R-PL3

Gabriel López, Sergio Sanz, Álvaro Zamorano

5 de noviembre de 2019

1. Ejercicio realizado en clase.

Para obtener la **función de clasificación** mediante el algoritmo construcción de **árboles de decisión de Hunt** es necesario usar los paquetes `rpart` y `tree`. Estos paquetes hay que descargarlos desde la página de CRAN y para instalarlos hay que ejecutar el siguiente código:

```
> install.packages("./Paquetes/rpart_4.1-15.zip")
> install.packages("./Paquetes/tree_1.0-40.zip")
```

De esta forma, los paquete únicamente estarán instalados. Para poder usarlos es necesario cargarlos:

```
> library(rpart)
> library(tree)
```

- Con rpart obtendremos las particiones recursivas para la clasificación y los árboles de decisión.
- Con tree, los árboles de clasificación y regresión.

1.1. Función de clasificación.

Los datos a usar en este primer ejercicio se componen de 9 calificaciones de estudiantes compuestas por Teoría, Laboratorio, Prácticas y Calificación Global.

Para introducir estos datos en el algoritmo a usar es necesario tener un fichero `.txt` con el siguiente aspecto.

Suceso	Teoría	Lab	Prac	Calif
s1	A	A	B	Ap
s2	A	B	D	Ss
s3	D	D	C	Ss
s4	D	D	A	Ss
s5	B	C	B	Ss
s6	C	B	B	Ap
s7	B	B	A	Ap
s8	C	D	C	Ss
s9	B	A	C	Ss

Procedemos a leer dicho fichero .txt mediante el uso de la función *read.table.*

```
> calificaciones<-read.table("./Datos/Calificaciones.txt")
```

Para asegurarnos de que todo irá bien a la hora de realizar la clasificación, convertimos los datos leídos en un **dataframe.**

```
> muestra<-data.frame(calificaciones)
```

Nuestros datos ya se encuentran preparados para aplicarles la función **rpart.** Es importante destacar el uso de **minsplit** ya que disponemos de una muestra con un número muy reducido de datos, este parámetro indica el número mínimo de obervaciones que deben existir en un nodo para que se considere a la hora de clasificar. Por otra parte, la función rpart usa como medida de impureza por defecto **Gini.**

```
> clasificacion<-rpart(Calif~.,data=muestra,method="class",minsplit=1)
```

Para mostrar el árbol de clasificación hacemos uso de una función que hemos definido, pero para poder usarla en primer lugar es necesario instalar el paquete **rpart.plot.**

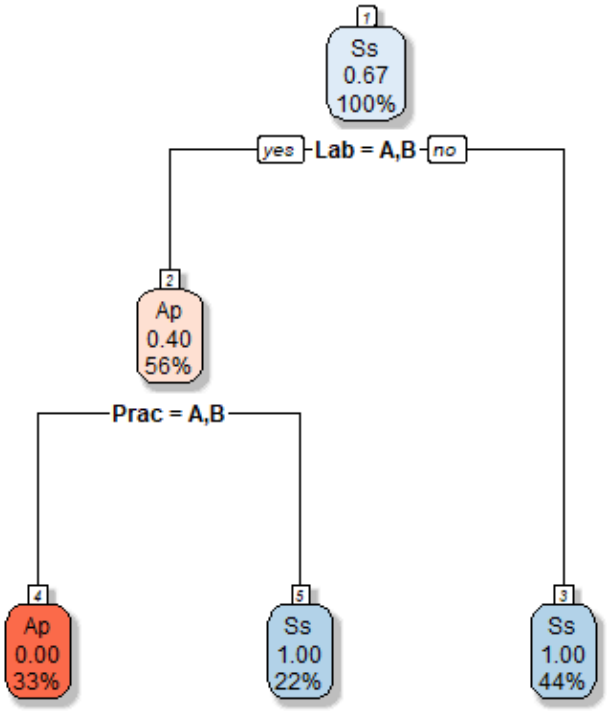
```
> install.packages("./Paquetes/rpart.plot_3.0.8.zip")
> library(rpart.plot)
```

Dicha función es:

```
> source("Funciones/plotTree.R")
> plotTree

function(tree, ruta) {

    png(paste("./tmp/",ruta,sep=""))

    rpart.plot(tree, box.palette="RdBu", shadow.col="gray", nn=TRUE)

    dev.off()
}
```

