USO DE HERRAMIENTAS INFORMÁTICAS PARA LA RECOPILACIÓN, ANÁLISIS E INTERPRETACIÓN DE DATOS DE INTERÉS EN LAS CIENCIAS BIOMÉDICAS

Módulo 8

Aprendizaje automatizado con R
Creación y edición de bibliotecas

Alfredo Rigalli
Maela Lupo

Centro Universitario de Estudios Medioambientales
Facultad de Ciencias Médicas
Universidad Nacional de Rosario

2023

ISBN 978-987-86-3474-6

USO DE HERRAMIENTAS INFORMÁTICAS PARA LA RECOPILACIÓN, ANÁLISIS E INTERPRETACIÓN DE DATOS DE INTERÉS EN LAS CIENCIAS BIOMÉDICAS

MODULO 8

Aprendizaje automatizado con R
Creación y edición de funciones y bibliotecas

Alfredo Rigalli
Maela Lupo

Centro Universitario de Estudios Medioambientales
Facultad de Ciencias Médicas
Universidad Nacional de Rosario

Uso de herramientas informáticas para la recopilación, análisis e interpretación de datos de interés en las ciencias biomédicas : módulo 9 : Comunicación con R / Alfredo Rigalli ... [et al.]. - 1a edición para eel alumno - Rosario : Alfredo Rigalli, 2020.
Libro digital, PDF

Archivo Digital: online
ISBN 978-987-86-3474-6

1. Matemática Aplicada. 2. Matemática Estadística. I. Rigalli, Alfredo
CDD 510

AUTORES

Lupo, Maela: Licenciada en biotecnología y Doctora en Ciencias Biomédicas. Investigadora del Laboratorio de Biología Ósea y del Centro Universitario de Estudios Medioambientales y docente de la cátedra de Química Biológica de la Facultad de Ciencias Médicas de la Universidad Nacional de Rosario.

Rigalli, Alfredo: Bioquímico y Doctor en bioquímica. Investigadora del Laboratorio de Biología Ósea y del Centro Universitario de Estudios Medioambientales y docente de la cátedra de Química Biológica de la Facultad de Ciencias Médicas de la Universidad Nacional de Rosario. Investigador independiente del Consejo de Investigaciones de la UNR y del CONICET

Tabla de contenidos

Módulo 8. Clase 1....7
Aprendizaje automatizado....7
Redes neuronales artificiales....9
Preparación de los datos....10
Entrenamiento de la red neuronal....15
Prueba del entrenamiento....19
Resumiendo todos los pasos....21
Predicción en datos solo con predictores....23
Módulo 8. Clase 2....1
Continuación redes neuronales para clasificación....1
Resumiendo todos los pasos....1
Algunas pruebas con cambios de parámetros....2
Parámetros originales....2
subimos el número de neuronas a 20....4
Dos capas de neuronas ocultas....5
Con 10 ciclos de entrenamiento....7
Con una capas ocultas de 50 neuronas....8
La función set.seed()....10
Grupo de entrenamiento y prueba....10
Módulo 8. Clase 3....14
Regresión con redes neuronales....15
Preparación de los datos....15
Entrenamiento de la red neuronal....18
Prueba del entrenamiento....19
Estimación del valor de arsénico en base a mediciones de fluoruro y alcalinidad....22
Modo optimo de trabajo....23
Módulo 8. Clase 4....24
Arboles de decisión....24
Preparación de los datos....25
Preparación set de entrenamiento y prueba....27
Entrenamiento del árbol de decisión....28
Prueba del modelo....33
Estimación del origen del agua en función de predictores....35
Módulo 8. Clase 5....37
Arboles de decisión continuación....37
La importancia de la semilla....39
Otras interpretaciones de rpart.plot....44
Módulo 8. Clase 6....50
Random Forest....50
Preparación de los datos....50
Entrenamiento del modelo....53
Testeo del modelo....57
Influencia del parámetro ntree en el entrenamiento....58
ntree=500....58
ntree=200....59
ntree=2000....59
ntree=100....60
Entrenamiento y prueba con variables acotadas....61

Módulo 8. Clase 7....63
Random Forest....63
Preparación de los datos....63
Entrenamiento del modelo....65
Testeo del modelo....68
Módulo 8. Clase 8....72
Funciones en R....72
Funciones definidas por el usuario....72
Modificación de una función definida por el usuario....73
Funciones no definidas por el usuario....75
La función sum()....75
Modificación de función instalada....78
Modificando t.test....80
Modificando archivos con código fuente....88
Módulo 8. Clase 9....97
Creación de un paquete....97
Partes de la biblioteca....97
Ordenamiento del directorio....99
Creación de paquete....100
Comenzando con edición de archivos .Rd....102
Modificación de archivos .Rd....103
El archivo DESCRIPTION....103
Modificación carpeta man....105
Modificación del archivo datosR89.Rd....106
Modificación del archivo desvioestandar.Rd....107
Modificación del archivo media.Rd....108
Modificación del archivo mediaSD-package.Rd....108
Control de archivo NAMESPACE....109
Instalación del paquete en la computadora....109
Prueba de instalación del paquete....110
Carga de la biblioteca....114
Preparación del archivo tar.gz....116
Instalación y chequeo de funcionamiento en Windows....118
Envio del paquete a RCRAN....118

Módulo 8. Clase 1

Aprendizaje automatizado

El aprendizaje lo podemos entender como respuestas que damos a situaciones en base a lo vivido. Por ejemplo, si en el laboratorio tenemos una platina de calentamiento, bastará solo una experiencia de tomarla tocando la platina, para que la próxima vez que tengamos que trasladarla nos aseguremos que está apagada, que ya hace rato que está sin uso y por las dudas la tomaremos de abajo, sin tocar la parte que toma temperatura.

En la ciencia las computadoras pueden aprender por nosotros y solucionarnos muchos problemas, llamamos a esto machine learning, aprendizaje automático o inteligencia artificial. Aunque puede haber alguna discrepancia entre los expertos en el tema, básicamente son algoritmos que pueden ser procesados por computadoras y que arribarán a conclusiones en base a observación de datos. La gran ventaja es que estos algoritmos pueden ser ejecutados, repetidos y chequeados miles de millones de veces más rápido y efectivamente que nuestra mente.

Los mecanismos de machine learning se agrupan en tres modelos básicos: regresión lineal, modelos basados en árboles y redes neuronales.

Los objetivos de machine learning son básicamente dos: clasificación y predicción. En las clases nos focalizaremos en un problema concreto de nuestro centro de investigación y veremos como el aprendizaje automatizado dio soluciones. Por supuesto, sabemos que estamos en el principio del aprovechamiento de esta tecnología.

La necesidad de implementar aprendizaje automatizado en el CUEM, surgió de la mano de la necesidad de realizar la medición de arsénico en agua. El arsénico, si bien es un componente común del agua potable, existen niveles máximos permitidos, por encima de los cuales comienzan a aparecer sus efectos adversos. El arsénico se mide por una técnica espectrofotométrica, que involucra equipos frágiles, muchos pasos y controles de calidad que hacen muy costosa la medición, especialmente si se desea medir una sola muestra. En el CUEM se procede a juntar 10 muestras y luego allí se hacen las mediciones, de esta manera se reduce el costo por muestra en al menos 10 veces. El problema radica en que juntar 10 muestras en algunas épocas del año, puede llevar varias semanas y esto retrasa la entrega de informes. Tratándose de la medición de arsénico, uno de los 10 tóxicos más importantes hallados en el agua, el retraso no es aceptable.

Se introdujeron herramientas de aprendizaje automatizado para poder anticipar un valor de arsénico en agua, sin haberlo medido, pero utilizando los valores de otras mediciones que son rápidas y económicas. Luego cuando se juntan las muestras suficientes, se hace la medición y se envía el informe definitivo. La Figura 1 esquematiza el problema y la solución. Ni bien la muestra ingresa, hay determinaciones que son muy rápidas, precisas y económicas, que además se pueden realizar sin un entrenamiento excesivo. Se hacen las determinaciones y la "inteligencia artificial" de Atlantis 3.0, nuestro software de administración del laboratorio de aguas, incorpora esos valores, hace un nuevo aprendizaje utilizando la base de datos, toma los valores medidos de las variables de la muestra (en el esquema: pH, alcalinidad y fluoruro) y genera un valor de arsénico en segundos. Si el valor de arsénico es "aceptable", se sigue adelante con el tratamiento de la muestra y se realizará la medición de arsénico "cuando se hayan juntado 10 muestras". Si "no es aceptable", Atlantis genera una alerta y envia un mail al encargado de la medición de arsénico, indicándole que aunque no tenga 10 muestras "debe medir la muestra inmediatamente".

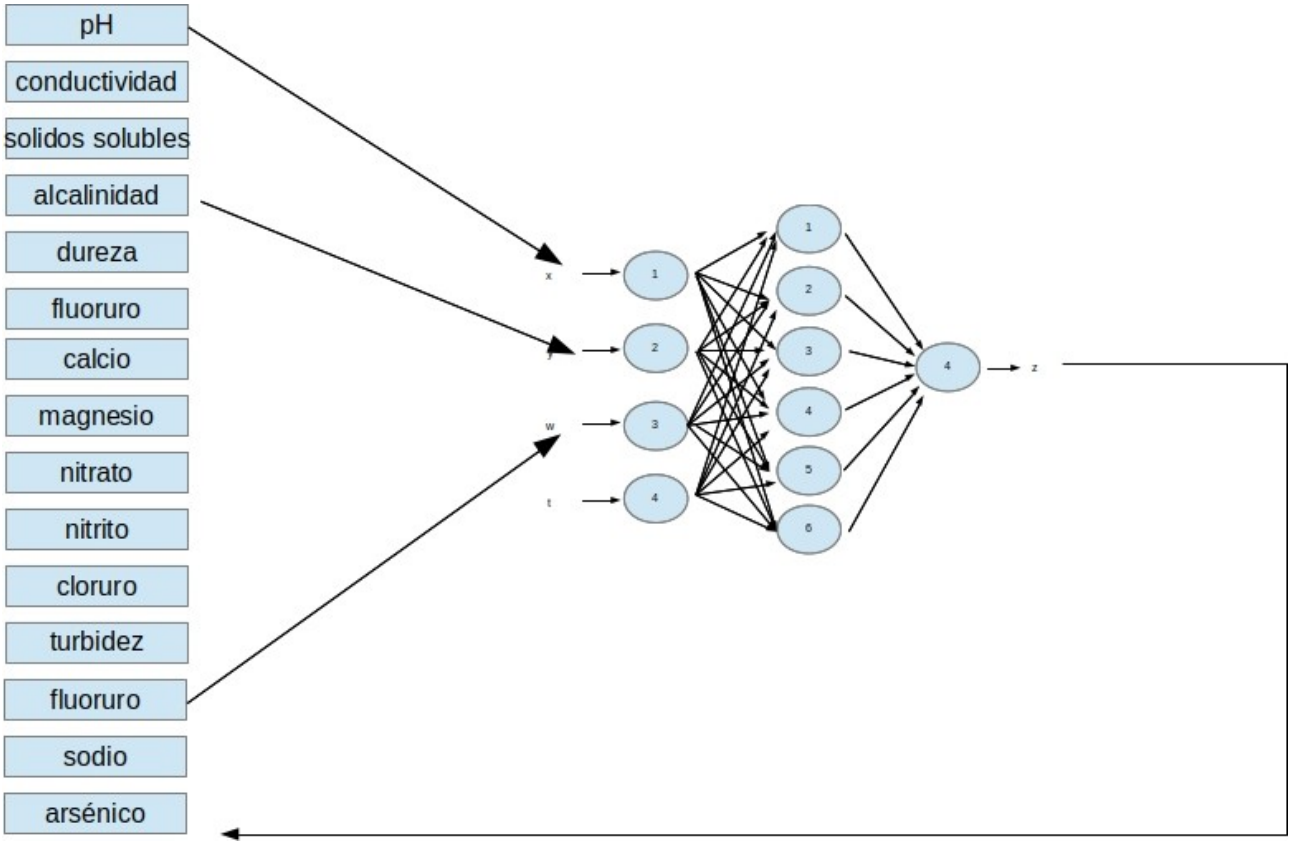


Figura 1. A la izquierda se muestran diferentes mediciones que se realizan sobre una muestra de agua. Un sistema de aprendizaje automatizado basado en redes neuronales toma tres de ellos y genera un valor de arsénico, en base a un aprendizaje realizado previamente con valores de las variables involucradas.

Veremos en base a este problema las dos situaciones planteadas: clasificación y predicción.

En el primer enfoque solo nos propondremos que "la inteligencia artificial de Atlantis 3.0" nos permita clasificar la muestra de agua en dos niveles

1- concentración de arsénico menor al límite impuesto por la Organización Mundial de la Salud: menor a 10 ug/L (10 ppb).

2- concentración de arsénico superior a dicho límite.

En el segundo enfoque iremos más lejos. Nos interesará anticipar un valor de la concentración de arsénico, en base a lo aprendido por Atlantis utilizando la base de datos. Este es el modelo que en realidad utiliza Atlantis en este momento. Lo desarrollaremos en clases venideras.

Nos apoyaremos mucho en lo aprendido en los módulos previos y haremos una breve revisión de términos e introduciremos algunos nuevos.

