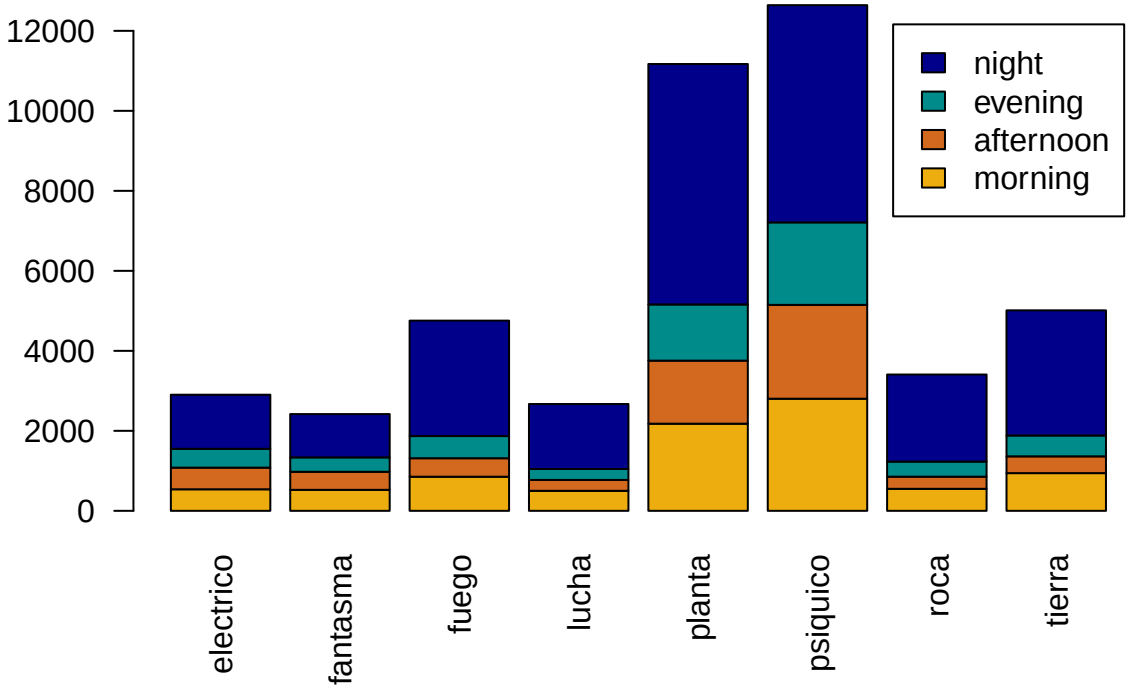
```
colores = rev(c("darkorchid", "burlywood3","darkgoldenrod4", "pink", "forestgreen", "darkgray", "brown4
barplot(T, main = "Distribución de tipos", horiz = TRUE, ylab= "Tipo", col = colores, legend.text = eti
```



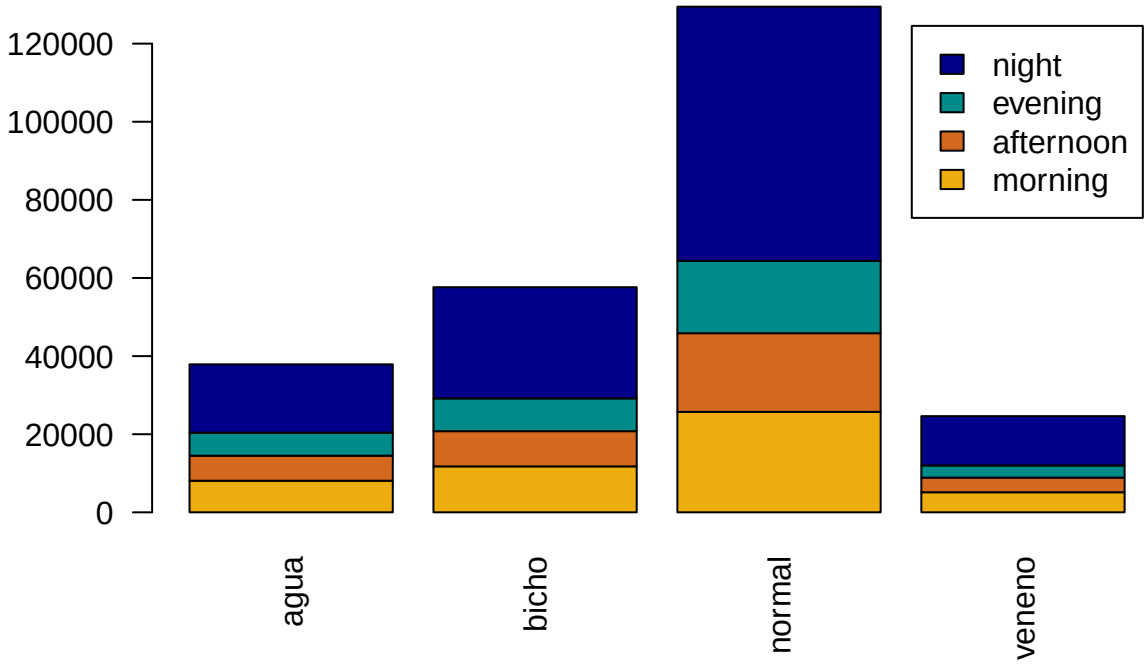
Como podemos ver una vez que hemos agrupado las muestras por el tipo de Pokemon vemos que las clases están tremendamente desequilibradas. Mientras que hay muchísimos avistamientos de Pokemon de tipo normal, las clases hielo, dragón, fastasma, eléctrico o roca resultan marginales. Con lo cual en primer lugar hemos de realizar un equilibrado de estas clases.

```
ds$appearedTimeOfDay <- factor(ds$appearedTimeOfDay, levels = c("morning", "afternoon", "evening", "nig
tabla <- table(ds$appearedTimeOfDay, ds$tipo)
tabla1 <- table(ds[! ds$tipo %in% c("normal", "bicho", "agua", "dragon", "hielo", "veneno"), c("appeare
tabla2 <- table(ds[ds$tipo %in% c("normal", "bicho", "agua", "veneno"), c("appearedTimeOfDay", "tipo")]
```