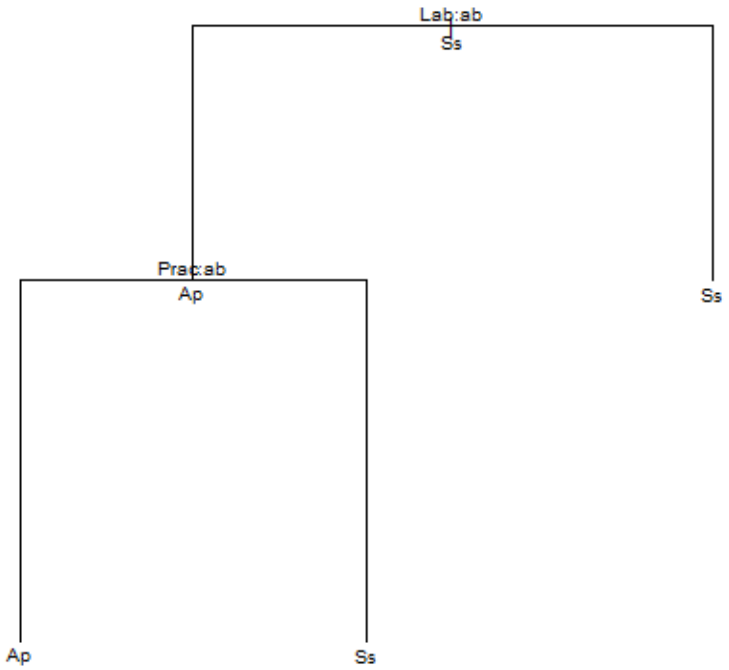
Procedemos a su ejecución.

```
> plotTree(clasificacion, "classTree.png")
```



Por último, se aplica la función **tree** a nuestros datos, para que esta use la medida de impureza `Gini` lo indicamos en el parámetro split.

```
> clasificaciontree<-tree(Calif~.,data=muestra,mincut=1,minsize=2,split="gini")
> source("./Funciones/plotT.R")
> plotT(clasificaciontree, "classT.png")
```



Se puede observar que ambas funciones de clasificación son iguales independientemente de la función que se use (rpart o tree).

1.2. Análisis de regresión lineal.

En este caso trabajaremos con datos de planetas, en concreto su Radio y su Diámetro. Los planetas de los que se tienen los datos son: Mercurio, Venus, Tierra y Marte, y el .txt del que se leen dichos datos tiene el aspecto que sigue.

Planeta	Radio	Diámetro
Mercurio	2.4	5.4
Venus	6.1	5.2
Tierra	6.4	5.5
Marte	3.4	3.9

Al igual que anteriormente, es necesario leer dicho fichero y pasarlo a dataframe.

```
> planetas<-read.table("./Datos/Planetas.txt")
> muestraP<-data.frame(planetas)
```

El análisis de regresión se hace mediante el uso de la función **lm** contenida en el paquete stats. Cabe destacar que el primero de sus argumentos es de tipo *fórmula* donde una expresión de la forma y ~ model se interpreta como una especificación de que la respuesta `y` está modelada por un predictor lineal especificado simbólicamente por model, es decir, en nuestro caso model=x por lo que su ejecución queda como:

```
> regresionP<-lm(D~R,data=muestraP)
> regresionP

Call:
lm(formula = D ~ R, data = muestraP)

Coefficients:
(Intercept)            R
     4.3624       0.1394
```

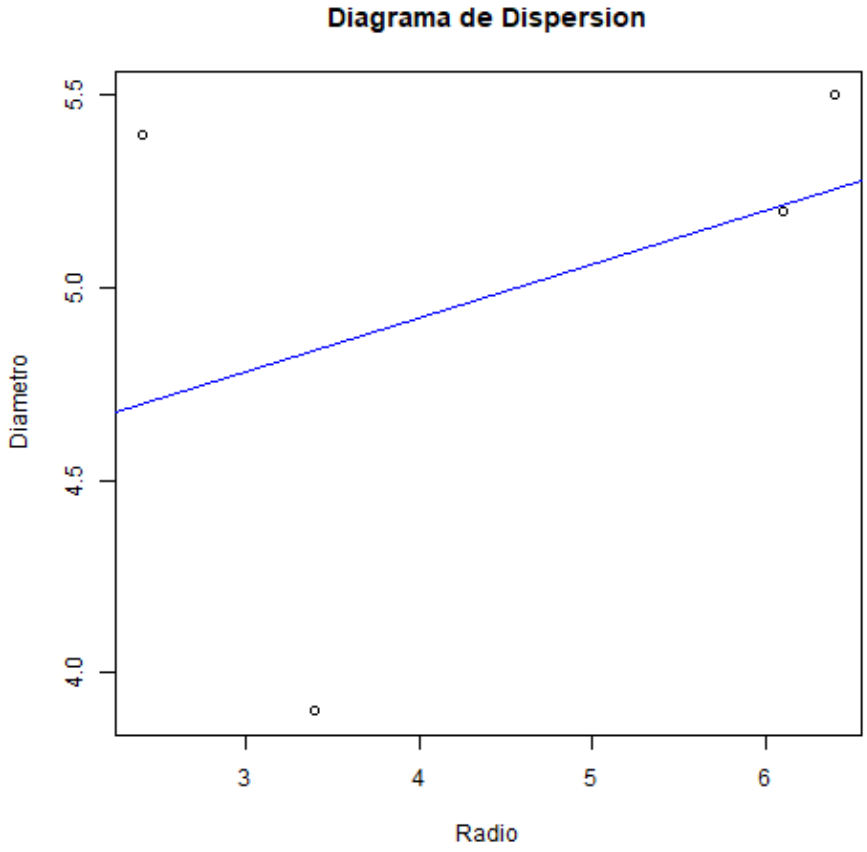
De acuerdo a la ecuación de una recta `y=a+b*x`, el primero de los coeficientes es el término independiente (a), y el segundo de ellos la b.