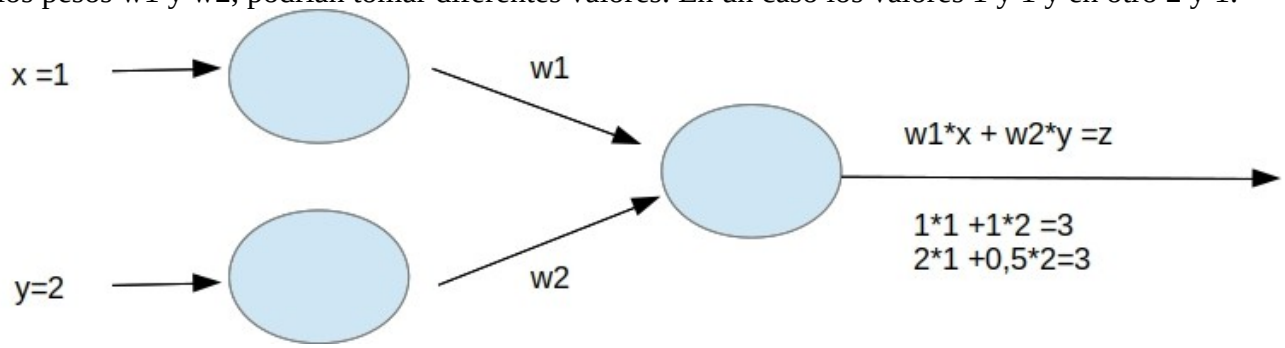
Un **modelo** es una representación matemática del mundo real en que una variable dependiente (también conocida como salida) se relaciona con otras variables (entradas, variables independientes o predictores). El **entrenamiento** del modelo es la confrontación del modelo con datos para que se ajuste a los mismos. Llamamos **set de entrenamiento** a estos datos, que contienen valores de las variables independientes y dependientes. Las **variables independientes o predictores** son aquellas que medimos y que nos servirán para **clasificar** a nuestras unidades experimentales en base al valor de la **variable dependiente o salida del modelo.** Otro grupo de datos con valores de variables predictivas y salidas, será utilizado para validar lo aprendido por el modelo, grupo de datos a los que llamamos **set de testeo o set de prueba**. Con estos datos utilizando los predictores, el modelo predice el valor de la salida y luego chequearemos si esa salida del modelo coincide con el valor de la variable dependiente del set de testeo. Luego tenemos el **set de predicción** en que conocemos las variables predictoras pero no la variable de salida o dependiente.

Cuando aplicamos cualquier mecanismo de aprendizaje automatizado al igual que cuando hacemos una regresión lineal, modelo lineal o regresión logística, por citar algunos modelos, tenemos un set de datos en que conocemos la o las variables respuesta y las variables predictoras a los que aplicamos el modelo y éste asigna valores a sus parámetros. Luego con esos parámetros aplicamos el modelo a otro set de datos en que conocemos valores de las variables explicativas y a través del modelo deseamos conocer los valores de la variable respuesta.
Un ejemplo concreto y sencillo es cuando realizamos una curva de calibración por ejemplo en una medición espectrofotométrica. Tenemos un grupo de soluciones en que conocemos la concentración de una sustancia (salida) y medimos una propiedad llamada absorbancia (predictor). Estos son los datos de entrenamiento, se los damos a una regresión lineal para que calcule los parámetros de la ecuación de la recta o curva de calibración. Tenemos luego una o varias soluciones a las que llamamos controles de calidad o quality control, en las que conocemos la concentración y medimos su concentración, pero no forman parte de la curva de calibración, esto son los datos de testeo. Alimentamos el modelo con la absorbancia medida y si el modelo aprendió, nos tiene que dar una concentración, cercana al valor nominal conocido de la solución control de calidad, por supuesto con un error preestablecido y aceptado. Luego con esta ecuación de la recta y otro set de datos en que solo conocemos la absorbancia, calculamos la concentración de la sustancia en cuestión. Este último set de datos es el de predicción.
No debemos caer en el error de pensar que un modelo es mejor que otro. Hay un mejor modelo para cada situación a resolver. Además nuestro mejor modelo será el que podamos conseguir con nuestros conocimientos y limitaciones actuales. No será raro que luego de pasado un tiempo volvemos sobre nuestros datos y nos demos cuenta que podemos modelizar aun mejor el fenómeno en estudio. El mejor modelo será el que mejor responda a nuestro set de testeo. Para ello tenemos varios indicadores.
1- Si se trata de un modelo de regresión, la suma de cuadrados de los desvíos y/o el R-squared son muy buenos predictores. Una suma de cuadrados menor o un R-squared más cercano a 1, nos indica un mejor modelo. Los modelos de regresión han sido ampliamente utilizados, especialmente en el módulo 4 de este curso.
2- En los modelos de clasificación, la exactitud (accuracy) es un buen predictor. La exactitud la definimos como el número de datos en que la nuestro modelo predice el mismo valor que el valor medido, dividido el número total de casos. También es útil el área bajo la curva ROC.
Comenzaremos el estudio de aprendizaje automatiza con redes neuronales.

Redes neuronales artificiales

Las redes neuronales son un mecanismo de cálculo enrolado dentro de lo que se conoce como aprendizaje automatizado o inteligencia artifical. Si bien el nombre puede amedrentar a los usuarios no es otra cosa que un mecanismo para hallar un modelo que permita vincular variables de un set de datos y hacer predicciones. Por supuesto que existen cursos muy completos y avanzados sobre el tema. En este módulo se darán principios básicos y posibles aplicaciones a las ciencias biomédicas.
A modo de ejemplo utilizaremos las redes neuronales para obtener información sobre la concentración de arsénico en muestras de agua de consumo humano. Haremos en primer lugar un enfoque dicotómico, reemplazando los valores medidos de arsénico por 0 y 1, correspondiendo el valor 0 a valores menores o iguales a 10 ppb y 1 a los valores mayores a 10 ppb. El límite de 10 ppb ha sido fijado siguiendo criterios de la Organización Mundial de la Salud en lo que respecta a calidad de agua. Se considera apta para el consumo si el arsénico tiene una concentración menor o igual a 10 ppb. Intentaremos utilizar las redes neuronales para predecir si una muestra de agua supera o no dicho límite. En clases siguientes haremos un enfoque de regresión, utilizando los valores medidos de arsénico.

En síntesis con un set de datos (datos entrenamiento) hacemos que nuestro modelo aprenda (se ajuste a los datos). Una vez que el modelo aprendió, le ofrecemos datos incompletos y el modelo predice valores faltantes.
Las redes neuronales no discrepan conceptualmente mucho de una regresión lineal, salvo en que se utiliza un sistema que intenta imitar las neuronas del sistema nervioso y sus conexiones y no presupone un modelo matemático que las relaciona ni una distribución de probabilidad de las variables. Los datos ingresan a módulos a los que llamamos neuronas (que son las variables explicativas) y estas generan valores que se conectan con otras neuronas que generan señales y se dirigen a las neuronas de salida que sería nuestra variable respuesta. Las señales de entrada y salida son afectadas por funciones que le dan peso y umbral y de esta manera la red va ajustando los valores de los pesos. Las redes neuronales a su vez llevan capas de neuronas que llamamos capas ocultas formadas por un número de neuronas que se fija por el usuario. Estas neuronas de la capa oculta reciben señales de la neurona de entrada, la procesan y envían la señal a las neuronas de salida. Podría asimilarse esto al sistema nervioso: una neurona del cerebro genera una señal, la envía por una vía eferente hacia la médula espinal donde hará sinapsis con otra neurona que enviará una señal a un músculo, que sería para nosotros la neurona de salida.
Veamos un ejemplo ultrasencillo mínimo. Este ejemplo solo tiene como objetivo comprender el funcionamiento de una red. Supongamos una red que tiene dos neuronas de ingreso, que reciben la información de una variable x (que toma el valor 1) y una variable y (que toma el valor 2). La variable respuesta la llamamos z y sabemos que toma el valor 3. Entonces la red ajustará los pesos de estas variables de manera que se ajusten al valor de la salida. Como podemos ver en la imagen, los pesos w1 y w2, podrían tomar diferentes valores. En un caso los valores 1 y 1 y en otro 2 y 1.



Por supuesto es un enfoque sencillo, pero podríamos pensar que así funcionan. Busca valores de los pesos de manera que las entradas x e y se ajusten a la salida z.
En su concepto básico, una red neuronal no discrepa de un mecanismo de regresión lineal entre dos variables o una regresión logística entre varias variables predictoras y una variable respuestas dicotómica o continua.
Para el desarrollo de este tema utilizaremos una base de datos del CUEM, que contiene múltiples mediciones de componentes químicos del agua de consumo humano y veremos una aplicación sencilla de las redes neuronales pero extremadamente efectiva.

Preparación de los datos

La preparación de los datos para la utilización en aprendizaje automatizado es muy importante. No agotaremos en las páginas siguientes los procedimientos que se deben aplicar. El tratamiento que realizaremos no es nada novedoso y se muestra solo a fines de revisión de conceptos.
Hallará un archivo con los datos, con el nombre DatosR81.RData, que corresponde a un data.frame que será de utilidad. Copie este archivo al directorio donde funcionará su espacio de trabajo.
Luego desde su espacio de trabajo cargue los datos con el siguiente código

```
> load("datosR81.RData")
```

compruebe que los datos fueron introducidos. Debe hallar en su espacio de trabajo un data.frame llamado datosaguas

```
> ls()

[1] "datosaguas"
```

Esta base de datos tiene una gran cantidad de registros (cada muestra de agua que si identifica con un código alfanumérico, ej A12) y de campos en que se almacen datos y valores de las mediciones. Verificamos primero el número de registros, que corresponde a uno por línea de la base de datos

```
> nrow(datosaguas)

[1] 425
```

y el nombre de los campos

```
> names(datosaguas)

 [1] "codigo"              "fecha"               "localidad"
 [4] "provincia"           "tipoagua"            "responsble"
 [7] "ph"                  "cvph"                "conductividad"
[10] "cvconductividad"     "cloruro"             "cvcloruro"
[13] "carbonato"           "cvcarb"              "bicarbonato"
[16] "cvbicar"             "solidostotales"      "cvsolidostotales"
[19] "fosfato"             "cvfosfato"           "nitrato"
[22] "cvnitrato"           "nitrito"             "cvnitrito"
[25] "fluoruro"            "cvfluoruro"          "arsenico"
[28] "cvarsenico"          "amonio"              "cvamonio"
[31] "observaciones"       "alcalinidadtotal"    "CValcalinidadtotal"
[34] "consumo"             "sulfato"             "cvsulfato"
[37] "informado"           "sodio"               "cvsodio"
[40] "dqo"                 "cvdqo"               "tkn"
[43] "cvtkn"               "calcio"              "cvcalcio"
[46] "durezatotal"         "cvdurezatotal"       "duda"
[49] "numeropersonas"      "orp"                 "cvorp"
[52] "magnesio"            "cvmagnesio"          "servicio"
[55] "materiaorganica"     "litio"               "cvlitio"
[58] "arsenicoIII"         "cvarsenicoIII"       "iodo"
[61] "cviodo"              "especiescloro"       "cvespeciescloro"
[64] "muestraagotada"      "manganeso"           "cvmanganeso"
[67] "estroncio"           "cvestroncio"         "plomo"
[70] "cvplomo"             "turbidez"            "cvturbidez"
```

En principio haremos un enfoque simplificado del tema, de manera de comprender el funcionamiento de las redes neuronales. Los campos que nos interesarán son arsenico, fluoruro, alcalinidadtotal y tipoagua, dado que hay evidencia que el arsénico guarda una buena relación con fluoruro y alcalinidadtotal. Esta información surgió de aplicar diversos mecanismos de análisis que involucraron modelos lineales multivariados y regresiones logísticas, entre otras.
Utilizaremos el paquete neuralnet, que puede descargar de https://cran.r-project.org/
Luego cargue la biblioteca en su espacio de trabajo

```
> library(neuralnet)
```

Nos quedamos con los datos de arsenico, fluoruro, tipoagua y alcalinidadtotal, que asignamos a un nuevo dataframe, al que llamamos datosnn1

```
> datosnn1<-datosaguas[,c(25,32,5,27)]
```

verificamos campos

```
> names(datosnn1)
```

```
[1] "fluoruro"        "alcalinidadtotal" "tipoagua"        "arsenico"
```

y valores

```
> summary(datosnn1)
```

```
    fluoruro          alcalinidadtotal       tipoagua             arsenico
 Min.   :0.04688   Min.   :  20.0     red             :216   Min.   :  0.000
 1st Qu.:0.21615   1st Qu.: 108.2     pozo            :151   1st Qu.:  2.905
 Median :0.54744   Median : 307.1     envasada        : 17   Median : 11.752
 Mean   :0.79580   Mean   : 328.2     osmosisinversa: 14     Mean   : 19.927
 3rd Qu.:1.11908   3rd Qu.: 467.7     filtro          : 13   3rd Qu.: 27.296
 Max.   :5.44843   Max.   :1828.7     otras           :  9   Max.   :220.741
 NA's   :1                            (Other)         :  5   NA's   :4
```

Además trabajaremos con una red donde los valores de arsénico serán dicotómicos. La concentración de arsénico tomará el valor 0 o 1 dependiendo que la medición sea menor o mayor a 10 ppb. Se eligió este valor dado que 10 ppb es el límite para la concentración de arsénico recomendado por la Organización Mundial de la Salud y por muchas regulaciones nacionales y provinciales de la República Argentina. Utilizaremos la funcion ifelse() que es muy práctica para reemplazar valores cuantitativos por una clasificación dicotómica

```
> datosnn1$arsenico<-ifelse(datosnn1$arsenico>10,1,0)
```

chequeamos que así se haya realizado

```
> summary(datosnn1)
```

```
    fluoruro          alcalinidadtotal       tipoagua             arsenico
 Min.   :0.04688   Min.   :  20.0     red             :216   Min.   :0.0000
 1st Qu.:0.21615   1st Qu.: 108.2     pozo            :151   1st Qu.:0.0000
 Median :0.54744   Median : 307.1     envasada        : 17   Median :1.0000
 Mean   :0.79580   Mean   : 328.2     osmosisinversa: 14     Mean   :0.5321
 3rd Qu.:1.11908   3rd Qu.: 467.7     filtro          : 13   3rd Qu.:1.0000
 Max.   :5.44843   Max.   :1828.7     otras           :  9   Max.   :1.0000
 NA's   :1                            (Other)         :  5   NA's   :4
```

vemos el número de datos de la base

```
> nrow(datosnn1)

[1] 425
```

Depuraremos un poco la base datos para evitarnos problemas. Eliminaremos filas con datos NA en fluoruro, arsenico, alcalinidadtotal o tipo de agua, reescribiendo el resultado sobre el data.frame datosnn1. Escribimos la función subset() utilizando la función is.na(), la que utilizamos evaluando su igualdad a FALSE. La función subset seleccionará todas las líneas que is.na() sea FALSE, es decir seleccionará todas las líneas de datosnn1 que tengan valor en fluoruro, arsenico y alcalinidad total

```
> datosnn1<-subset(datosnn1,is.na(datosnn1$fluoruro)==FALSE & is.na(datosnn1$arsenico)==FALSE & is.na(datosnn1$alcalinidadtotal)==FALSE & is.na(datosnn1$tipoagua)==FALSE)
```

