UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
CAMPUS SANTIAGO
DEPARTAMENTO DE INFORMÁTICA
PRIMER SEMESTRE 2020

Ayudantía 5: Árboles de Clasificación (en R)

Profesor Nicolás Rojas
Francisca Ramírez
Felipe Vega

[34]:
```
#configuraciones
library(repr)
options(repr.plot.width=30,repr.plot.height=10)
IRdisplay::display_html("<style>.container { width:95% !important; }</style>")
options(jupyter.plot_mimetypes = c("text/plain", "image/png" ))
```

1 Ayudantía Árboles de Clasificación (en R)

El objetivo de este notebook es mostrar una manera de utilizar árboles de clasificación en R. Para esto, se realizará un pequeño ejercicio de clasificación utilizando un dataset de Pokémon con los datos de Smogon University, la cual es una comunidad especializada en las batallas competitivas.

El dataset original fue subido a la plataforma Kaggle por el usuario Gibs. Este puede ser descargado en el siguiente link. Para efectos de la ayudantía se subirá a aula una versión lista para ser utilizada.

1.1 Antes de comenzar

Es necesario instalar R antes de seguir con la ayudantía, puedes descargarlo en el siguiente link.

Se recomienda R Studio como entorno de desarrollo, pero en caso de querer utilizar Jupyter pueden instalar el kernel de R siguiendo el siguiente tutorial.

1.2 Librería

Para construir los árboles de clasificación es necesario instalar el paquete `tree`. Para esto deben utilizar el siguiente comando.

[2]:
```
install.packages('tree')
```

```
Installing package into 'C:/Users/felip/Documents/R/win-library/4.0'
(as 'lib' is unspecified)


package 'tree' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\felip\AppData\Local\Temp\RtmpqwIa6H\downloaded_packages
```

Si utilizan R Studio, puedes instalarlo seleccionando ***Tools*** ← ***Install Packages*** en la barra de herramientas.

Nota: Existen otros paquetes para la construcción de árboles, pero se recomienda utilizar el paquete antes mencionado.

Una vez instalado, se debe importar el paquete con el comando `library`.

[3]:
```
library(tree)
```

1.3 Dataset

El dataset a utilizar está compuesto por 15 columnas, las cuales se describen a continuación.

- ***DexNum:*** (Int) Corresponde al número en la Pokedex del Pokémon.
- ***Name:*** (String) Nombre del Pokémon
- ***Type1:*** (String) Tipo primario del Pokémon, puede tomar uno de los siguientes valores: Water; Normal; Psychic; Bug; Grass; Fire; Electric; Dragon; Rock; Dark; Ground; Steel; Ghost; Fighting; Ice; Poison; Fairy; Flying.
- ***Type2:*** (String) Tipo secundario del Pokémon, puede tomar los mismos valores anteriores o el valor "NoType".
- ***Total:*** (Int) Suma de los stats base del Pokémon.
- ***HP:*** (Int) Health Point, stat de vida del Pokémon.
- ***Atk:*** (Int) Stat de ataque del Pokémon.
- ***Def:*** (Int) Stat de defensa del Pokémon.
- ***SAtk:*** (Int) Stat de ataque especial del Pokémon.
- ***SDef:*** (Int) Stat de defensa especial del Pokémon.
- ***Spd:*** (Int) Stat de velocidad del Pokémon.
- ***Generation:*** (Int) Generación en la cual fue creado el Pokémon. Toma valores del 1 al 6.
- ***Legendary:*** (Boolean) indica si el Pokémon es legendario o no.
- ***Mega:*** (Boolean) indica si el Pokémon es una mega evolución o no.
- ***Tier:*** (String) clasificación entregada por Smogon a cada Pokémon según su aptitud en el juego competitivo. Puede tomar los valores PU, NU (NeverUsed), RU (RarelyUsed), UU (UnderUsed), OU (OverUsed) y Uber.

El objetivo de este ejercicio es predecir la clasificación (tier) de cada Pokémon basado en los otros atributos. Esto podría ser útil para Smogon cuando sale un nuevo juego y hay que armar nuevamente las clasificaciones, tener un modelo que prediga una posible clase podría crear una clasificación inicial sobre la cual se puede iterar para llegar a nuevas clasificaciones.

2 Manos a la obra

2.1 Lectura de datos

Primero se deben cargar los datos en memoria, para esto se debe usar el siguiente comando.

[4]:
```
data <- read.table("smogon_cleaned.csv",header=TRUE,sep=';')
```

La línea anterior guarda una tabla con el dataset en la variable `data`. El primer parámetro corresponde a la ruta del archivo a leer, mientras los otros dos indican si el archivo cuenta con en-

cabezados y el símbolo de separación de los datos.

Para ver los datos de una determinada columna se puede utilizar lo siguiente `variable_con_datos$nombre_columna`. Por ejemplo, podemos ver los datos de la variable `Name`. Por temas de espacio se mostrarán solo los primeros 10 Pokémon del dataset.