Para mostrar el gráfico de dispersión y la recta de ajuste es necesario hacer uso de varias librerías.

```
> library(foreign)
> library(ggplot2)
> library(psych)
```

Estas librerías se usan en funciones externas usadas para representar los gráficos requeridos. En dicha función es necesario indicar las columnas de los datos con los que se realizará la regresión.

```
> source("Funciones/plotDisp.R")
> plotDisp(planetas,1,2,regresionP,"Radio","Diametro","regPlanetas.png")
```



2. Segunda parte

2.1. Función de clasificación.

En este ejercicio usaremos datos correspondientes a ventas de coches, en concreto son 10 muestras compuestas por: TipoCarnet, NúmeroRuedas, NúmeroPasajeros y TipoVehículo.

Para introducir estos datos en el algoritmo a usar es necesario tener un fichero `.txt` con el siguiente aspecto.

Vehículo	TipoCarnet	NúmeroRuedas	NúmeroPasajeros	TipoVehículo
v1	B	4	5	Coche
v2	A	2	2	Moto
v3	N	2	1	Bicicleta
v4	B	6	4	Camión
v5	B	4	6	Coche
v6	B	4	4	Coche
v7	N	2	2	Bicicleta
v8	B	2	1	Moto
v9	B	6	2	Camión
v10	N	2	1	Bicicleta

Procedemos a leer dicho fichero como anteriormente y pasarlo a dataframe.

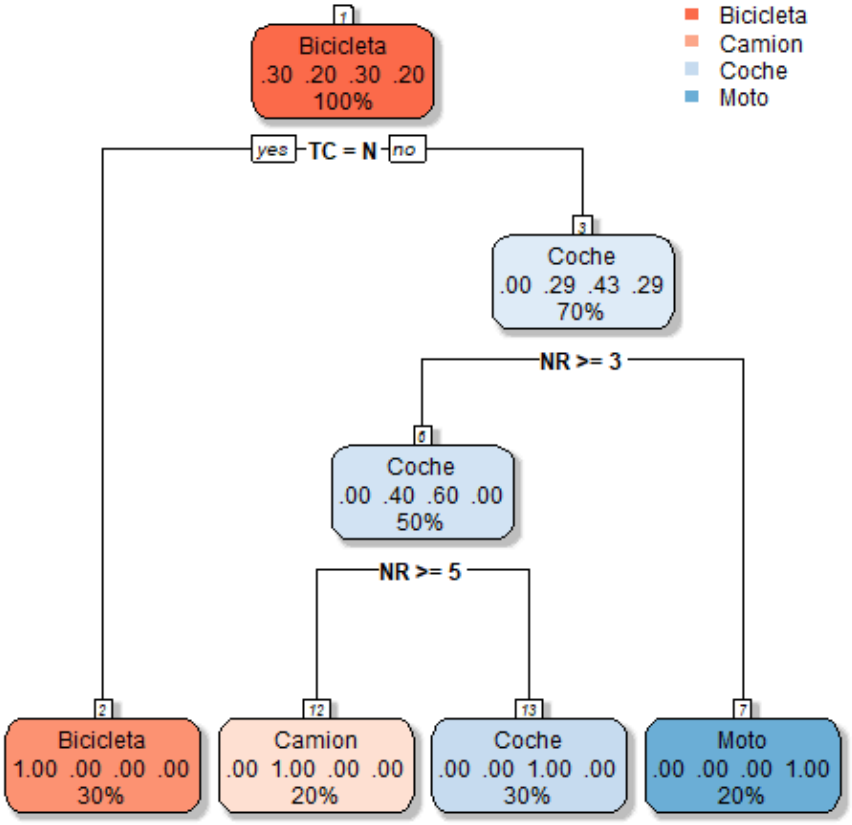
```
> vehiculos<-read.table("./Datos/Vehiculos.txt")
> muestraV<-data.frame(vehiculos)
```