Comprobemos el número de filas que quedan en la base de datos

```
> nrow(datosnn1)
```

```
[1] 421
```

Por último haremos un ajuste del campo tipoagua. Esta base contiene análisis de aguas muy variadas, nos van a interesar en el estudio algunos niveles del factor. Veamos la cantidad de datos según cada tipo de agua

```
> table(datosnn1$tipoagua)
```

```
                pozo          red osmosisinversa       envasada
           0     150          214           14           16
  superficial       filtro        otras
           5           13            9
```

Tenemos algunos valores que eliminaremos.
0 que no tienen categoría
5 son aguas superficiales
9 son otro tipo de aguas
Eliminaremos estos tres grupos quedándonos con los niveles: pozo, red, osmosisinversa, envasada y filtro

```
> datosnn1<-subset(datosnn1,datosnn1$tipoagua=="pozo" | datosnn1$tipoagua=="red" | datosnn1$tipoagua=="osmosisinversa" | datosnn1$tipoagua=="envasada" | datosnn1$tipoagua=="filtro")
```

comprobamos si realmente eliminamos los 14 registros correspondientes a superficial y otra

```
> nrow(datosnn1)
```

```
[1] 407
```

verificamos los niveles del campo tipo agua

```
> table(datosnn1$tipoagua)
```

```
                pozo          red osmosisinversa       envasada
           0     150          214           14           16
  superficial       filtro        otras
           0           13            0
```

vemos que aun nos falta eliminar un nivel sin nombre y con 0 registro, que seguramente nos molestará en el futuro análisis.Para eliminarlos utilizamos la funcion droplevels()

```
> datosnn1<-droplevels(datosnn1)
```

y verificamos el proceso

```
> table(datosnn1$tipoagua)
```

```
        pozo          red osmosisinversa       envasada       filtro
         150          214           14           16           13
```

ahora sí. Asignaremos valores numéricos a los diferentes tipos de agua.
pozo=1
red=2
osmosisinversa=3
envasada=4
filtro=5
Utilizaremos la función gsub() aprendida en los primeros módulos

```
> datosnn1$tipoagua<-gsub("pozo",1,datosnn1$tipoagua)
```

```
> datosnn1$tipoagua<-gsub("red",2,datosnn1$tipoagua)

> datosnn1$tipoagua<-gsub("osmosisinversa",3,datosnn1$tipoagua)

> datosnn1$tipoagua<-gsub("envasada",4,datosnn1$tipoagua)

> datosnn1$tipoagua<-gsub("filtro",5,datosnn1$tipoagua)
```

verificamos los reemplazos

```
> table(datosnn1$tipoagua)

  1   2   3   4   5
150 214  14  16  13
```

comprobamos que son los mismos valores, salvo que cambiaron los nombres
Transformamos los valores que son niveles de un factor en numérico

```
> datosnn1$tipoagua<-as.numeric(datosnn1$tipoagua)
```

La base utilizada tiene 407 filas o registros. Utilizaremos 250 registros para entrenar la red neuronal, donde las neuronas de entrada serán las variables fluoruro, tipoagua y alcalinidad total y la neurona de salida será arsenico. Con los datos restantes construimos un data.frame de prueba con el que verificaremos si nuestra red es capaz de predecir los valores de arsenico en base a los valores de fluoruro, tipoagua y alcalinidadtotal, que se pueden conocer en minutos de que una muestra ingresó al laboratorio.
En este caso hicimos una división en entrenamiento y prueba, simplemente tomando los primeros 250 registros para entrenamiento y el resto para prueba. Podríamos decir que en este caso es aplicable, ya que cada muestra ingresa de manera aleatoria, sin seguir ningún patrón de lugar, tipo de agua, contenido de componentes, etc. Pero veremos más adelante que es conveniente hacer esta selección con un mecanismo aleatorio.
Creamos entonces los grupos entrenamientonn1 y pruebann1

```
> entrenamientonn1<-datosnn1[c(1:250),]

> pruebann1<-datosnn1[c(251:407),]

> summary(entrenamientonn1)
```

```
    fluoruro        alcalinidadtotal      tipoagua        arsenico
 Min.   :0.06477   Min.   :  20.0    Min.   :1.000   Min.   :0.000
 1st Qu.:0.24854   1st Qu.: 148.4    1st Qu.:2.000   1st Qu.:0.000
 Median :0.6285 0  Median : 317.4    Median :2.000   Median :1.000
 Mean   :0.86253   Mean   : 341.3    Mean   :1.992   Mean   :0.648
 3rd Qu.:1.40136   3rd Qu.: 465.3    3rd Qu.:2.000   3rd Qu.:1.000
 Max.   :4.35136   Max.   :1828.7    Max.   :5.000   Max.   :1.000
```

```
> summary(pruebann1)
```

```
    fluoruro        alcalinidadtotal      tipoagua        arsenico
 Min.   :0.04688   Min.   :  23.42    Min.   :1.000   Min.   :0.0000
 1st Qu.:0.21604   1st Qu.:  85.09    1st Qu.:1.000   1st Qu.:0.0000
 Median :0.43210   Median : 286.28    Median :1.000   Median :0.0000
 Mean   :0.74115   Mean   : 324.13    Mean   :1.599   Mean   :0.3822
 3rd Qu.:0.85566   3rd Qu.: 511.03    3rd Qu.:2.000   3rd Qu.:1.0000
 Max.   :5.44843   Max.   :1434.60    Max.   :5.000   Max.   :1.0000
```

Los problemas de aprendizaje automatizado así como los de optimización utilizan en sus procesos números aleatorios. Estos números no son completamente aleatorios pero cambian cada vez que se realiza una generación de ellos y esto trae a nuestro modelo problemas de reproducibilidad. Por ello es necesario crear una "semilla" de manera que cada vez que corramos nuestro modelo los números aleatorios sean los mismos. Para crear una semilla se utiliza la funcion set.seed()
En este caso utilizaremos el valor 300, que bien podría ser otro.

```
> set.seed(300)
```

Entrenamiento de la red neuronal

Construimos una red neuronal en la que relacionamos el valor de arsénico con fluoruro, tipoagua y alcalinidadtotal, utilizando los datos entrenamientonn1. Crearemos la red con una capa oculta de 10 neuronas. Dado que arsénico tiene datos dicotómicos utilizamos la función de activación (act.fct) logistic y los datos de la red los adjudicamos al objeto nn1.
Fijamos algunos argumentos
stepmax=10000. Limitará los pasos de cálculo si superan a 10000 indicará que no ha obtenido un resultado
rep=1. Realiza el entrenamiento 1 sola vez
threshold=0.001. Es un criterio de finalización utilizado en el proceso de derivación.
hidden=10. Si bien no hay reglas muy claras al respecto, se suele recomendar 2/3 de las neuronas de la capa oculta respecto de las entradas. En nuestro caso tenemos tres entradas (fluoruro, alcalinidad y tipo de agua), la capa oculta debería tener 2 neuronas si sigue esa regla. Pero la prueba y error suelen ser buenos aliados. En nuestro caso 10 neuronas han sido satisfactorias.
Sembramos la semilla que nos dará previsibilidad al proceso aleatorio

```
> set.seed(300)
```

aplicamos la funcion neuralnet()

```
> nn1<-neuralnet(arsenico~fluoruro+alcalinidadtotal+tipoagua,data=entrenamientonn1,stepmax=100000, rep=1, threshold=0.001, hidden= 10,act.fct="logistic",linear.output=FALSE)
```

Puede ocurrir que no converja con rep=1, es decir con un solo ciclo de entrenamiento. R le mostrará
Warning message:
Algorithm did not converge in 1 of 1 repetition(s) within the stepmax.
De ser así, puede modificar el número de neuronas de la capa oculta, threshold o stepmax. Iremos viendo soluciones.
Veamos ahora las salida del análisis. La salida se almacena en el objeto nn1 que guarda abundante información. Veremos la más sencilla y significativa. Si ejecuta

```
>nn1
```

tendrá excesiva información, toda junta. Iremos viendo por partes.
Podemos ver los datos de arsénico del set de entrenamiento con

```
>nn1$response
```

```
   arsenico
2         1
3         1
4         1
5         1
6         0
.......
```

257 0
258 0
259 0

Con la función head podemos ver los primeros valores de arsenico de la base de entrenamiento y comprobamos que coincide con nn1$response

```
> head(entrenamientonn1)
```

```
  fluoruro alcalinidadtotal tipoagua arsenico
2   0.7251        531.06500        1        1
3   0.9681        599.27280        2        1
4   0.3074        756.82045        1        1
5   1.2165        327.82072        2        1
6   0.5218         65.30059        2        0
```

```
>nn1$covariate
```

nos muestra los datos de la variables de entrada del grupo de entrenamiento, en este caso fluoruro y alcalinidad y tipo agua, coincidiendo con lo que vemos con head().

```
      fluoruro alcalinidadtotal tipoagua
2   0.72510000        531.06500        1
3   0.96810000        599.27280        2
4   0.30740000        756.82045        1
5   1.21650000        327.82072        2
6   0.52180000         65.30059        2
.......
256 0.99912675        461.24619        2
257 0.38406222        459.30251        2
258 0.30861030        222.61039        4
259 1.09986802        756.32793        1
```

```
>nn1$model.list$variables
```

nos muestra los nombres de las variables de entrada: fluoruro y alcalinidadtotal.

```
[1] "fluoruro"         "alcalinidadtotal" "tipoagua"
```

```
>nn1$model.list$response
```

nos muestra el nombre de la variable respuesta: en este caso arsenico

```
[1] "arsenico"
```

```
>nn1$data
```

nos muestra los datos de entrenamiento. Básicamente nos muestra junto nn1$response, nn1$covariate.

```
      fluoruro alcalinidadtotal tipoagua arsenico
2   0.72510000        531.06500        1        1
3   0.96810000        599.27280        2        1
4   0.30740000        756.82045        1        1
5   1.21650000        327.82072        2        1
6   0.52180000         65.30059        2        0
.........
256 0.99912675        461.24619        2        1
257 0.38406222        459.30251        2        0
```

258 0.30861030 222.61039 4 0
259 1.09986802 756.32793 1 0

```
>nn1$net.result
```

nos muestra los valores de arsenico estimados por la red

```
>  nn1$net.result
[[1]]
        [,1]
2   0.9240977
3   0.9278831
4   0.9278984
5   0.9130921
6   0.1876223
7   0.9297218
8   0.2203532
9   0.9303613
10  0.9299667
.....
254 0.9305444
255 0.9257718
256 0.9212774
257 0.8593067
258 0.1704281
259 0.9303140
```

pronto volveremos sobre estos datos

```
>nn1$result.matrix
```

nos muestra algunos datos interesantes como el error, el umbral alcanzado y los pasos

```
                                  [,1]
error                      1.336855e+01
reached.threshold          7.342463e-04
steps                      2.657000e+03
......
```

Podemos graficar la red obtenida donde vemos algunos de los datos ya mostrados como el error y los pasos.

```
> plot(nn1)
```

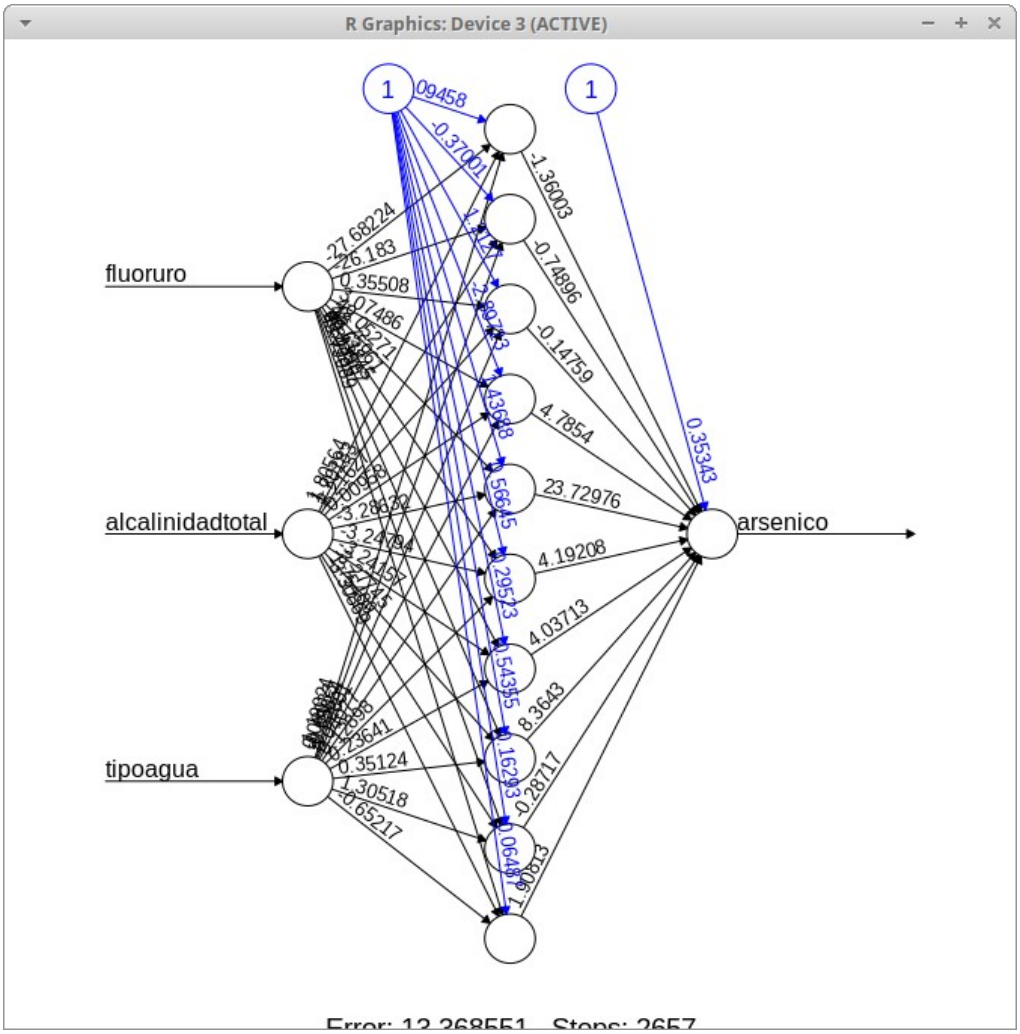


Figura 2

Haremos una primera interpretación de los datos obtenidos y para ello observaremos dos de las salidas mostradas anteriormente

nn1$response: que son nuestros valores de arsénico medidos, tratados dicotómicamente.

nn1$net.result: son los valores que la red asignó a cada registro.