[5]:
```
data$Name[0:10]
```

1. 'Mega Rayquaza' 2. 'Mega Gengar' 3. 'Mega Kangaskhan' 4. 'Mewtwo' 5. 'Mega Mewtwo X' 6. 'Mega Mewtwo Y' 7. 'Lugia' 8. 'Ho-oh' 9. 'Blaziken' 10. 'Mega Blaziken'

2.2 Breve exploración de datos

Una de las fases más importantes al momento de realizar un trabajo de predicción a partir de datos es conocer los datos con los que se van a trabajar. Para lo anterior se realiza un proceso de Análisis Exploratorio de Datos (EDA: exploratory data analysis).

A continuación se muestra una pequeña exploración del dataset a utilizar.

[6]:
```
dim(data)
```

1. 499 2. 15

Nuestro dataset cuenta con 499 registros, cada uno con 15 atributos. Si alguien conoce de Pokémon, puede saber que existen muchos más, esto se debe a que Smogon no considera a todos los Pokémon existente para sus clasificaciones en partidas competitivas.

2.2.1 Cantidad de Pokémon por cada tier

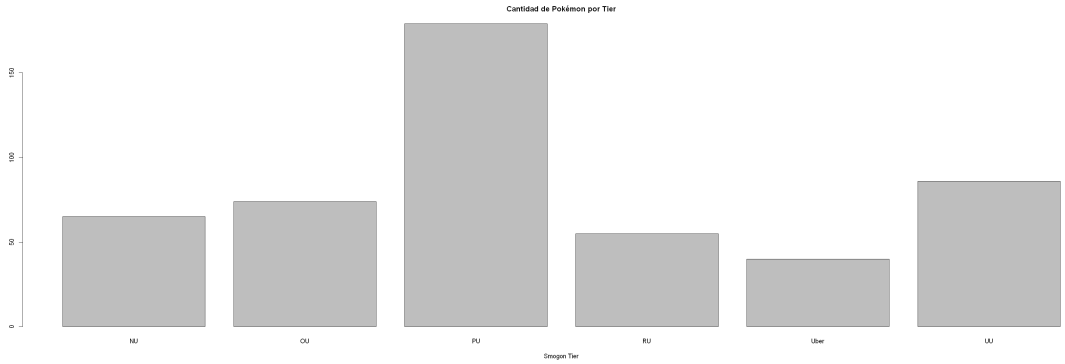
Una de las cosas importantes a conocer sobre el dataset es cuantos datos tenemos por cada clase a clasificar. Para esto, podemos utilizar el comando `xtabs`, con el cual podemos ver la cantidad de registros por cada valor posible de una columna. Esta función recibe como parámetros la columna a analizar y la tabla con los datos.

[7]:
```
xtabs(~Tier,data=data)
```

```
Tier
  NU   OU   PU   RU Uber   UU
  65   74  179   55   40   86
```

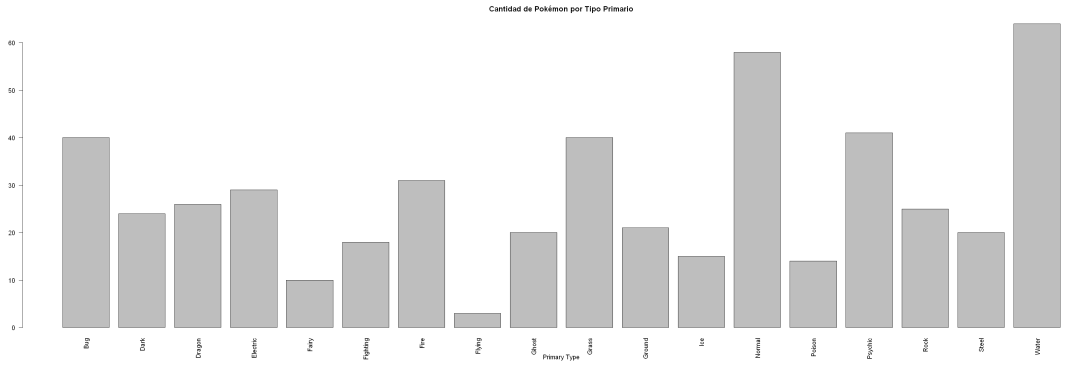
También podemos ver esta misma información mediante un gráfico de barras.

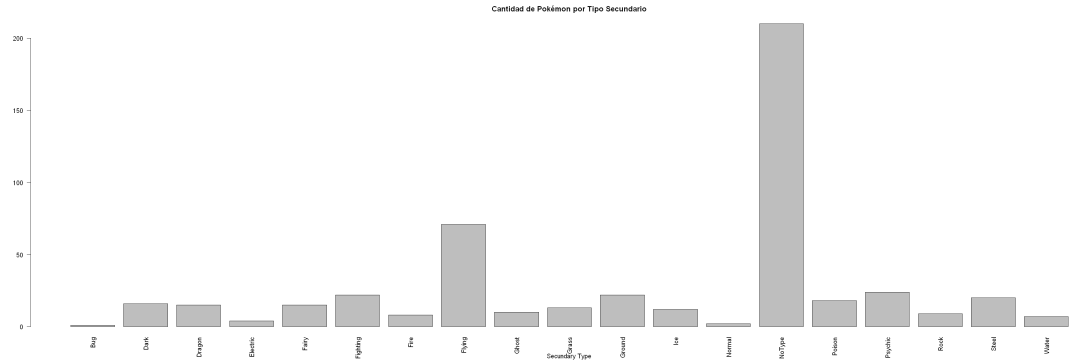
[8]:
```
tier_count <- table(data$Tier)
barplot(tier_count, main='Cantidad de Pokémon por Tier',xlab='Smogon Tier')
```



No solo es importante conocer como se distribuye el atributo a predecir, también es importante ver esta información para cada atributo del dataset. En este caso se omitirán las columnas `DexNum` y `Name` ya que no entregan mucha información. Por temas de espacio solo se realizarán dos gráficos.