Nuestros datos ya se encuentran preparados para aplicarles la función **rpart.** La clasificación a obtener será el tipo de vehículo al que pertenece cada uno de ellos.

```
> clasV<-rpart(TV~.,data=muestraV,method="class",minsplit=1)
```

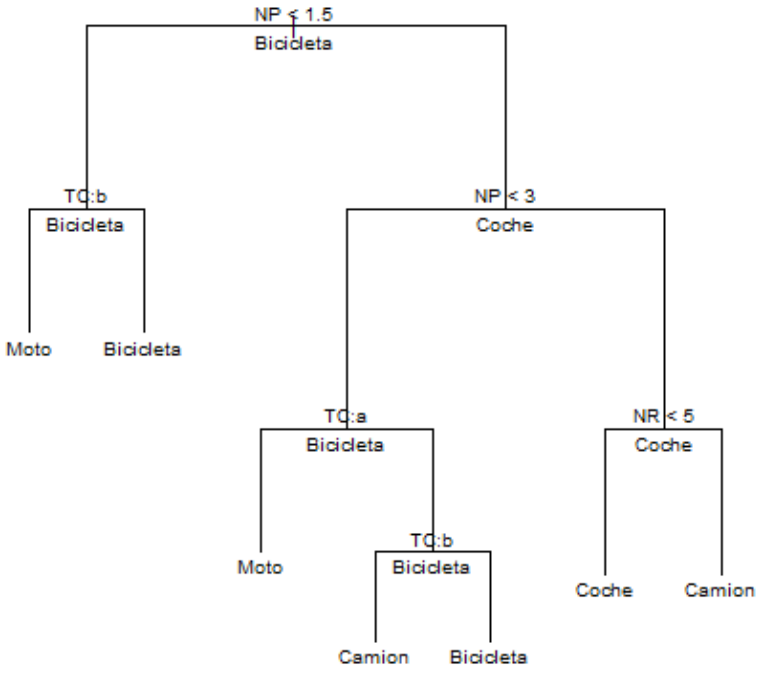
Procedemos a mostrar el árbol de clasificación.

```
> plotTree(clasV, "classTreeV.png")
```



Por último, se aplica la función **tree** a nuestros datos.

```
> classTV<-tree(TV~.,data=muestraV,mincut=1,minsize=2,split="gini")
> plotT(classTV, "classTV.png")
```



En este caso las funciones de clasificación son distintas si usamos rpart o usamos tree. La obtenida por la primera de ellas es igual a la que se obtiene al realizar de clasificación manualmente; la que nos muestra tree es una función mucho más engorrosa y necesita muchos más pasos para obtener la clasificación final.

2.2. Análisis de regresión lineal.

En este caso tenemos que hacer un análisis de regresión lineal para 4 muestras distintas compuestas por pares de datos.

Como en ocasiones anteriores, procedemos a leer dichas muestras y pasarlas a dataframe.

```
> pares<-read.table("./Datos/Pares.txt")
> muestraPS<-data.frame(pares)
```

El análisis de regresión se hace mediante el uso de la función **lm.** En este caso será neceario hacer cuatro análisis diferentes.

```
> (r1<-lm(V2~V1,data=muestraPS))
```

```
Call:
lm(formula = V2 ~ V1, data = muestraPS)

Coefficients:
(Intercept)           V1
     3.0001       0.5001

> (r2<-lm(V4~V3,data=muestraPS))

Call:
lm(formula = V4 ~ V3, data = muestraPS)

Coefficients:
(Intercept)           V3
      3.001        0.500

> (r3<-lm(V6~V5,data=muestraPS))

Call:
lm(formula = V6 ~ V5, data = muestraPS)

Coefficients:
(Intercept)           V5
     3.0025       0.4997

> (r4<-lm(V8~V7,data=muestraPS))

Call:
lm(formula = V8 ~ V7, data = muestraPS)

Coefficients:
(Intercept)           V7
     3.0017       0.4999
```

De acuerdo a la ecuación de una recta `y=a+b*x`, el primero de los coeficientes es el término independiente (a), y el segundo de ellos la b.