Podemos ver que para los valores medidos 1, la red arrojó resultados muy cercanos a 1 y en aquellos casos que era 0, asigno un valor bajo. En el número 8 vemos que no hay coincidencia

```
> nn1$net.result

[[1]]
        [,1]
2   0.9240977
3   0.9278831
4   0.9278984
5   0.9130921
6   0.1876223
7   0.9297218
8   0.2203532
9   0.9303613
10  0.9299667
.....
```

```
> nn1$response
```

```
   arsenico
2         1
3         1
4         1
5         1
6         0
7         1
8         1
9         1
```

Obviamente, la red neuronal hace maravillas con la predicción, pero no milagros!!!
Prueba del entrenamiento

Ahora aplicaremos la red ya entrenada al set de datos de prueba: pruebann1. Para esto utilizamos la función predict()

```
> nn1prueba=predict(nn1,pruebann1)
```

vemos las primeras 20 líneas

```
> head(nn1prueba,20)
```

```
          [,1]
260 0.9286885
261 0.9292261
262 0.8840157
263 0.9178111
264 0.9304361
265 0.9305439
266 0.8271038
267 0.9164212
268 0.9230270
269 0.6137022
270 0.1447420
271 0.7100729
272 0.9305444
273 0.1499424
274 0.2105726
275 0.1231503
276 0.8605450
277 0.4067817
278 0.9305431
279 0.8446495
```

Como podemos ver la prueba tiene valores entre 0 y 1 algunos muy cercanos a uno u otro extremo. Creamos un objeto resultadopruebann1, con valores 0 y 1, con el siguiente criterio si es mayor a 0.5 le asignamos 1 y en caso contrario 0

```
> resultadopruebann1<-ifelse(nn1prueba>0.5,1,0)

> head(resultadopruebann1,20)
```

```
    [,1]
260    1
261    1
```

262 1
263 1
264 1
265 1
266 1
267 1
268 1
269 1
270 0
271 1
272 1
273 0
274 0
275 0
276 1
277 0
278 1
279 1
.....

Así vemos que nuestra red neuronal predice que las 10 primeras muestras superarán 10 ppb, mientras que la siguiente no lo hará y así sucesivamente.

agregamos a pruebann1, los valores hallados que tenemos en resultadopruebann1

Recordamos que pruebann1, son los datos seleccionados como datos de testeo de funcionamiento

```
> pruebann1<-cbind(pruebann1,prediccion=resultadopruebann1)

> summary(pruebann1)
```

fluoruro	alcalinidadtotal	tipoagua	arsenico	prediccion
Min. :0.04688	Min. : 23.42	Min. :1.000	Min. :0.0000	Min. :0.0000
1st Qu.:0.21604	1st Qu.: 85.09	1st Qu.:1.000	1st Qu.:0.0000	1st Qu.:0.0000
Median :0.43210	Median : 286.28	Median :1.000	Median :0.0000	Median :1.0000
Mean :0.74115	Mean : 324.13	Mean :1.599	Mean :0.3822	Mean :0.5796
3rd Qu.:0.85566	3rd Qu.: 511.03	3rd Qu.:2.000	3rd Qu.:1.0000	3rd Qu.:1.0000
Max. :5.44843	Max. :1434.60	Max. :5.000	Max. :1.0000	Max. :1.0000

y realizamos una tabla con los valores de arsenico medidos (cuarta columna) y calculados por la red neuronal (quinta columna), a esta tabla se la llama **matriz de confusión**. Esta matriz la hacemos con una conocida función de R, table()

```
> matrizconfusion<- table(pruebann1$arsenico,pruebann1$prediccion)

> matrizconfusion
```

	0	1
0	56	41
1	10	50

pasamos a una tabla más organizada.

		medido	
		1	0
estimado	1	vp=50	fp=10

	0	fn=41	vn=56

Veamos una interpretación para seguir adelante. En la tabla vemos el valor 50, que se halla en la intersección de medido=1 y estimado =1. Es decir que nuestro modelo halló 50 muestras en que el valor medido coincidió con el valor estimado, en que el valor de arsénico supero 10 ppm. Llamamos a este valor true positive = verdadero positivo = vp. El valor 10 se halla en la intersección de medido=0 y estimado=1. Es decir el modelo estimó que el arsénico supera 10 ppb, cuando no es así. Estos datos son los falsos positivos = fp. El valor 41, tiene arsenico mayor a 10 ppm, pero nuestro modelo dice que no es así. Son estos valores los falsos negativos: fn. Por último hay 56 muestras en que el valor de arsénico no supera 10 ppb y el modelo predice lo mismo, son los verdaderos negativos: vn.
hacemos una prueba chisq.test() para evaluar si hay asociación entre los valores estimados y medidos.

```
> chisq.test(table(pruebann1$prediccion,pruebann1$arsenico))
```

Pearson's Chi-squared test with Yates' continuity correction

data: table(prediccionnn1$prediccion, prediccionnn1$arsenico)
X-squared = 23.998, df = 1, p-value = 9.642e-07

obtenemos un p-value<<0.05, lo que nos indica buena asociación entre las variables involucradas. Lo que nos está indicando que la red neuronal predice los valores de arsénico de una muestra de agua. Podemos calcular sensibilidad, especificidad y exactitud de nuestra estimación en base a los datos de verdaderos positivos y negativos, falsos positivos y negativos.
Sensibilidad= vp / (vp+fn) = 50/(50+41) = 0.55
Especificidad= vn / (vn + fp) = 56 / (56+10)= 0.85
exactitud= (vp + vn)/(vp+vn+fp+fn) = (50+56)/(50+56+10+41)=0.68
Puede revisar conceptos sobre estos conceptos en la clase referida a curvas ROC desarrollado en el módulo 4.
El valor de exactitud no es nada mal para comenzar. Se podrán variar argumentos en búsqueda de mejores resultados. También se puede incluir otras covariables, cambiar algunas de ellas o modificar aspectos de la red.
Podemos calcular la exactitud con

```
> >
exactitud<-(matrizconfusion[1,1]+matrizconfusion[2,2])/(matrizconfusion[1,1]+matrizconfusion[1,2]+matrizconfusion[2,1]+matrizconfusion[2,2])

> exactitud
```

[1] 0.6751592

Resumiendo todos los pasos

1- Preparamos el set de entrenamiento y de prueba

```
> entrenamientonn1<-datosnn1[c(1:250),]

> pruebann1<-datosnn1[c(251:407),]
```

2- Cargamos la biblioteca necesaria

```
> library(neuralnet)
```

3- sembramos la semilla

```
> set.seed(300)
```

4- Ajustamos el modelo con el set de entrenamiento

```
> nn1<-neuralnet(arsenico~fluoruro+alcalinidadtotal+tipoagua,data=entrenamientonn1,stepmax=100000, rep=1, threshold=0.001, hidden= 10,act.fct="logistic",linear.output=FALSE)
```

5- hacemos la predicción con el set de datos de prueba

```
> nn1prueba=predict(nn1,pruebann1)
```

6- transformamos los valores a 0 y 1.

```
> resultadopruebann1<-ifelse(nn1prueba>0.5,1,0)
```

7- juntamos datos en un solo data.frame

```
> pruebann1<-cbind(pruebann1,prediccion=resultadopruebann1)
```

8- construimos la matriz de confusion

```
> matrizconfusion<- table(pruebann1$arsenico,pruebann1$prediccion)

> matrizconfusion
```

9- calculamos exactitud

```
> exactitud<-(matrizconfusion[1,1]+matrizconfusion[2,2])/(matrizconfusion[1,1]+matrizconfusion[1,2]+matrizconfusion[2,1]+matrizconfusion[2,2])

> exactitud
```

[1] 0.6751592

10- pedimos los datos del ajuste de red neuronal a los datos de entrenamiento, pero solo las tres primeras filas

```
> nn1$result.matrix[c(1:3),]
```

error	reached.threshold	steps
28.449965007	0.000874281	68.000000000

11- pedimos el gráfico del modelo

```
> plot(nn1)
```

Podemos eliminar símbolos y explicaciones de los pasos anteriores, de manera que podamos ejecutar todos los pasos, con solo copiarlos y pegarlos en la consola

```
# copie al portapapeles las líneas siguientes y péguelas en la consola. Luego ejecute enter.
entrenamientonn1<-datosnn1[c(1:250),]

pruebann1<-datosnn1[c(251:407),]

set.seed(300)

nn1<-neuralnet(arsenico~fluoruro+alcalinidadtotal+tipoagua,data=entrenamientonn1,stepmax=100000, rep=1, threshold=0.001, hidden= 10,act.fct="logistic",linear.output=FALSE)

nn1prueba=predict(nn1,pruebann1)
```

```
resultadopruebann1<-ifelse(nn1prueba>0.5,1,0)

pruebann1<-cbind(pruebann1,prediccion=resultadopruebann1)

matrizconfusion<- table(pruebann1$arsenico,pruebann1$prediccion)

matrizconfusion

exactitud<-(matrizconfusion[1,1]+matrizconfusion[2,2])/
(matrizconfusion[1,1]+matrizconfusion[1,2]+matrizconfusion[2,1]+matrizconfusion[2,2])

exactitud

nn1$result.matrix[c(1:3),]

plot(nn1)
```

Utilizaremos las líneas anteriores en la próxima clase para realizar algunas pruebas y comprobaciones.

Predicción en datos solo con predictores

Por último hagamos una predicción de la concentración de arsénico en un grupo de datos en que no se ha realizado aun la medición de arsénico.

Para ello introduzca los datos de la tablaR811 que hallará en la planilla de cálculo tablaR8-1.ods/xls

```
> prediccionarsenico<-read.table("clipboard",header=TRUE,dec=",",sep="\t")

> prediccionarsenico
```

	fluoruro	alcalinidadtotal	tipoagua
1	0.25	150	2
2	1.53	450	1
3	0.15	130	3

tenemos tres muestras, con los valores de fluoruro y alcalinidad, correspondientes a aguas de: red, pozo y ósmosisinversa.

Aplicamos predict() a estos valores para estimar el arsénico

```
> prediccionvalores=predict(nn1,prediccionarsenico)
```

vemos el resultado

```
> prediccionvalores
```

```
        [,1]
[1,] 0.1853688
[2,] 0.9293850
[3,] 0.1290517
```

que nos indican que las aguas 1 y 3, con valores cercanos a 0, tiene arsénico menor a 10 ppb y la 2 cuyo valor es cercano a 1, tendría arsénico mayor a 10 ppm.

Módulo 8. Clase 2

Continuación redes neuronales para clasificación

En la clase anterior construimos una red neuronal para predecir si la concentración de arsénico en agua es superior o no al límite de 10 ppb impuesto por la OMS. Veremos a continuación la modificación de algunos parámetros y su efecto sobre la exactitud de su funcionamiento.