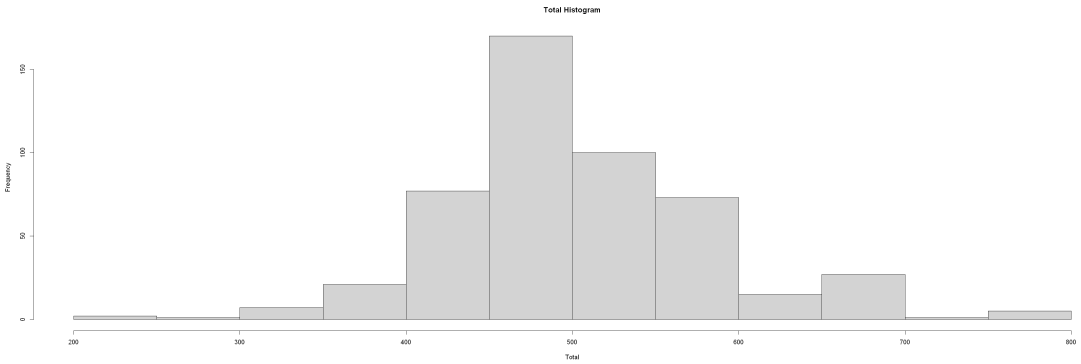
[9]:
```
type1_count <- table(data$Type1)
barplot(type1_count, main='Cantidad de Pokémon por Tipo Primario',xlab='Primary␣
 ↪Type',las=2)

type2_count <- table(data$Type2)
barplot(type2_count, main='Cantidad de Pokémon por Tipo␣
 ↪Secundario',xlab='Secundary Type',las=2)
```

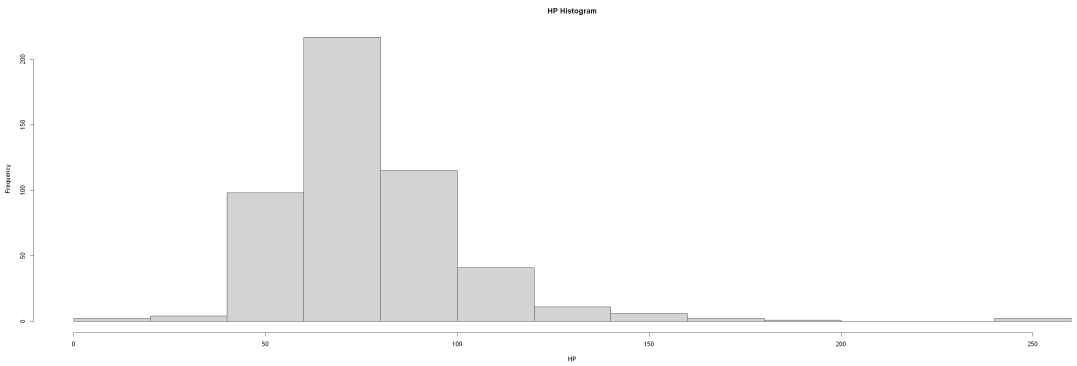




Como las columnas de stats son numéricas se utilizarán histogramas para observar como se distribuyen. Solo se graficarán dos stats, nuevamente por temas de espacio.

[10]:
```
hist(data$Total, main='Total Histogram',xlab='Total')
hist(data$HP, main='HP Histogram',xlab='HP')
```





2.2.2 Distribución por cada clase

Una cosa interesante de ver es si existe alguna diferencia entre los stats de los Pokémon de cada Tier. Esto lo podemos comparar utilizando Boxplot por cada uno de los stats.

Para esto, instalemos la librería `ggplot2`, la cual hace mucho más sensilla la tarea de crear visualizaciones.