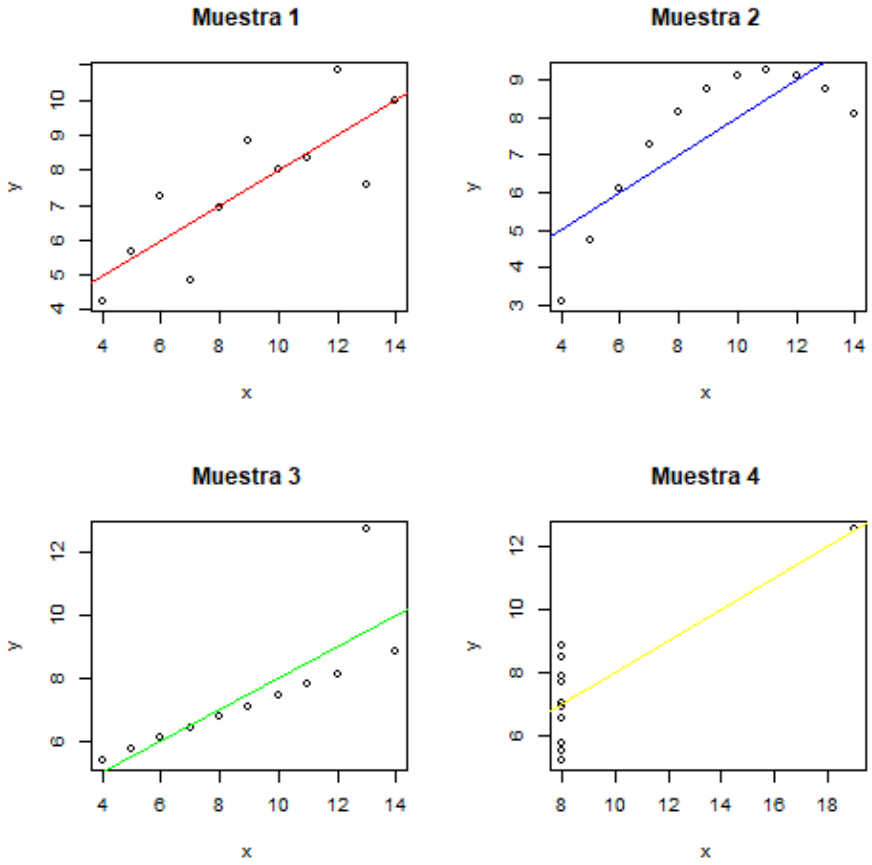
Mostraremos estos análisis en una misma figura mediante el uso de:

```
> source("Funciones/plotDisp2.R")
> plotDisp2

function(data, r1, r2, r3, r4, ruta) {

    png(paste("./tmp/",ruta,sep=""))

    par(mfrow = c(2, 2))

    plot(data[,1], data[,2], xlab="x", ylab="y", main="Muestra 1")
    abline(r1, col = "red")
```

```
    plot(data[,3], data[,4], xlab="x", ylab="y", main="Muestra 2")
    abline(r2, col = "blue")

    plot(data[,5], data[,6], xlab="x", ylab="y", main="Muestra 3")
    abline(r3, col = "green")

    plot(data[,7], data[,8], xlab="x", ylab="y", main="Muestra 4")
    abline(r4, col = "yellow")

    dev.off()
}
```

Procedemos a su ejecución.

```
> plotDisp2(pares, r1, r2, r3, r4, "rPares.png")
```



2.3. Desarrollo por parte del alumno.

Para la parte de **clasificación** usaremos datos de *Setas*, las cuales clasificaremos en función de si son venenosas (p) o comestibles (e), es decir, los tipos de

setas que encontramos en el archivo. Procedemos a leer dicho fichero como en prácticas anteriores haciendo uso de la librería readr.

```
> install.packages("readr")
> library("readr")
> setas<-read.csv("./Datos/mushrooms.csv")
```

Los atributos que encontramos en este documento son:

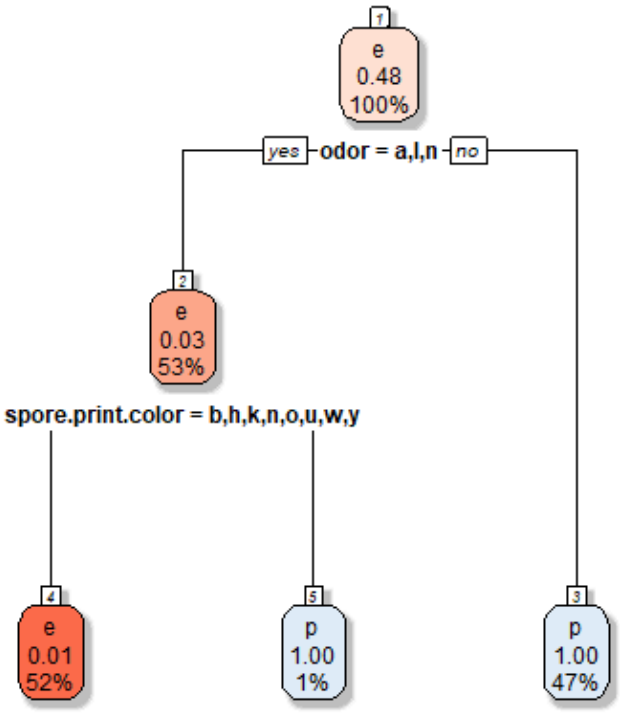
- class: edible=e, poisonous=p
- cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
- cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s
- cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e, white=w,yellow=y
- bruises: bruises=t,no=f
- odor: almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s
- gill-attachment: attached=a,descending=d,free=f,notched=n
- gill-spacing: close=c,crowded=w,distant=d
- gill-size: broad=b,narrow=n
- gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p, purple=u,red=e,white=w,yellow=y
- stalk-shape: enlarging=e,tapering=t
- stalk-root: bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?
- stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s
- stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s
- stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e, white=w,yellow=y
- stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e, white=w,yellow=y
- veil-type: partial=p,universal=u
- veil-color: brown=n,orange=o,white=w,yellow=y
- ring-number: none=n,one=o,two=t
- ring-type: cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z
- spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u, white=w,yellow=y
- population: abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y

- habitat: grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d

```
> clasM<-rpart(class~.,data=setas,method="class")
```

Procedemos a mostrar el árbol de clasificación.

```
> plotTree(clasM, "classTreeM.png")
```



Función de clasificación obtenida:

- Si el olor no es anisado ni almendrado, se trata de una seta venenosa.
- En caso contrario, si su color no es verde, estamos ante una seta comestible, para el resto es venenosa.

Para usar la función `predict` con el fin de generar predicciones a partir de un modelo entrenado, necesitamos un set de entrenamiento y otro de prueba que compruebe la eficacia de las predicciones. Para obtener estos sets utilizamos la función `sample_frac` la cual pertenece al paquete **dplyr**, además por el

parámetro *newdata* es necesario el uso del paquete **e1071.** Posteriormente, con la función `confusionMatrix` obtenemos la matriz de confusión del conjunto de pruebas, dicha función se encuentra en el paquete **caret**.

Instalación de paquetes.

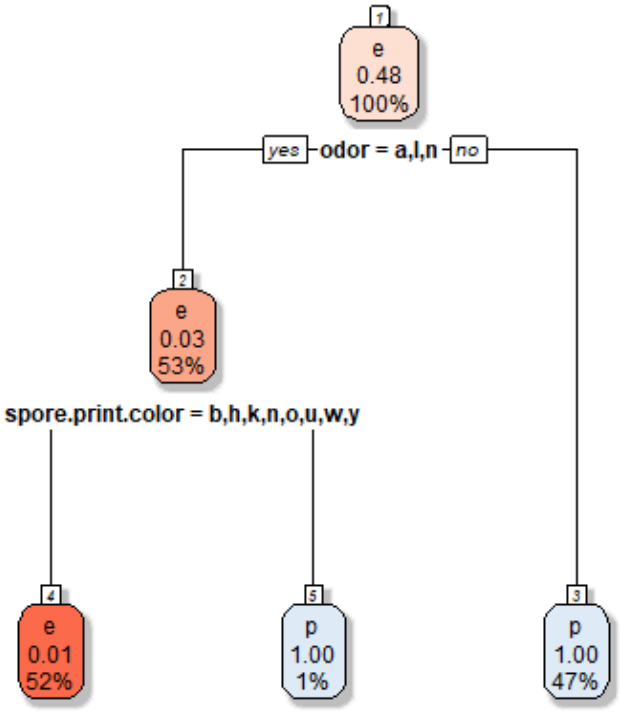
```
> install.packages("dplyr")
> library("dplyr")
> install.packages("e1071")
> library("e1071")
> install.packages("caret")
> library("caret")
```

Separación de los datos para el set de entrenamiento y pruebas.

```
> setasTrain<-sample_frac(setas,.7)
> setasP<-setdiff(setas,setasTrain)
```

Mostramos la función de clasificación perteneciente al 70 % del conjunto de datos.

```
> clas70<-rpart(class~.,data=setasTrain,method="class")
> plotTree(clas70, "classTree70.png")
```