Resumiendo todos los pasos

Luego de haber preparado los datos, realizamos una serie de pasos hasta hallar la matriz de confusión y la exactitud de funcionamiento. Estos pasos quedaron resumidos en los siguientes comandos:

```
entrenamientonn1<-datosnn1[c(1:250),]

pruebann1<-datosnn1[c(251:407),]

set.seed(300)

nn1<-
neuralnet(arsenico~fluoruro+alcalinidadtotal+tipoagua,data=entrenamientonn1,stepmax=100000,
rep=1, threshold=0.001, hidden= 10,act.fct="logistic",linear.output=FALSE)

nn1prueba=predict(nn1,pruebann1)

resultadopruebann1<-ifelse(nn1prueba>0.5,1,0)

pruebann1<-cbind(pruebann1,prediccion=resultadopruebann1)

matrizconfusion<- table(pruebann1$arsenico,pruebann1$prediccion)

matrizconfusion

exactitud<-(matrizconfusion[1,1]+matrizconfusion[2,2])/
(matrizconfusion[1,1]+matrizconfusion[1,2]+matrizconfusion[2,1]+matrizconfusion[2,2])

exactitud

nn1$result.matrix[c(1:3),]

plot(nn1)
```

En esta clase trabajaremos con los mismos datos de la clase R81. Por lo tanto podemos realizar diferentes acciones para trabajar con ellos

1- Utilizar el mismo espacio de trabajo. Cosa que no haremos por cuestiones de orden, pero sería totalmente válido.
2- Grabar los data frame utilizados con la función

```
> Save()
```

Copiar los archivos al nuevo espacio de trabajo y luego con

```
> Load()
```

introducirlos en el nuevo espacio, pero esto deberíamos hacerlo para cada data.frame.
3- Un recurso muy efectivo es desde un administrador de archivos, copiar el .RData al nuevo directorio: R82 y luego cuando ingresemos a nuestro espacio de trabajo tendremos todos los objetos. Es un "efectivo" mecanismo, pero poco "elegante" para un usuario avanzado de R.
4- **Lo que haremos es algo más elegante:** abriremos un nuevo espacio de trabajo, luego con la función setwd() nos moveremos al espacio de la clase R81, con lo que accederemos a todos los objetos de ese espacio.
Vemos el camino de nuestro nuevo espacio de trabajo con la funcion getwd(). Supongamos que esta es nuestra situación, para el espacio de trabajo de la clase R82

```
> getwd()
```

[1] "/media/alfredo/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R8/R82"
Entonces cambiamos de espacio de trabajo a la clase R81

```
> setwd("/media/alfredo/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R8/R81")
```

como podemos ver aun no tenemos ningún objeto en el espacio de trabajo

```
> ls()
```

character(0)
cargamos el .RData del espacio R81

```
> load(".RData")
```

y luego con setwd() volveremos al nuevo espacio que llamaremos R82.

```
> setwd("/media/alfredo/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R8/R82")
```

y podemos comprobar con

```
> ls()
```

que los objetos están en nuestro espacio

```
 [1] "aprendizaje"        "datosaguas"         "datosnn1"
 [4] "entrenamientonn1"   "exactitud"          "matrizconfusion"
 [7] "nn1"                "nn1prueba"          "prediccionarsenico"
[10] "prediccionvalores"  "pruebann1"          "resultadopruebann1"
```

y que además estamos en el nuevo espacio, R82

```
> getwd()
```

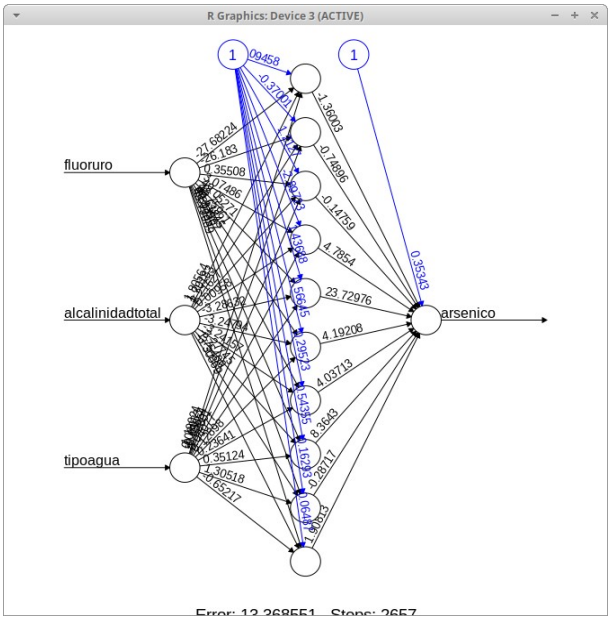
[1] "/media/alfredo/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R8/R82"
Algunas pruebas con cambios de parámetros

Parámetros originales

Primero veamos la situación planteada nuevamente. Resetee el set de prueba ya que se modifica en el proceso por agregado de columnas. Los pasos siguientes son los indicados anteriormente. Para mayor eficiencia y rapidez puede copiar el bloque resaltado a continuación y pegarlo en la consola, ejecutándolo como un script. Cargue primero la biblioteca neuralnet

```
entrenamientonn1<-datosnn1[c(1:250),]
```

```
pruebann1<-datosnn1[c(251:407),]

set.seed(300)

nn1<-
neuralnet(arsenico~fluoruro+alcalinidadtotal+tipoagua,data=entrenamientonn1,stepmax=100000,
rep=1, threshold=0.001, hidden= 10,act.fct="logistic",linear.output=FALSE)

nn1prueba=predict(nn1,pruebann1)

resultadopruebann1<-ifelse(nn1prueba>0.5,1,0)

pruebann1<-cbind(pruebann1,prediccion=resultadopruebann1)

print("Matriz de confusión")

matrizconfusion<- table(pruebann1$arsenico,pruebann1$prediccion)

matrizconfusion

exactitud<-(matrizconfusion[1,1]+matrizconfusion[2,2])/
(matrizconfusion[1,1]+matrizconfusion[1,2]+matrizconfusion[2,1]+matrizconfusion[2,2])

print("Exactitud")

exactitud

nn1$result.matrix[c(1:3),]

plot(nn1)
```



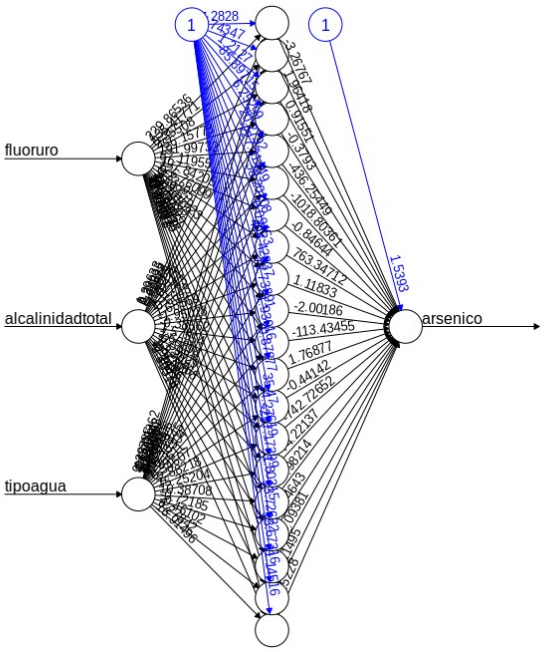
podrá comprobar que obtiene los mismos datos que en el desarrollo de la clase R81. Iremos haciendo un resumen de lo hallado

análisis	exactitud	
1.3.1	0,6751592	10 neuronas capa oculta

subimos el número de neuronas a 20

Utilizamos el mismo bloque de instrucciones, pero modificamos el parámetro hidden

```
entrenamientonn1<-datosnn1[c(1:250),]

pruebann1<-datosnn1[c(251:407),]

set.seed(300)

nn1<-
neuralnet(arsenico~fluoruro+alcalinidadtotal+tipoagua,data=entrenamientonn1,stepmax=100000,
rep=1, threshold=0.001, hidden= 20,act.fct="logistic",linear.output=FALSE)

nn1prueba=predict(nn1,pruebann1)

resultadopruebann1<-ifelse(nn1prueba>0.5,1,0)

pruebann1<-cbind(pruebann1,prediccion=resultadopruebann1)

print("Matriz de confusión")

matrizconfusion<- table(pruebann1$arsenico,pruebann1$prediccion)

matrizconfusion

exactitud<-(matrizconfusion[1,1]+matrizconfusion[2,2])/
(matrizconfusion[1,1]+matrizconfusion[1,2]+matrizconfusion[2,1]+matrizconfusion[2,2])

print("Exactitud")

exactitud

nn1$result.matrix[c(1:3),]

plot(nn1)
```



Habrá comprobado que el tiempo invertido en entrenar a la red ha sido más largo. Sin embargo si observa la exactitud verá que no ha ganado nada, sino todo lo contrario.

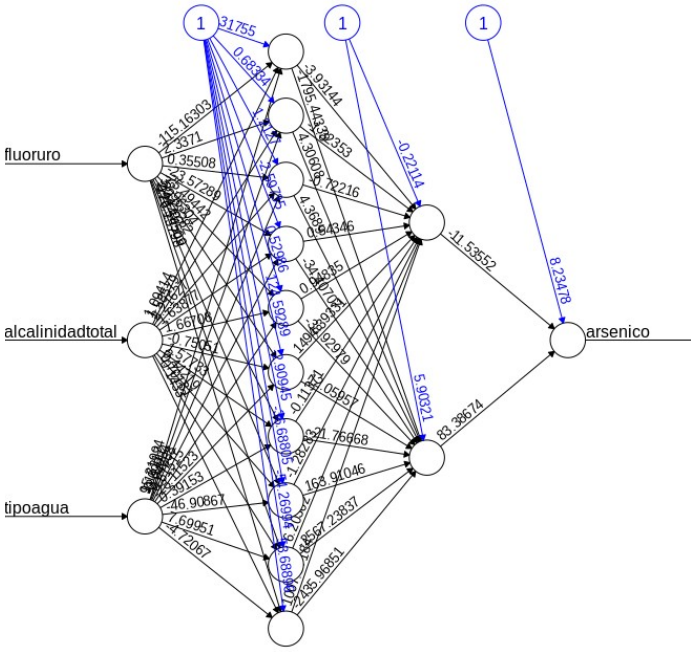
análisis	exactitud	
1.3.1	0,6751592	10 neuronas capa oculta
1.3.2	0,5987261	20 neuronas capa oculta

Más neuronas no implica mejor ajuste, al menos en este caso. Puede probar otros valores del parámetro hidden, quizás halle una mayor exactitud en la clasificación del set de prueba.

Dos capas de neuronas ocultas

Ahora pondremos dos capas ocultas, la primera de 10 neuronas y la segunda de 2. No existen reglas claras sobre el número de neuronas de las capas. Puede hallar entre los expertos diferentes opiniones, pero no tenemos la nuestra. Utilizaremos prueba y error.

```
entrenamientonn1<-datosnn1[c(1:250),]

pruebann1<-datosnn1[c(251:407),]

set.seed(300)

nn1<-
neuralnet(arsenico~fluoruro+alcalinidadtotal+tipoagua,data=entrenamientonn1,stepmax=100000,
rep=1, threshold=0.001, hidden= c(10,2),act.fct="logistic",linear.output=FALSE)

nn1prueba=predict(nn1,pruebann1)

resultadopruebann1<-ifelse(nn1prueba>0.5,1,0)

pruebann1<-cbind(pruebann1,prediccion=resultadopruebann1)

print("Matriz de confusión")

matrizconfusion<- table(pruebann1$arsenico,pruebann1$prediccion)

matrizconfusion

exactitud<-(matrizconfusion[1,1]+matrizconfusion[2,2])/
(matrizconfusion[1,1]+matrizconfusion[1,2]+matrizconfusion[2,1]+matrizconfusion[2,2])

print("Exactitud")

exactitud

nn1$result.matrix[c(1:3),]

plot(nn1)
```



análisis	exactitud	
1.3.1	0,6751592	10 neuronas capa oculta 1 ciclo entrenamiento
1.3.2	0,5987261	20 neuronas capa oculta 1 ciclo de entrenamiento
1.3.3.	0,6687898	dos capas de neuronas ocultas 1 ciclos entrenamiento

no hemos ganado exactitud con 2 capas ocultas. Puede probar otras combinaciones. Tiene que ser cuidadoso ya que el análisis puede hacerse pero en realidad la red no entrenó adecuadamente. La matriz de confusión tiene que tener la forma

```
> matrizconfusion
```

```
    0  1
0  54 43
1   9 51
```

Cuando la red no logra entrenarse pueden aparecer resultados de la matriz, de la forma

```
    1
0  97
1  60
```

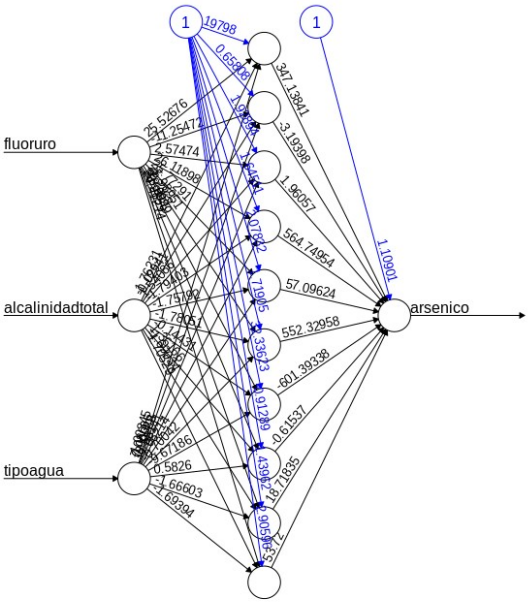
¿Qué nos indica? La matriz nos está diciendo que hay 97 falsos negativos y 60 verdaderos positivos. Pero como en nuestro bloque de instrucciones calculamos la exactitud en base a una tabla de 2x2, nos dará otro valor. En este caso la exactitud es 60/(97+60)= 0,38, que no coincide con el valor que le mostrará la pantalla. Como siempre y en todos los temas, al ejecutar comandos debemos estar muy alertas.

Será realmente frustante y vergonzoso mostrar análisis de datos utilizando "inteligencia artificial" pero dejar demostrado que nos ha faltado "inteligencia natural". Si al aplicar su algoritmo halla en la salida algún Warning o Error, no los desestime. El tiempo que invierta en dilucidar el origen, lo recuperará rápidamente. Comparta sus análisis con colegas y escúche sus opiniones!