[11]:
```
install.packages('ggplot2')
```

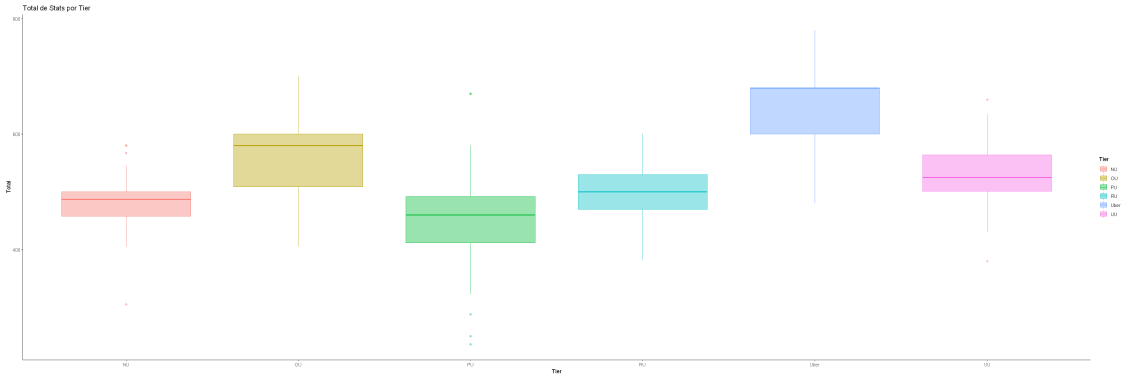
```
Installing package into 'C:/Users/felip/Documents/R/win-library/4.0'
(as 'lib' is unspecified)


package 'ggplot2' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\felip\AppData\Local\Temp\RtmpqwIa6H\downloaded_packages
```

[12]:
```
library(ggplot2)
```

En este caso se creará un boxplot para comparar el atributo `Total`. Para esto debemos utilizar la función `ggplot` y la función `geom_boxplot`.

Queremos que el eje X de nuestro boxplot sea cada una de nuestras Tiers, lo cual indicamos con `x=Tier` en el segundo parámetro de `ggplot`. El eje Y se lo indicamos con `y=Total`, mientras que los parámetros `color` y `fill` sirven para cambiar el estilo del gráfico. Finalmente con la función `geom_boxplot` indicamos que queremos realizar dicho gráfico, cuyo parámetro alpha controla la transparencia de los colores utilizados.

[13]:
```
ggplot(data, aes(x=Tier,color=Tier,y=Total,fill=Tier)) +
    geom_boxplot(alpha=0.4) +
    labs(title="Total de Stats por Tier") + theme_classic()
```



¿Será una buena idea realizar una partición por este atributo?

Se podrían realizar muchas otras visualizaciones para comprender nuestro dataset. Algunas po-

drían indicarnos que es necesario realizar algunas transformaciones en algunos atributos, como cambios de escalas por poner un ejemplo, pero dicha labor se escapa del enfoque de este curso.

3 Manos a la obra

3.1 Transformación de la variable a predecir

La librería `tree` no acepta que el atributo a predecir sea String, es por esto que se cambia de tipo a `factor`, lo que hace que esta columna cambie a una columna categorica. También cambiaremos los otros atributos de tipo String.

[14]:
```
data$Tier <- factor(data$Tier)
data$Legendary <- factor(data$Legendary)
data$Mega <- factor(data$Mega)
data$Type1 <- factor(data$Type1)
data$Type2 <- factor(data$Type2)
```

3.2 Train - Test Split

Algo importante cuando se trabaja con técnicas de aprendizaje es asegurarnos que nuestro modelo logra generalizar de buena manera y no solo funcione con los datos con que lo entrenamos. Una manera de verificar esto es separar nuestro conjunto de datos en dos sub-conjuntos de datos: uno de entrenamiento, para construir el árbol de clasificación; y uno de pruebas, para probar el desempeño de este.

Esto lo podemos realizar de la siguiente manera.

[15]:
```
#Seteamos una semilla para replicar resultados
set.seed(42)
#En este caso utilizaremos un conjunto de entrenamiento con el 80% de los datos
train_size <- floor(0.80 * nrow(data)) #El tamaño será el 80% de la cantidad de␣
 ↪filas
train_mask <- sample(seq_len(nrow(data)),size= train_size) #Mascara aleatoria
#Separamos los conjuntos
train <- data[train_mask, ]
test <- data[-train_mask, ]
```

Esto guarda en la variable `train` el 80 de los datos, y en `test` el 20 restante.

[16]:
```
dim(train)
```

1. 399 2. 15

[17]:
```
dim(test)
```

1. 100 2. 15

3.3 Ármando el árbol

Para construir nuestro árbol utilizaremos el método `tree` de la librería del mismo nombre.

[18]:
```
arbol = tree(Tier ~ Total + HP + Atk + Def + SAtk + SDef + Spd +
             Generation + Legendary + Mega, data = train)
```

Utilizaremos dos parámetros para el método `tree`. El primero se conoce como fórmula, donde la primera columna que aparece es la columna que queremos predecir, luego sigue el símbolo ~ para luego colocar todas las columnas que queremos utilizar para construir nuestro árbol. En caso de querer excluir una columna, en este caso `DexNum` y `Name`, es cosa de no agregarla en la fórmula. El segundo parámetro es el conjunto de datos que se utilizará para entrenar el árbol.

Nota: Los atributos `Type` fueron excluidos para que en el futuro árbol aparezca el campo Generation. Esto fue realizado para ejemplificar una situación en la siguente sección.

Una vez construído, podemos ver un resumen del árbol con el comando `summary`.