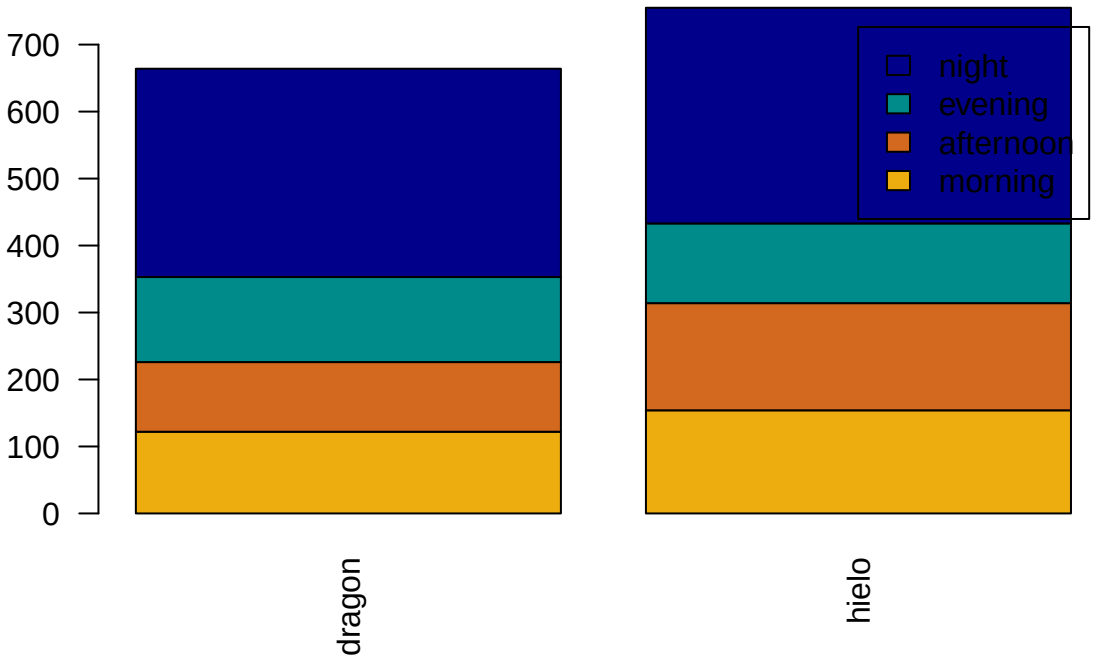
```
tabla3 <- table(ds[ds$tipo %in% c("dragon","hielo"), c("appearedTimeOfDay", "tipo")])
barplot(tabla1, legend = levels(ds$appearedTimeOfDay), las = 2, col = c("darkgoldenrod2", "chocolate",
```




```
barplot(tabla2, legend = levels(ds$appearedTimeOfDay), las = 2, col = c("darkgoldenrod2", "chocolate",
```




```
barplot(tabla3, legend = levels(ds$appearedTimeOfDay), las = 2, col = c("darkgoldenrod2", "chocolate",
```



Aquí podemos apreciar que para cualquier tipo de pokemon la mayoría de los avistamientos se producen durante la noche. Creíamos que ibamos a apreciar una tendencia en la que los pokemon de tipo fantasma fuesen los que presentaban este comportamiento de una forma más acusada, sin embargo todos los tipos siguen este patrón de aparición.