Con 10 ciclos de entrenamiento

Como hemos visto la red se entrena con los datos de entrenamiento. Hasta ahora hicimos que lo haga con un solo ciclo. Probaremos si con 10 ciclos mejora, dejando 10 neuronas en la capa oculta. Insumirá un poco más de tiempo de cómputo!

```
entrenamientonn1<-datosnn1[c(1:250),]

pruebann1<-datosnn1[c(251:407),]

set.seed(300)

nn1<-
neuralnet(arsenico~fluoruro+alcalinidadtotal+tipoagua,data=entrenamientonn1,stepmax=100000,
rep=10, threshold=0.001, hidden= 10,act.fct="logistic",linear.output=FALSE)

nn1prueba=predict(nn1,pruebann1)

resultadopruebann1<-ifelse(nn1prueba>0.5,1,0)

pruebann1<-cbind(pruebann1,prediccion=resultadopruebann1)

print("Matriz de confusión")

matrizconfusion<- table(pruebann1$arsenico,pruebann1$prediccion)

matrizconfusion

exactitud<-(matrizconfusion[1,1]+matrizconfusion[2,2])/
(matrizconfusion[1,1]+matrizconfusion[1,2]+matrizconfusion[2,1]+matrizconfusion[2,2])

print("Exactitud")

exactitud

nn1$result.matrix[c(1:3),]

plot(nn1)
```



No hemos ganado nada en exactitud y solo hemos hecho trabajar más en la etapa de entrenamiento

análisis	exactitud	
1.3.1	0,6751592	10 neuronas capa oculta 1 ciclo entrenamiento
1.3.2	0,5987261	20 neuronas capa oculta 1 ciclo de entrenamiento
1.3.3.	0,6687898	dos capas de neuronas ocultas 1 ciclos entrenamiento
1.3.4	0,6751592	10 neuronas ocultas 10 ciclos de entrenamiento

No hemos ganado en exactitud, hemos entrenado más tiempo. Verá que la salida es diferente, nos mostrará 10 gráficas, donde la primer coincide con la red hallada con 1 ciclo de entrenamiento.

Con una capas ocultas de 50 neuronas

Durante el entrenamiento exigimos recursos de nuestros procesadores. Esto podrá notarlo por aumento de la velocidad del cooler o bien si los dispone, a través de sensores de temperatura de los procesadores. Veremos a modo de ejemplo el entrenamiento con una red con capa oculta de 50 neuronas.

```
entrenamientonn1<-datosnn1[c(1:250),]

pruebann1<-datosnn1[c(251:407),]

set.seed(300)

nn1<-
neuralnet(arsenico~fluoruro+alcalinidadtotal+tipoagua,data=entrenamientonn1,stepmax=100000,
rep=1, threshold=0.001, hidden= 50, act.fct="logistic",linear.output=FALSE)

nn1prueba=predict(nn1,pruebann1)

resultadopruebann1<-ifelse(nn1prueba>0.5,1,0)
```

```
pruebann1<-cbind(pruebann1,prediccion=resultadopruebann1)

matrizconfusion<- table(pruebann1$arsenico,pruebann1$prediccion)

print("Matriz de confusión")

matrizconfusion

exactitud<-(matrizconfusion[1,1]+matrizconfusion[2,2])/
(matrizconfusion[1,1]+matrizconfusion[1,2]+matrizconfusion[2,1]+matrizconfusion[2,2])

print("Exactitud")

exactitud

nn1$result.matrix[c(1:3),]

plot(nn1)
```

A continuación mostramos las capturas de pantalla de la temperatura de los microprocesadores. Si es usuario de Linux, abra una consola aparte de la que está ejecutando R y escriba el comando:

```
sensors
```

le mostrará la temperatura de los microprocesadores
Al inicio vemos que las temperaturas oscilan entre 41 y 45ºC, siendo el máximo 80 y la crítica 98ºC

```
coretemp-isa-0000
Adapter: ISA adapter
Package id 0:  +45.0°C  (high = +80.0°C, crit = +98.0°C)
Core 0:        +45.0°C  (high = +80.0°C, crit = +98.0°C)
Core 1:        +42.0°C  (high = +80.0°C, crit = +98.0°C)
Core 2:        +41.0°C  (high = +80.0°C, crit = +98.0°C)
Core 3:        +42.0°C  (high = +80.0°C, crit = +98.0°C)
```

Durante el entrenamiento, que afortunadamente finalizó, llegamos a los valores mostrados en la siguiente figura. Donde podemos comprobar que quedamos a prácticamente 15ºC del máximo

```
alfredo@alfredo-System-Product-Name:/media/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R8/R82$ sensors
coretemp-isa-0000
Adapter: ISA adapter
Package id 0:  +65.0°C  (high = +80.0°C, crit = +98.0°C)
Core 0:        +62.0°C  (high = +80.0°C, crit = +98.0°C)
Core 1:        +61.0°C  (high = +80.0°C, crit = +98.0°C)
Core 2:        +65.0°C  (high = +80.0°C, crit = +98.0°C)
Core 3:        +62.0°C  (high = +80.0°C, crit = +98.0°C)
```

Lo más probable es que no rompa la computadora durante el entrenamiento, pero es bueno conocer el estrés al que sometemos a nuestra PC.
Luego de ejecutar el script anterior, vemos que no llegamos a resultado satisfactorio. Hemos hallado un warning indicandonos que no se alcanzó el resultado porque se superó el máximo de pasos establecidos en el parámetro stepmax.

```
Warning message:
Algorithm did not converge in 1 of 1 repetition(s) within the stepmax.
Error in cbind(1, pred) %*% weights[[num_hidden_layers + 1]] :
  requires numeric/complex matrix/vector arguments
```

Podrá realizar otras pruebas con los parámetros y como siempre el entrenamiento, da buenos dividendos, en todos los aspectos de la vida.
Vemos que no hemos logrado mejor ajuste del modelo prácticamente con ningún cambio. Esto pasa casi siempre con todos los datos y los modelos a ajustar, se trate de redes neuronales o el modelo

que sea. En general el resultado del ajuste es muy dependiente de los datos y las variables elegidas. No es mucho lo que se gana con cambio en parámetros de ajuste, aunque siempre vale el esfuerzo de hacer sucesivas pruebas.
Reforcemos algunos conceptos utilizados en el desarrollo previo.
La función set.seed()

R tiene generadores de números aleatorios de diferentes tipos, los que han sido visto en el módulo 1. A modo de ejemplo, generemos 5 números aleatorios con distribución normal

```
> rnorm(5)
```

```
[1] -0.8204684  0.4874291  0.7383247  0.5757814 -0.3053884
```

si generamos otros 5 números aleatorios

```
> rnorm(5)
```

```
[1]  1.5117812  0.3898432 -0.6212406 -2.2146999  1.1249309
```

y otros

```
> rnorm(5)
```

```
[1] -0.04493361 -0.01619026  0.94383621  0.82122120  0.59390132
```

podemos ver las tres veces nos dieron valores diferentes, lo cual es esperable, dado que generamos números aleatorios Pero estos número en realidad no son completamente aleatorios y dependen en cierta medida de la hora del sistema. Si necesitamos números aleatorios, pero que sean los mismos, es bueno "sembrar una semilla" con la función set.seed(). Por ejemplo

```
> set.seed(1)
```

generamos 5 números aleatorios de una distribución normal

```
> rnorm(5)
```

```
[1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078
```

si generamos nuevamente números aleatorios

```
> rnorm(5)
```

```
[1] -0.8204684  0.4874291  0.7383247  0.5757814 -0.3053884
```

vemos que los valores son diferentes, porque nuestro sistema cambió. Si volvemos a sembrar la misma semilla

```
> set.seed(1)

> rnorm(5)
```

```
[1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078
```

vemos que tenemos los mismos números aleatorios.
En las redes neuronales y otros modelos de optimización esto es importante, para obtener resultados reproducibles. Al intentar ajustar un modelo de redes neuronales, si no sembramos la misma semilla tendremos resultados diferentes y podemos aun en algunos casos no hallar solución al modelo.

Grupo de entrenamiento y prueba

En el caso que aplicamos la red neuronal creamos dos grupos: aprendizaje y prueba y la división la hicimos simplemente tomando los primeros 250 datos para entrenamiento y los finales restantes para prueba, pero podrían existir problemas en que la aleatoriedad de la distribución no fuera la adecuada. El planteo se hizo de esta manera para no distraer la atención de la red neuronal, pero ahora veremos si los datos fueron realmente aleatorios y en caso de no serlos que solución darle al problema.
Podemos verificar esto sacando las proporciones en ambos grupos

Buscamos el número de datos dentro de entrenamientonn1 que tiene arsenico igual a 1 y lo dividimos por el total de datos del grupo

```
> nrow(entrenamientonn1[entrenamientonn1$arsenico==1,])/nrow(entrenamientonn1)
```

```
[1] 0.648
```

un 64.8 % de las muestras de entrenamiento tienen arsénico mayor a 10 ppb
lo mismo hacemos con el grupo de prueba

```
> nrow(pruebann1[pruebann1$arsenico==1,])/nrow(pruebann1)
```

```
[1] 0.3821656
```

un 38,2% de las muestras del grupo de prueba tiene arsénico mayor a 10 ppb
¿Son realmente diferentes estas proporciones? Tenemos recursos para analizarlo, a través de un test de proporciones, visto en módulo 3 de este curso.
Primero nos fijamos el número de datos totales de cada grupo

```
> nrow(entrenamientonn1)
```

```
[1] 250
```

```
> nrow(pruebann1)
```

```
[1] 157
```

y la cantidad de datos con arsénico mayor a 10 ppm en cada grupo

```
> nrow(entrenamientonn1[entrenamientonn1$arsenico==1,])
```

```
[1] 162
```

```
> nrow(pruebann1[pruebann1$arsenico==1,])
```

```
[1] 60
```

construimos un data.frame al que llamamos proporcion que contengan estos datos.

```
> proporcion<-data.frame(grupo=c("entremiento","prueba"),total=c(250,157),arsenicomayor10=c(162,60))
```

vemos su contenido

```
> proporcion
```

```
          grupo total arsenicomayor10
1  entremiento   250             162
2      prueba 1   57              60
```

hacemos el test de proporciones

```
> prop.test(proporcion$arsenicomayor10,proporcion$total, conf.level=0.05)
```

```
        2-sample test for equality of proportions with continuity correction

data:  proporcion$arsenicomayor10 out of proporcion$total
X-squared = 26.426, df = 1, p-value = 2.739e-07
alternative hypothesis: two.sided
5 percent confidence interval:
 0.2575673 0.2741015
sample estimates:
   prop 1    prop 2
0.6480000 0.3821656
```

El p-value es mucho menor a 0.05, por lo tanto el test nos indica que no están igualmente distribuidos los datos de arsénico dentro de los grupos de entrenamiento y prueba. Esto puede afectar el ajuste del modelo y por ende de nuestras conclusiones.
Para estas situaciones R, tiene herramientas que permiten una división equilibrada de los datos. Utilizaremos la biblioteca caTools, instálela y cárguela en su espacio de trabajo

```
> library(caTools)
```

permite dividir un data.frame manteniendo una proporción entre valores de una de sus variables.
Nuestros datos procesados tenían el nombre datosnn1 y la variable arsénico toma el valor 0 a 1.
Veamos los datos

```
> summary(datosnn1)
```

```
    fluoruro        alcalinidadtotal      tipoagua        arsenico
 Min.   :0.04688   Min.   :  20.0    Min.   :1.00   Min.   :0.0000
 1st Qu.:0.23389   1st Qu.: 110.5    1st Qu.:1.00   1st Qu.:0.0000
 Median :0.56086   Median : 309.3    Median :2.00   Median :1.0000
 Mean   :0.81571   Mean   : 334.7    Mean   :1.84   Mean   :0.5455
 3rd Qu.:1.17133   3rd Qu.: 473.5    3rd Qu.:2.00   3rd Qu.:1.0000
 Max.   :5.44843   Max.   :1828.7    Max.   :5.00   Max.   :1.0000
```

El total de nuestros datos es 407 y queremos 250 en entrenamiento

```
> 250/407
```

```
[1] 0.6142506
```

es decir el 61.42506% pertenecen al grupo de entrenamiento. Entonces con la función sample.split() fijamos este datos.
Un detalle importante es que la formación de los grupos utilizando caTools, es un proceso aleatorio y cambiará de una vez a la otra que lo realicemos y por ende cambiará luego el entrenamiento de nuestra red. Debemos por lo tanto sembrar una semilla, que permita repetir el proceso de la misma manera.
En este caso sembraremos una semilla en 1100.

```
> set.seed(1100)

>  split<-sample.split(datosnn1$arsenico,SplitRatio=0.6142506)
```

y construimos dos vectores, que llamaremos entrenamientonn1split y pruebann1split, para diferenciarlos de los dos vectores creados para entrenamiento y prueba

```
> entrenamientonn1split<-subset(datosnn1,split==1)

> pruebann1split<-subset(datosnn1,split==0)
```

podemos ver la proporción de muestras con arsénico igual a 1 en ambos grupos

```
> nrow(entrenamientonn1split[entrenamientonn1split$arsenico==1,])/nrow(entrenamientonn1split)

[1] 0.544

> nrow(pruebann1split[pruebann1split$arsenico==1,])/nrow(pruebann1split)

[1] 0.5477707
```

Casi que a "simple vista" podemos decir que no son diferentes las proporciones. Pero para usuarios avanzados de R, el test de propociones nos dará un respuesta "sustentable". Hacemos otro data frame, vemos primero la cantidad de muestras con arsenico=1 en cada grupo

```
> nrow(entrenamientonn1split[entrenamientonn1split$arsenico==1,])
```

```
[1] 136
> nrow(pruebann1split[pruebann1split$arsenico==1,])

[1] 86
```

Construimos un data.frame para luego hacer el test de proporciones

```
> proporcionsplit<-data.frame(grupo=c("entremiento","prueba"),total=c(250,157),arsenicomayor10=c(136,86))
```

y realizamos el test de proporciones

```
> prop.test(proporcionsplit$arsenicomayor10,proporcionsplit$total, conf.level=0.05)


        2-sample test for equality of proportions with continuity correction

data:  proporcionsplit$arsenicomayor10 out of proporcionsplit$total
X-squared = 7.1986e-30, df = 1, p-value = 1
alternative hypothesis: two.sided
5 percent confidence interval:
 -0.010720380  0.003178978
sample estimates:
   prop 1    prop 2
0.5440000 0.5477707
```

El valor de p-value igual a 1, nos indica ausencia total de diferencia entre las proporciones de arsénico mayor que 10 en los grupos de entrenamiento y prueba

En el futuro, cuando separemos los datos en un test de entrenamiento y prueba, utilizaremos este recurso.

Entonces, volvamos hacia atrás y apliquemos el código original a estos grupos de entrenamiento y prueba.