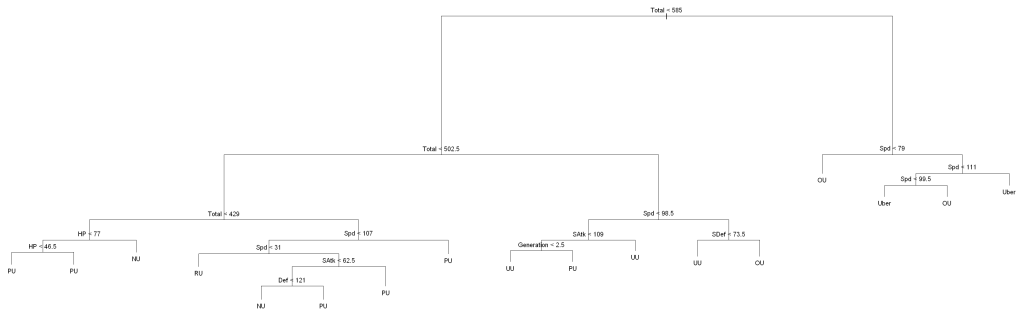
[19]:
```
summary(arbol)
```

```
Classification tree:
tree(formula = Tier ~ Total + HP + Atk + Def + SAtk + SDef +
    Spd + Generation + Legendary + Mega, data = train)
Variables actually used in tree construction:
[1] "Total"      "HP"         "Spd"        "SAtk"       "Def"
[6] "Generation" "SDef"
Number of terminal nodes:  17
Residual mean deviance:  2.138 = 816.8 / 382
Misclassification error rate: 0.4236 = 169 / 399
```

Esto nos muestra la fórmula con la que fue construido el árbol, los atributos utilizados para la construcción del árbol, el error de clasificación, entre otras.

Una de las ventajas que tiene un árbol de clasificación por sobre otras técnicas de aprendizaje automático es que podemos visualizar este para ver como es que se llega a una determinada predicción.

[20]:
```
plot(arbol)
text(arbol,pretty=1)
```



Si la condición del nodo se cumple, se sigue con la rama izquierda, caso contrario se sigue la rama derecha.

3.4 Discretización

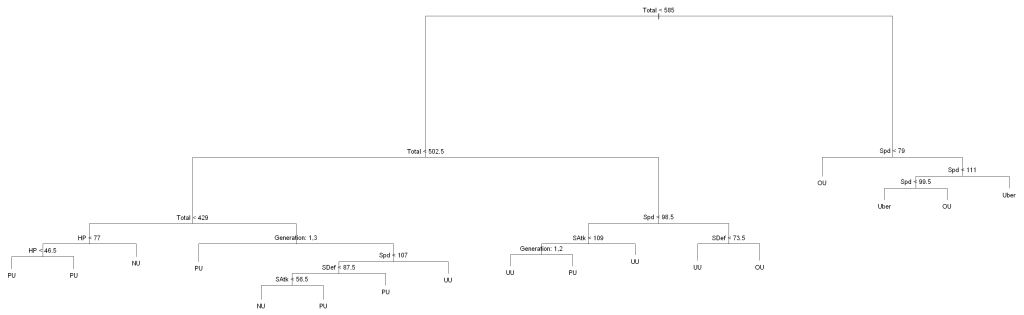
Viendo la imagen anterior podemos notar una partición un poco rara, dado que no es posible que un Pokémon pertenezca a la generación 2.5 (ignoremos Pokémon presentados anticipadamente en el anime). Para arreglar esta situación debemos cambiar a categorica la variable `Generation`, tal como lo hicimos con las variables de tipo String.

[21]:
```
data$Generation <- factor(data$Generation)
```

Repetimos el proceso anterior para obtener un nuevo árbol.

[22]:
```
#Seteamos una semilla para replicar resultados
set.seed(42)
#En este caso utilizaremos un conjunto de entrenamiento con el 80% de los datos
train_size <- floor(0.80 * nrow(data)) #El tamaño será el 80% de la cantidad de␣
 ↪filas
train_mask <- sample(seq_len(nrow(data)),size= train_size) #Mascara aleatoria
#Separamos los conjuntos
train <- data[train_mask, ]
test <- data[-train_mask, ]

arbol_discretizado = tree(Tier ~ Total +
             HP + Atk + Def + SAtk + SDef + Spd +
             Generation + Legendary + Mega, data = train)
```

[23]:
```
plot(arbol_discretizado)
text(arbol_discretizado,pretty=1)
```



Ahora, dicha partición se realiza si el Pokémon pertenece a la generación 1 o 2.

Nota: Los Stats en el juego no pueden tomar valores intermedios por lo que también deberían ser discretizados. Sin embargo, el módulo `tree` solo permite variables categóricas con menos de 32 valores posibles dado que aumenta mucho el procesamiento computacional necesario para probar los posibles splits. Lo que se debería hacer en este caso es agrupar dichos valores en niveles, por ejemplo, separar un stat en alto, medio y bajo. Esto no se realiza ya que se escapa un poco de lo que les pedirán en el control.

3.5 Viendo el árbol más a fondo

Con las siguientes instrucciones se pueden ver más detalles del árbol, incluyendo cuantos datos existen en cada partición, la probabilidad de cada clase en cada partición, y la predicción que se haría en cada nodo de ser un nodo terminal. Con estos valores podrían calcular la impureza que tiene cada nodo.