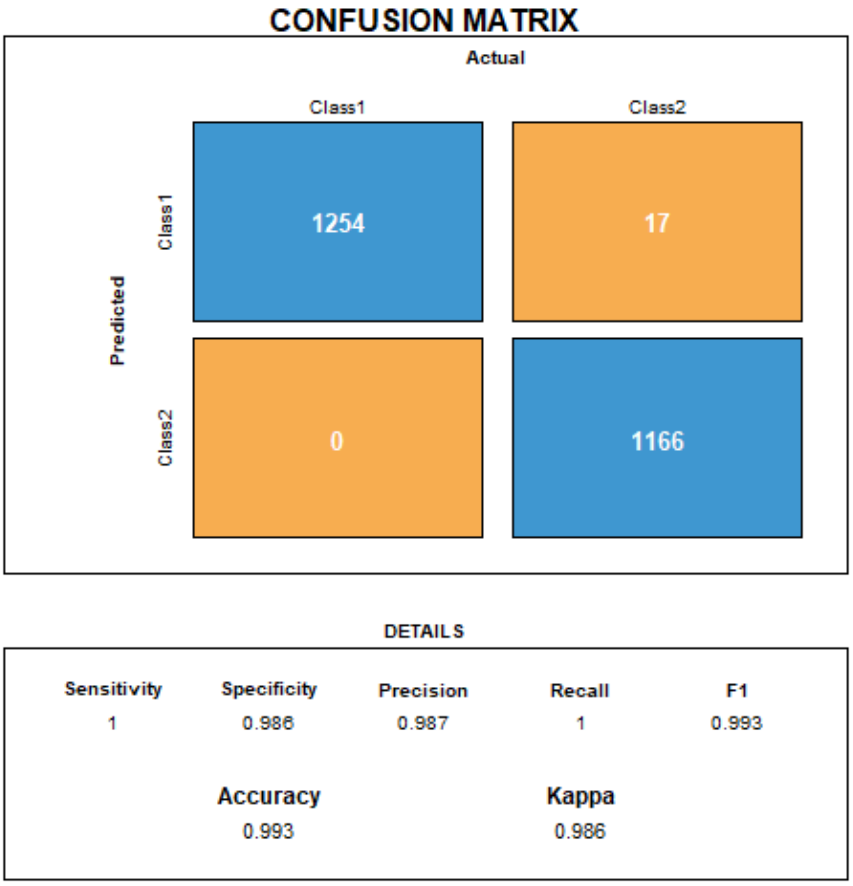


Ejecución de la predicción y la posterior matriz de confusión.

```
> prediccion<-predict(clas70, newdata=setasP, type="class")
> cm<-confusionMatrix(prediccion, setasP[["class"]])
```

Una vez calculada la matriz de confusión procedemos a mostrarla gráficamente.

```
> source("./Funciones/drawCM.R")
> drawCM(cm,"cmSetas.png")
```



En la parte del **análisis de regresión lineal** usaremos datos pertenecientes al clima, de los cuales usaremos la temperatura y humedad.

En primer lugar debemos leer estos datos contenidos en un .csv y convertirlos en dataframe.

```
> w<-read.csv("./Datos/weather.csv")
> weather<-data.frame(w)
```

Guardamos en dos variables los datos de temperatura y humedad para facilitar su utilización.

```
> t<-weather$tempm
> h<-weather$hum
```

Se han realizado funciones para calcular los datos necesarios a la hora de obtener la recta de regresión. Hemos observado que el valor de la Covarianza no es correcto y por ello hemos hecho una función con dicho fin. Las funciones son las siguientes:

```
> source("./Funciones/anovaR.R")
> anovaR

function(x,y) {
    mediaX<-mean(x)
    mediaY<-mean(y)

    recta<-calcularRecta(x,y,mediaX,mediaY)

    r2<-(ssR(x,mediaY,recta))/(ssY(y,mediaY))

    return(r2)
}

> ssR

function (x,mY,r) {
    sum<-0
    for (data in x){
        sum<-sum+((r$a+r$b*data)-mY)^2
    }

    return(sum)
}

> ssY

function (y, mY) {
    sum<-0
    for (data in y){
        sum<-sum+(data-mY)^2
    }

    return(sum)
}

> calcularRecta

function (x,y,mX,mY) {
    b<-covarianza(x,y,mX,mY)/(desviacion(x))^2
    a<-mY-b*mX

    l<-list("a"=a, "b"=b)

    return(l)
}

> desviacion

function (var) {
    media<-mean(var)
    num<-0
```

```
    for (data in var){
        num<-num+((data-media)^2)
    }

    s<-sqrt(num/length(var))

    return(s)
}
```

```
> covarianza
```

```
function(x,y,mX,mY){
    sz<-length(x)
    sum<-0
    for(i in 1:sz){
        sum<-sum+(x[i]*y[i])
    }

    cov <- (sum/sz) - mX*mY

    return(cov)
}
```

```
> correlacion
```

```
function(cov,x,y){
    cr<-cov/((desviacion(x))*desviacion(y))

    return(cr)
}
```