Es una buena idea hacer un script con los cálculo para dividir los datos en entrenamiento y prueba y además realizar el entrenamiento y el testeo de la red. A continuación se muestra el código del script, que no es otra cosa que lo que hicimos antes, eliminando los comentarios

```
set.seed(1100)

split<-sample.split(datosnn1$arsenico,SplitRatio=0.6142506)

entrenamientonn1<-subset(datosnn1,split==1)

pruebann1<-subset(datosnn1,split==0)

set.seed(300)

nn1<-
neuralnet(arsenico~fluoruro+alcalinidadtotal+tipoagua,data=entrenamientonn1,stepmax=100000,
rep=1, threshold=0.001, hidden= 10,act.fct="logistic",linear.output=FALSE)

nn1prueba=predict(nn1,pruebann1)

resultadopruebann1<-ifelse(nn1prueba>0.5,1,0)

pruebann1<-cbind(pruebann1,prediccion=resultadopruebann1)

matrizconfusion<- table(pruebann1$arsenico,pruebann1$prediccion)
```

```
print(matrizconfusion)

exactitud<-(matrizconfusion[1,1]+matrizconfusion[2,2])/
(matrizconfusion[1,1]+matrizconfusion[1,2]+matrizconfusion[2,1]+matrizconfusion[2,2])

print(exactitud)

nn1$result.matrix[c(1:3),]

plot(nn1)
```

Puede ocurrir que aparezca este mensaje, en este o en otros casos
Warning message:
Algorithm did not converge in 1 of 1 repetition(s) within the stepmax.
Indicando que 100000 pasos no fueron suficiente. Amplie el número de pasos, por ejemplo a 1000000
veamos la matriz de confusión

```
> matrizconfusion
```

```
     0  1
0   57  14
1   13  73
```

y la exactitud

```
> exactitud
```

[1] 0.8280255
De la observación del valor, vemos lo indiscutible del procesamiento y orden previo en el manejo de los datos de entrenamiento y prueba

análisis	exactitud	
1.3.1	0,6751592	10 neuronas capa oculta 1 ciclo entrenamiento
1.3.2	0,5987261	20 neuronas capa oculta 1 ciclo de entrenamiento
1.3.3.	0,6687898	dos capas de neuronas ocultas 1 ciclos entrenamiento
1.3.4	0,6751592	10 neuronas ocultas 10 ciclos de entrenamiento
1.5	0.8280255	10 neuronas ocultas 1 ciclos 100000 pasos grupos de entrenamiento preparados con sample.split()

Erroneamente muchas personas piensan que "el aprendizaje automatizados" o "la inteligencia artificial", nos ahorrará horas de aprendizaje y de uso de nuestra inteligencia. No hay peor inteligencia artificial que la que se hace sin inteligencia natural.

Módulo 8. Clase 3

Regresión con redes neuronales

Como hemos mencionado, en el CUEM se introdujeron herramientas de aprendizaje automatizado para poder anticipar un valor de arsénico en agua, sin haberlo medido, pero utilizando los valores de otras mediciones que son rápidas, precisas y económicas. Luego cuando se juntan las muestras suficientes, se hace la medición y se envía el informe definitivo. La Figura 1 esquematiza el problema y la solución. Ni bien la muestra ingresa, se hacen las determinaciones y la "inteligencia artificial" de Atlantis 3.0, nuestro software de administración del laboratorio de aguas, incorpora esos valores, hace un nuevo aprendizaje utilizando la base de datos, toma los valores medidos de las variables: alcalinidad y fluoruro, y genera un valor de arsénico en segundos. Si el valor de arsénico es "aceptable", se sigue adelante con el tratamiento de la muestra y se hará la medición de arsénico "cuando se hayan juntado 10 muestras". Si la concentración de arsénico "no es aceptable", Atlantis genera una alerta y envía un mail al encargado de la medición de arsénico, indicándole que aunque no tenga 10 muestras "debe medir la muestra inmediatamente".

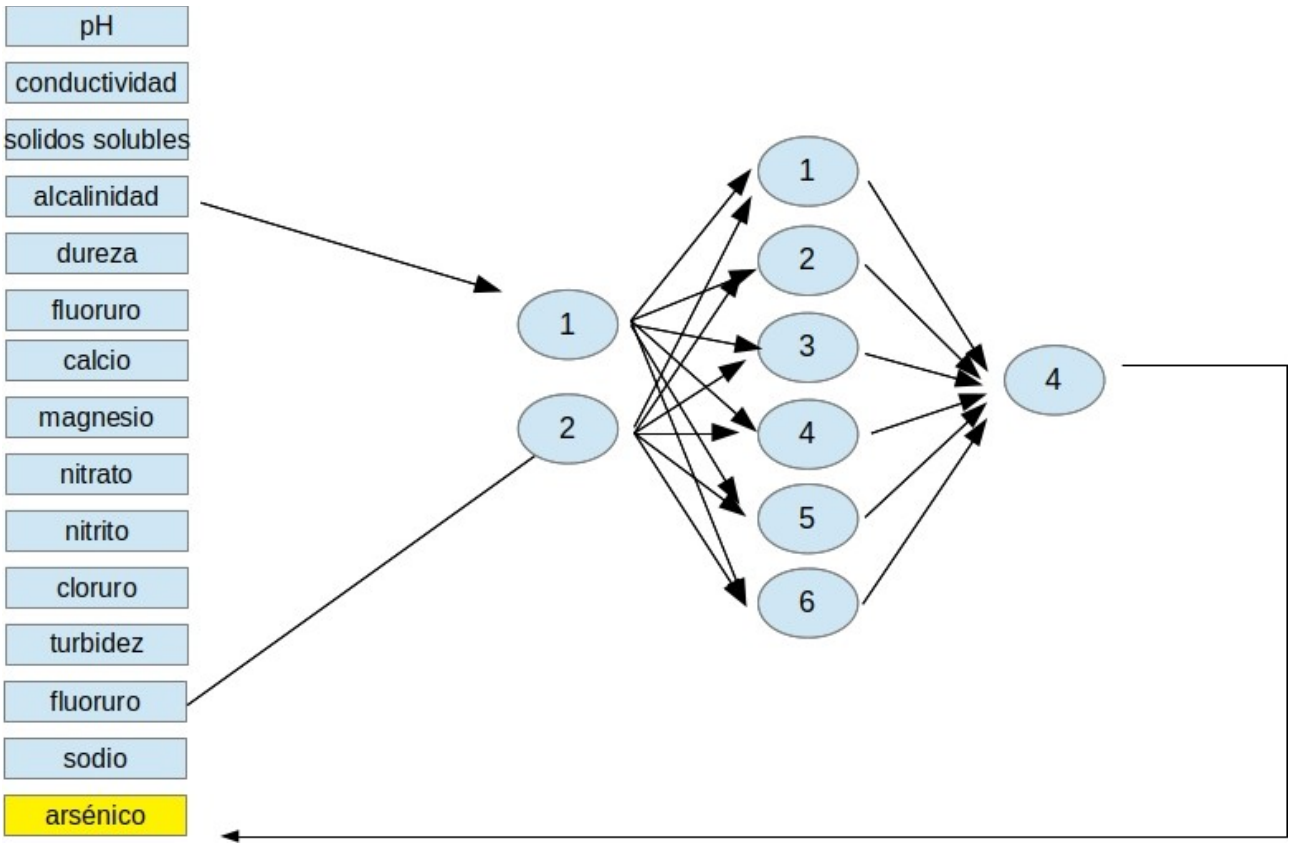


Figura 1

En clases anteriores vimos la solución a este problema, utilizando redes neuronales que nos permitieron clasificar a una muestra como arsenico>10ppb o arsénico<10ppb, siendo 10 ppb el límite establecido por la OMS y otras legislaciones sobre la calidad del agua potable. Veamos entonces como procedemos para "anticipar" el valor de la concentración de arsénico.

Preparación de los datos

Utilizaremos los mismos datos que en las clases anteriores, aunque por cuestiones de orden hallará los mismos en un archivo .RData llamado DatosR83.RData, que corresponde a un data.frame que será de utilidad. Copie este archivo al directorio donde funcionará su espacio de trabajo.
Luego desde su espacio de trabajo cargue los datos con el siguiente código

```
> load("datosR83.RData")
```

compruebe que los datos fueron introducidos. Debe hallar en su espacio de trabajo un data.frame llamado datosaguas

```
> ls()

[1] "datosaguas"
```

Esta base de datos tiene una gran cantidad de registros (cada muestra de agua se identifica con código alfanumérico, ej A12) y campos en que se almacen datos y valores de las mediciones.
Verificamos primero el número de registros, que corresponde a uno por línea de la base de datos

```
 > nrow(datosaguas)
```

[1] 425
y el nombre de los campos

```
> names(datosaguas)
```

```
 [1] "codigo"              "fecha"               "localidad"
 [4] "provincia"           "tipoagua"            "responsble"
 [7] "ph"                  "cvph"                "conductividad"
[10] "cvconductividad"     "cloruro"             "cvcloruro"
[13] "carbonato"           "cvcarb"              "bicarbonato"
[16] "cvbicar"             "solidostotales"      "cvsolidostotales"
[19] "fosfato"             "cvfosfato"           "nitrato"
[22] "cvnitrato"           "nitrito"             "cvnitrito"
[25] "fluoruro"            "cvfluoruro"          "arsenico"
[28] "cvarsenico"          "amonio"              "cvamonio"
[31] "observaciones"       "alcalinidadtotal"    "CValcalinidadtotal"
[34] "consumo"             "sulfato"             "cvsulfato"
[37] "informado"           "sodio"               "cvsodio"
[40] "dqo"                 "cvdqo"               "tkn"
[43] "cvtkn"               "calcio"              "cvcalcio"
[46] "durezatotal"         "cvdurezatotal"       "duda"
[49] "numeropersonas"      "orp"                 "cvorp"
[52] "magnesio"            "cvmagnesio"          "servicio"
[55] "materiaorganica"     "litio"               "cvlitio"
[58] "arsenicoIII"         "cvarsenicoIII"       "iodo"
[61] "cviodo"              "especiescloro"       "cvespeciescloro"
[64] "muestraagotada"      "manganeso"           "cvmanganeso"
[67] "estroncio"           "cvestroncio"         "plomo"
[70] "cvplomo"             "turbidez"            "cvturbidez"
```

Cargue la biblioteca neuralnet

```
> library(neuralnet)
```

En este caso utilizaremos solo los datos de fluoruro, alcalinidadtotal y arsenico. Con estas variables se podrá entender el enfoque, pero no es una limitación a que se vayan incorporando nuevas variables a la red neuronal. Asignamos los datos de esas variables a un nuevo objeto

```
> datosnn3<-datosaguas[,c(25,32,27)]
```

verificamos el nombre de los campos

```
> names(datosnn3)
```

```
[1] "fluoruro"         "alcalinidadtotal"         "arsenico"
```

y sus valores

```
> summary(datosnn3)
```

```
   fluoruro          alcalinidadtotal    arsenico
 Min.   :0.04688     Min.   :  20.0   Min.   :  0.000
 1st Qu.:0.21615     1st Qu.: 108.2   1st Qu.:  2.905
 Median :0.54744     Median : 307.1   Median : 11.752
 Mean   :0.79580     Mean   : 328.2   Mean   : 19.927
 3rd Qu.:1.11908     3rd Qu.: 467.7   3rd Qu.: 27.296
 Max.   :5.44843     Max.   :1828.7   Max.   :220.741
 NA's   :1                            NA's   :4
```

El objetivo en este caso es trabajar con los valores de arsénico y poder anticipar un valor numérico del mismo y en base a él tomar futuras decisiones sobre el destino de la muestra en el laboratorio y el informe al interesado.
Eliminaremos filas con datos NA en fluoruro, arsenico y alcalinidadtotal, reescribiendo el resultado sobre el data.frame datosnn3. Escribimos la función subset() utilizando la función is.na(), la que utilizamos evaluando su igualdad a FALSE. La función subset seleccionará todas las líneas que is.na() sea FALSE, es decir seleccionará todas las líneas de datosnn3 que tengan un valor medido para las variables fluoruro, arsenico y alcalinidadtotal

```
> datosnn3<-subset(datosnn3,is.na(datosnn3$fluoruro)==FALSE & is.na(datosnn3$arsenico)==FALSE & is.na(datosnn3$alcalinidadtotal)==FALSE)
```

Comprobemos el número de filas que quedan en la base de datos

```
> nrow(datosnn3)

[1] 421
```

De los 421 registros tomaremos 250 como grupo de entrenamiento y 171 como grupo de prueba. Es decir 59.4% de los datos para entrenamiento y el resto para prueba. Utilizaremos para hacer la división a la herramienta sample.split() de la biblioteca caTools, como hemos visto en la clase anterior.

```
> library(caTools)
```

generamos el objeto split, con el set de datos y la variable arsenico, fijando el porcentaje de división en 0.594
Como el proceso de división utiliza procesos aleatorios, sembramos una semilla, por si tenemos que repetir el proceso

```
> set.seed(100)

> split<-sample.split(datosnn3$arsenico,SplitRatio=0.594)
```

y construimos dos vectores, que llamaremos entrenamientonn3 y pruebann3

```
> entrenamientonn3<-subset(datosnn3,split==1)

> pruebann3<-subset(datosnn3,split==0)
```

podemos ver la proporción de muestras en cada grupo

```
> nrow(entrenamientonn3)/nrow(datosnn3)

[1] 0.5928242
> nrow(pruebann3)/nrow(datosnn3)

[1] 0.4061758
```

Podemos verificar que los grupos han quedado bastante homogéneos en lo que respecta a los valores de arsénico, si calculamos las medias para ambos grupos

```
> mean(entrenamientonn3$arsenico)
```

[1] 19.63957

```
> mean(pruebann3$arsenico)
```

[1] 20.34814

Una comprobación adicional que podemos hacer para evaluar la distribución de las muestras entre ambos grupos por sample.split() es comparar ambos grupos. Como se ve en los cálculos siguientes, los valores de arsénico no siguen distribución normal, analizada con shapiro.test()

```
> shapiro.test(pruebann3$arsenico)
```

Shapiro-Wilk normality test

data: pruebann3$arsenico

W = 0.63298, p-value < 2.2e-16

```
> shapiro.test(entrenamientonn3$arsenico)
```

Shapiro-Wilk normality test

data: entrenamientonn3$arsenico

W = 0.72386, p-value < 2.2e-16

por ende para comparar los valores de arsénico entre los grupos de entrenamiento y prueba utilizaremos el test no paramétrico, wilcox.test()

```
> wilcox.test(entrenamientonn3$arsenico,pruebann3$arsenico)
```

Wilcoxon rank sum test with continuity correction

data: entrenamientonn3$arsenico and pruebann3$arsenico

W = 21686, p-value = 0.8002

alternative hypothesis: true location shift is not equal to 0

Por el valor de p-value vemos que la selección de las muestras ha sido de tal manera que no hay una tendencia marcada en los valores de arsénico dentro de cada grupo.