[24]:
```
arbol_discretizado
```

```
node), split, n, deviance, yval, (yprob)
      * denotes terminal node
  1) root 399 1327.00 PU ( 0.137845 0.155388 0.358396 0.110276 0.075188 0.162907 )
    2) Total < 585 332 1001.00 PU ( 0.165663 0.096386 0.430723 0.123494 0.015060 0.168675 )
      4) Total < 502.5 227  583.50 PU ( 0.189427 0.061674 0.559471 0.105727 0.008811 0.074890 )
        8) Total < 429 62   89.70 PU ( 0.096774 0.048387 0.806452 0.032258 0.000000 0.016129 )
         16) HP < 77 52   53.15 PU ( 0.038462 0.019231 0.884615 0.038462 0.000000 0.019231 )
           32) HP < 46.5 6   12.14 PU ( 0.333333 0.166667 0.500000 0.000000 0.000000 0.000000 ) *
           33) HP > 46.5 46   26.00 PU ( 0.000000 0.000000 0.934783 0.043478 0.000000 0.021739 ) *
         17) HP > 77 10   21.10 NU ( 0.400000 0.200000 0.400000 0.000000 0.000000 0.000000 ) *
        9) Total > 429 165  468.60 PU ( 0.224242 0.066667 0.466667 0.133333 0.012121 0.096970 )
         18) Generation: 1,3 66  167.50 PU ( 0.333333 0.060606 0.484848 0.030303 0.030303 0.060606 ) *
         19) Generation: 2,4,5,6 99  279.30 PU ( 0.151515 0.070707 0.454545 0.202020 0.000000 0.121212 )
           38) Spd < 107 84  224.00 PU ( 0.178571 0.047619 0.488095 0.202381 0.000000 0.083333 )
             76) SDef < 87.5 57  153.50 PU ( 0.228070 0.070175 0.491228 0.105263 0.000000 0.105263 )
              152) SAtk < 56.5 10   27.32 NU ( 0.300000 0.200000 0.000000 0.300000 0.000000 0.200000 ) *
              153) SAtk > 56.5 47  108.80 PU ( 0.212766 0.042553 0.595745 0.063830 0.000000 0.085106 ) *
             77) SDef > 87.5 27   55.76 PU ( 0.074074 0.000000 0.481481 0.407407 0.000000 0.037037 ) *
           39) Spd > 107 15   40.87 UU ( 0.000000 0.200000 0.266667 0.200000 0.000000 0.333333 ) *
      5) Total > 502.5 105  336.20 UU ( 0.114286 0.171429 0.152381 0.161905 0.028571 0.371429 )
       10) Spd < 98.5 72  221.90 UU ( 0.138889 0.083333 0.222222 0.125000 0.027778 0.402778 )
```

```
      20) SAtk < 109 53  154.90 UU ( 0.150943 0.056604 0.245283 0.169811 0.000000 0.377358 )
        40) Generation: 1,2 16   26.60 UU ( 0.000000 0.000000 0.187500 0.125000 0.000000 0.687500 ) *
        41) Generation: 3,4,5,6 37  114.50 PU ( 0.216216 0.081081 0.270270 0.189189 0.000000 0.243243 ) *
      21) SAtk > 109 19   53.61 UU ( 0.105263 0.157895 0.157895 0.000000 0.105263 0.473684 ) *
    11) Spd > 98.5 33   89.04 OU ( 0.060606 0.363636 0.000000 0.242424 0.030303 0.303030 )
      22) SDef < 73.5 8   11.77 UU ( 0.125000 0.000000 0.000000 0.000000 0.125000 0.750000 ) *
      23) SDef > 73.5 25   56.94 OU ( 0.040000 0.480000 0.000000 0.320000 0.000000 0.160000 ) *
  3) Total > 585 67  152.30 OU ( 0.000000 0.447761 0.000000 0.044776 0.373134 0.134328 )
    6) Spd < 79 14   28.12 OU ( 0.000000 0.428571 0.000000 0.142857 0.000000 0.428571 ) *
    7) Spd > 79 53  100.80 Uber ( 0.000000 0.452830 0.000000 0.018868 0.471698 0.056604 )
     14) Spd < 111 39   64.86 OU ( 0.000000 0.589744 0.000000 0.000000 0.358974 0.051282 )
       28) Spd < 99.5 21   38.22 Uber ( 0.000000 0.333333 0.000000 0.000000 0.571429 0.095238 ) *
       29) Spd > 99.5 18   12.56 OU ( 0.000000 0.888889 0.000000 0.000000 0.111111 0.000000 ) *
     15) Spd > 111 14   21.14 Uber ( 0.000000 0.071429 0.000000 0.071429 0.785714 0.071429 ) *
```