Cálculo de la **covarianza** y la **correlación.**

```
> (cov<-covarianza(h,t,mean(h),mean(t)))
```

```
[1] -70.46875
```

```
> (correlacion(cov,h,t))
```

```
[1] -0.9778414
```

El análisis de la covarianza resulta muy complicado ya que se trata de una medida que depende mucho del dominio de los datos a estudiar, es decir, no se encuentra normalizada. Por ello surge la correlación, una medida cuyo valor se encuentra en el intervalo [-1,1]. Un valor de la correlación igual a -1 (recta con pendiente negativa) o 1 (recta con pendiente positiva), indicará un buen ajuste de los datos a la recta de regresión.

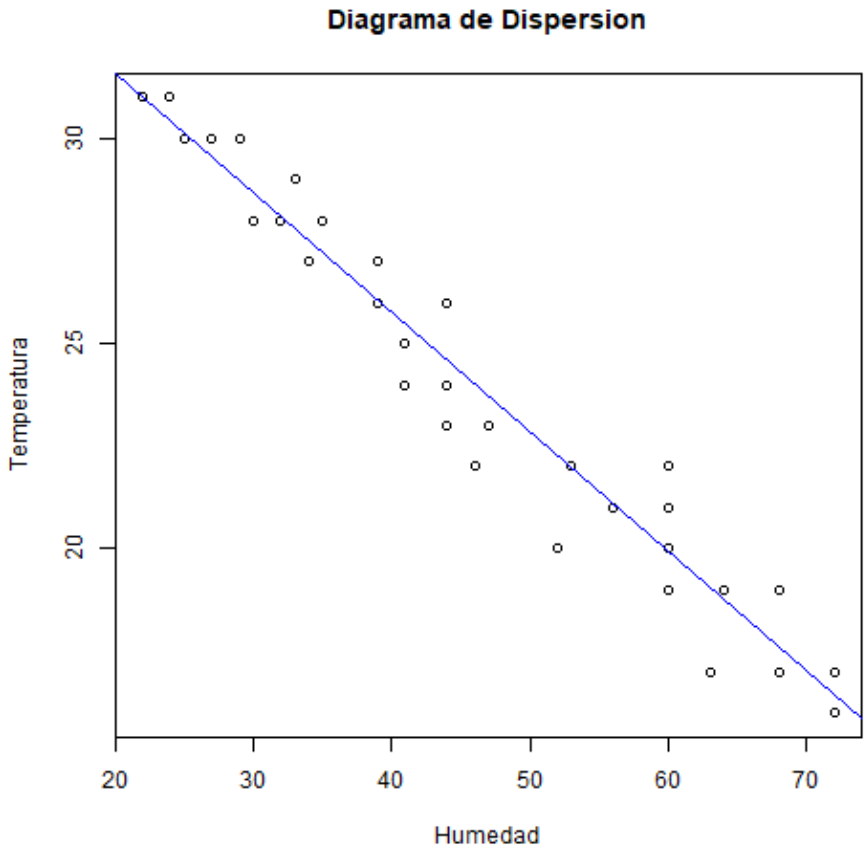
Cálculo de los coeficientes de la recta de acuerdo a `y=a+b*x`.

```
> (recta<-calcularRecta(h,t,mean(h),mean(t)))
```

```
$a
[1] 37.39831

$b
[1] -0.2909511
```

Mostrar gráficamente la recta de regresión.

```
> plotDisp(weather,7,12,lm(t~h,data=weather),"Humedad","Temperatura","regWeather.png")
```



El segundo paso consiste en realizar el análisis **ANOVA.** Obtendremos a partir de la *Dispersión SSR* y la *Dispersión SSY* la **Correlación cuadrada.**

```
> (anovaR(h,t))

[1] 0.9561739
```

El valor de la correlación cuadrada se encuentra en el intervalo [0,1], indicando 1 un ajuste perfecto entre las variables. Como se puede observar, dicho ajuste es casi perfecto.

Por último, calculamos el **Error estándar** con la función:

```
> source("./Funciones/errorE.R")
> errorE

function(x,y,r) {
    sz<-length(x)
    sum<-0
    for (i in 1:sz){
        sum<-sum+((r$a+r$b*x[i])-y[i])^2
    }

    sR<-sqrt(sum/sz)

    return(sR)
}
```

Procedemos a su cálculo.

```
> (errorE(h,t,recta))

[1] 0.9694076
```

Un valor, de dicho error, próximo a 0 indica un buen ajuste de la recta.