[25]:
```
arbol_discretizado$frame
```

	var	n	dev	yval	splits	yprob
	<fct>	<dbl>	<dbl>	<fct>	<chr[,2]>	<dbl[,6]>
1	Total	399	1327.49546	PU	<585 , >585	0.13784461, 0.15538847, 0.3583960, 0.11027569, 0.075187970, 0.16290727
2	Total	332	1001.17563	PU	<502.5, >502.5	0.16566265, 0.09638554, 0.4307229, 0.12349398, 0.015060241, 0.16867470
4	Total	227	583.49855	PU	<429 , >429	0.18942731, 0.06167401, 0.5594714, 0.10572687, 0.008810573, 0.07488987
8	HP	62	89.69699	PU	<77 , >77	0.09677419, 0.04838710, 0.8064516, 0.03225806, 0.000000000, 0.01612903
16	HP	52	53.14916	PU	<46.5 , >46.5	0.03846154, 0.01923077, 0.8846154, 0.03846154, 0.000000000, 0.01923077
32	<leaf>	6	12.13685	PU	,	0.33333333, 0.16666667, 0.5000000, 0.00000000, 0.000000000, 0.00000000
33	<leaf>	46	25.99921	PU	,	0.00000000, 0.00000000, 0.9347826, 0.04347826, 0.000000000, 0.02173913
17	<leaf>	10	21.09840	NU	,	0.40000000, 0.20000000, 0.4000000, 0.00000000, 0.000000000, 0.00000000
9	Generation	165	468.55305	PU	:ac , :bdef	0.22424242, 0.06666667, 0.4666667, 0.13333333, 0.012121212, 0.09696970
18	<leaf>	66	167.49557	PU	,	0.33333333, 0.06060606, 0.4848485, 0.03030303, 0.030303030, 0.06060606
19	Spd	99	279.28281	PU	<107 , >107	0.15151515, 0.07070707, 0.4545455, 0.20202020, 0.000000000, 0.12121212
38	SDef	84	223.96046	PU	<87.5 , >87.5	0.17857143, 0.04761905, 0.4880952, 0.20238095, 0.000000000, 0.08333333
76	SAtk	57	153.52313	PU	<56.5 , >56.5	0.22807018, 0.07017544, 0.4912281, 0.10526316, 0.000000000, 0.10526316
152	<leaf>	10	27.32318	NU	,	0.30000000, 0.20000000, 0.0000000, 0.30000000, 0.000000000, 0.20000000
153	<leaf>	47	108.80410	PU	,	0.21276596, 0.04255319, 0.5957447, 0.06382979, 0.000000000, 0.08510638
77	<leaf>	27	55.76022	PU	,	0.07407407, 0.00000000, 0.4814815, 0.40740741, 0.000000000, 0.03703704
39	<leaf>	15	40.87342	UU	,	0.00000000, 0.20000000, 0.2666667, 0.20000000, 0.000000000, 0.33333333
5	Spd	105	336.23896	UU	<98.5 , >98.5	0.11428571, 0.17142857, 0.1523810, 0.16190476, 0.028571429, 0.37142857
10	SAtk	72	221.93848	UU	<109 , >109	0.13888889, 0.08333333, 0.2222222, 0.12500000, 0.027777778, 0.40277778
20	Generation	53	154.92019	UU	:ab , :cdef	0.15094340, 0.05660377, 0.2452830, 0.16981132, 0.000000000, 0.37735849
40	<leaf>	16	26.60488	UU	,	0.00000000, 0.00000000, 0.1875000, 0.12500000, 0.000000000, 0.68750000
41	<leaf>	37	114.50070	PU	,	0.21621622, 0.08108108, 0.2702703, 0.18918919, 0.000000000, 0.24324324
21	<leaf>	19	53.61011	UU	,	0.10526316, 0.15789474, 0.1578947, 0.00000000, 0.105263158, 0.47368421
11	SDef	33	89.03638	OU	<73.5 , >73.5	0.06060606, 0.36363636, 0.0000000, 0.24242424, 0.030303030, 0.30303030
22	<leaf>	8	11.76995	UU	,	0.12500000, 0.00000000, 0.0000000, 0.00000000, 0.125000000, 0.75000000
23	<leaf>	25	56.94461	OU	,	0.04000000, 0.48000000, 0.0000000, 0.32000000, 0.000000000, 0.16000000
3	Spd	67	152.27146	OU	<79 , >79	0.00000000, 0.44776119, 0.0000000, 0.04477612, 0.373134328, 0.13432836
6	<leaf>	14	28.11879	OU	,	0.00000000, 0.42857143, 0.0000000, 0.14285714, 0.000000000, 0.42857143
7	Spd	53	100.76889	Uber	<111 , >111	0.00000000, 0.45283019, 0.0000000, 0.01886792, 0.471698113, 0.05660377
14	Spd	39	64.85888	OU	<99.5 , >99.5	0.00000000, 0.58974359, 0.0000000, 0.00000000, 0.358974359, 0.05128205
28	<leaf>	21	38.21685	Uber	,	0.00000000, 0.33333333, 0.0000000, 0.00000000, 0.571428571, 0.09523810
29	<leaf>	18	12.55796	OU	,	0.00000000, 0.88888889, 0.0000000, 0.00000000, 0.111111111, 0.00000000
15	<leaf>	14	21.13991	Uber	,	0.00000000, 0.07142857, 0.0000000, 0.07142857, 0.785714286, 0.07142857

3.6 Evaluación

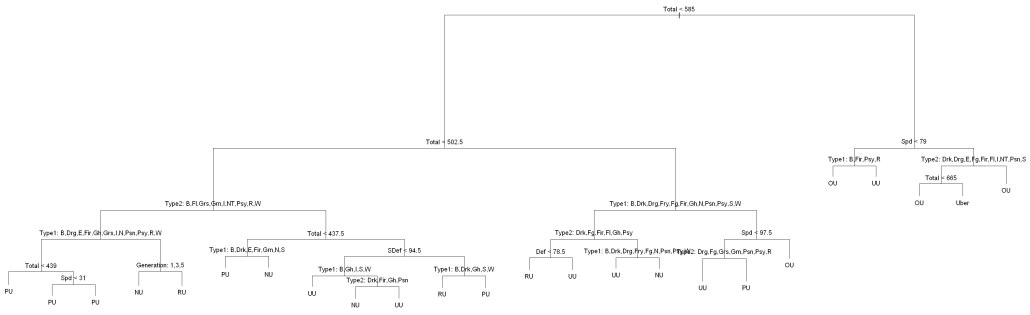
Finalmente, evaluamos el comportamiento de nuestro árbol en el conjunto de pruebas. Pero antes de eso voy a agregar los atributos del tipo de cada Pokémon, ya que dichos campos aumentan el desempeño del árbol.

[26]:
```
arbol_completo = tree(Tier ~ Type1 + Type2 + Total +
              HP + Atk + Def + SAtk + SDef + Spd +
              Generation + Legendary + Mega, data = train)
```

[27]:
```
summary(arbol_completo)
```

```
Classification tree:
tree(formula = Tier ~ Type1 + Type2 + Total + HP + Atk + Def +
    SAtk + SDef + Spd + Generation + Legendary + Mega, data = train)
Variables actually used in tree construction:
[1] "Total"      "Type2"      "Type1"      "Spd"        "Generation"
[6] "SDef"       "Def"
Number of terminal nodes:  24
Residual mean deviance:  1.63 = 611.2 / 375
Misclassification error rate: 0.3183 = 127 / 399
```

[28]:
```
plot(arbol_completo)
text(arbol_completo,pretty=1)
```



Para evaluar crearemos una matriz de confusión, la cual nos mostrará cuantos datos fueron bien o mal clasificados, esto para cada una de nuestras clases.

[29]:
```
# Predecimos cada tier para los datos de pruebas
pred <- predict(arbol_completo,test,type='class')
# Creamos la matriz de confusión
conf_matrix <- with(test,table(pred,test$Tier))
```

[30]:
```
conf_matrix
```

```
pred   NU OU PU RU Uber UU
  NU    1  1  4  2    0  1
  OU    1  4  0  1    3  7
  PU    3  3 28  4    0  3
  RU    3  1  2  1    0  1
  Uber  0  0  2  0    7  1
  UU    2  3  0  3    0  8
```

Esta matriz nos muestra los valores predichos (en las filas) y los valores reales (columnas). Por ejemplo, el valor [0][1] de la matriz corresponde a los Pokémon que fueron clasificados como NU, pero en verdad eran OU.

De acá podemos ver para los Pokémon pertenecientes de PU, el modelo clasificó correctamente 28 Pokémon, y se equivocó en 8 (4 en NU, 2 en RU y 2 en Uber). En Uber también el modelo presenta un buen desempeño, pero en las otras tiers el modelo se equivoca más de lo que acierta.

Con esta matriz podemos calcular el `accuracy` del modelo, sumando la diagonal (aciertos) y dividiendolo por la cantidad de datos en el conjunto. El error de clasificación se puede calcular como $1 - accuracy$.

[31]:
```
acc <- sum(diag(conf_matrix))/nrow(test)
miss_class_error <- 1 - acc
```

[32]:
```
acc
```

0.49

[33]:
```
miss_class_error
```

0.51

Podemos ver que el modelo presenta un 51% de error de clasificación en el conjunto de pruebas, pero en el conjunto de entrenamiento tenía un 32% (ver summary(arbol)). Esto nos indica que el modelo está sobreajustado a los datos de entrenamiento.

Hay distintas maneras con la cual se podría atacar este sobreajuste: limitar la altura del árbol al momento de construirlo, podarlo una vez construido, agregar más datos, etc. También el conocimiento del dominio te podría ayudar, muchas veces en el Pokémon Competitivo no importan tan solo los stats que tiene un Pokémon, también toman mucha relevancia los ataques que tiene un Pokémon o la habilidad. Agregar dichos datos podría llegar a mejorar los resultados obtenidos por nuestro árbol.

4 Cambios sobre el dataset original

- Cambio en los nombres de las columnas.
- Relleno de valores faltantes en columna Type2, ya que no todos los Pokémon tienen segundo tipo.
- Cambio del nombre de Farfetch'd a Farfetchd.

- Quien conoce un poco de las tiers de Smogon sabe que estas incluyen Ban List para cada tiers, estas las modifiqué agregando cada Pokémon de una determinada Ban List a la tier siguiente. En el caso de Mega-Rayquaza, fue incluído dentro de Uber. Esto fue realizado para disminuir el número de clases, ya que tener más clases complejiza el problema.