Entrenamiento de la red neuronal

Construimos una red neuronal en la que relacionamos el valor de arsenico con fluoruro y alcalinidadtotal, utilizando los datos entrenamientonn3.

Fijamos algunos argumentos

Crearemos la red con una capa oculta de 10 neuronas, hidden=10

La función de activación (act.fct) la dejamos en logistic.

stepmax=100000. Limitará los pasos de cálculo si superan a 100000 indicará que no ha obtenido un resultado

rep=1. Realiza el entrenamiento 1 sola vez

threshold=0.001. Es un criterio de finalización utilizado en el proceso de derivación.

Sembramos la semilla que nos dará previsibilidad al proceso aleatorio

```
> set.seed(200)
```

aplicamos la funcion neuralnet()

```
> nn3<-neuralnet(arsenico~fluoruro+alcalinidadtotal,data=entrenamientonn3,stepmax=500000,rep=1,threshold=0.001,hidden=10,linear.output=TRUE)
```

Como hemos visto podemos revisar los datos de la variable respuesta (arsenico), las variables explicativas o predictoras (fluoruro y alcalinidadtotal), la lista de variables, el nombre de la variable

respuesta, todos los datos y los valores calculados de arsenico del grupo entrenamientonn3 con los siguientes comandos

```
>nn3$response

>nn3$covariate

>nn3$model.list$variables

>nn3$model.list$response

>nn3$data

>nn3$net.result
```

Podemos graficar la red obtenida donde vemos algunos de los datos ya mostrados como el error y los pasos.

```
> plot(nn3)
```

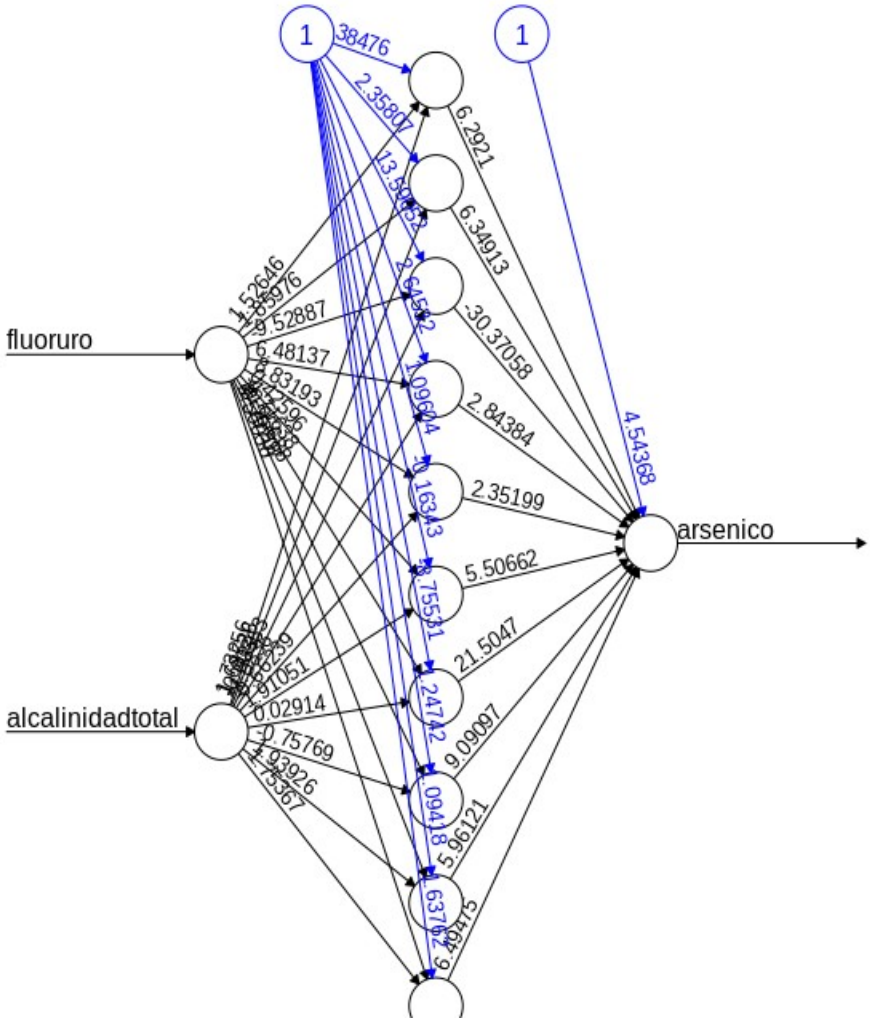


Figura 2

Prueba del entrenamiento

Ahora aplicaremos la red ya entrenada al set de datos de prueba: pruebann3. Para esto utilizamos la función predict()

```
> prediccionnn3=predict(nn3,pruebann3)
```

vemos las primeras 20 líneas de los valores de arsénico que nuestra red nn3 predijo.

```
> head(prediccionnn3,10)
```

```
       [,1]
8   4.993803
12 52.289568
13 11.495769
16 26.016315
17 55.403822
21 43.850879
23 44.961750
25 42.096803
28  9.381906
29 41.220953
```

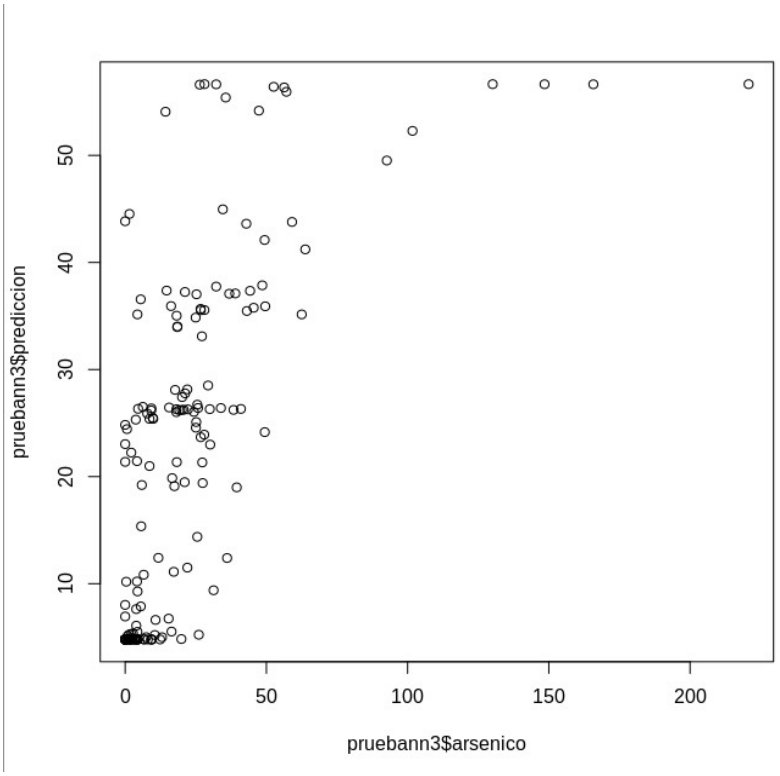
agregamos al objeto pruebann3, los valores hallados que tenemos en prediccionnn3
Recordamos que pruebann3, son los datos seleccionados como datos de testeo de funcionamiento

```
> pruebann3<-cbind(pruebann3,prediccion=prediccionnn3)

> summary(pruebann3)
```

```
   fluoruro          alcalinidadtotal      arsenico          prediccion
 Min.   :0.04688   Min.   :  23.42     Min.   :  0.000   Min.   : 4.782
 1st Qu.:0.21611   1st Qu.:  96.69     1st Qu.:  2.597   1st Qu.: 4.827
 Median :0.56975   Median : 305.08     Median : 10.749   Median :21.351
 Mean   :0.87191   Mean   : 330.32     Mean   : 20.348   Mean   :21.217
 3rd Qu.:1.26392   3rd Qu.: 458.23     3rd Qu.: 26.762   3rd Qu.:33.992
 Max.   :5.44843   Max.   :1828.68     Max.   :220.741   Max.   :56.652
```

Antes de seguir con los análisis, hagamos una observación de los resultados anteriores. Si se comparan las estadísticas de la columna arsenico (valores medido de arsénico en grupo de prueba) y la columna prediccion (valores de arsénico que la red nn3 ha predicho), los números son alentadores. Vemos una buena coincidencia entre los valores, siendo la principal diferencia en el valor máximo medido: 220.741, contra el valor que la red ha predicho en 56.652.
Una buena comprobación de la predicción será hacer una correlación entre las columnas arsenico y prediccion del objeto pruebann3. Veamos primero la gráfica

```
> plot(pruebann3$arsenico,pruebann3$prediccion)
```



Pero la respuesta final la dará el cor.test() entre las variables arsenico y prediccion de pruebann3

```
> cor.test(pruebann3$arsenico,pruebann3$prediccion)
```

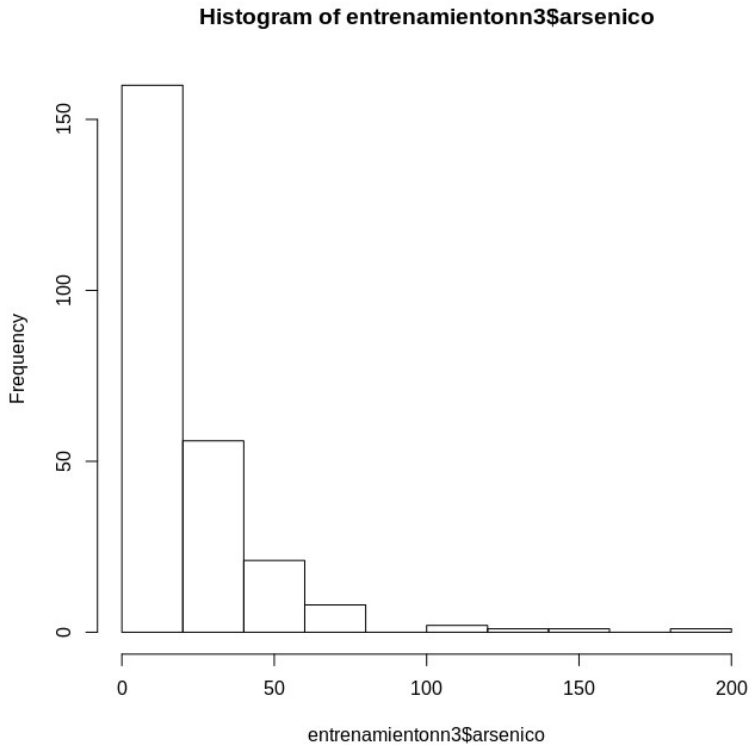
Pearson's product-moment correlation

data: pruebann3$arsenico and pruebann3$prediccion
t = 11.621, df = 169, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
0.5737794 0.7422935
sample estimates:
cor
0.6664636

El valor de p-value<0.05 nos está indicando una buena correlación entre valores medidos y predichos. Podríamos confiar entonces bastante en los valores de arsénico que nn3 es capaz de generar con solo disponer de las mediciones de fluoruro y alcalinidad.

Si bien el coeficiente de correlación no es malo y el p-value es bueno la observación de la gráfica nos indica que la predicción subestima los valores de arsénico, no habiendo valores que superen 60 ppb cuando hay valores medidos que llegan a 200ppb. En primer lugar podríamos suponer que la red no está aprendiendo adecuadamente, para predecir en valores elevados. Esto podría estar ocurriendo por el escaso número de muestras con valores elevados de arsénico. Si hacemos un histograma del grupo de entrenamientonn3

```
> hist(entrenamientonn3$arsenico)
```



Donde vemos que hay muy pocas muestras con arsénico que supera 100. A pesar de este resultado no muy alentador, la parte buena es que la red predijo para todos aquellas muestras que tenian arsénico mayor a 50, valores mayores a 30 ppm. Es decir que no estaría generando falsos negativos.

Estimación del valor de arsénico en base a mediciones de fluoruro y alcalinidad

Supongamos que ingresaron 5 muestras con los valores de fluoruro y alcalinidad que usted puede hallar en la tablaR8-3.xls/ods
introduzca estos valores en su espacio de trabajo

```
> tablaR831<-read.table("clipboard",header=TRUE,dec=",",sep="\t")

> tablaR831
```

```
  fluoruro alcalinidadtotal
1     0.25            150
2     1.53            450
3     0.15            130
4     0.09             95
5     2.70            500
```

Realizamos la predicción de los valores de arsénico de estas muestras

```
> predicciontablaR831=predict(nn3,tablaR831)
```

vemos los valores estimados

```
> predicciontablaR831
```

```
         [,1]
[1,]  4.950754
[2,] 51.361465
[3,]  4.892251
[4,]  4.823195
[5,] 51.593238
```

Al ver estos valores, Atlantis 3.0 tomará las siguientes decisiones

1- Enviará un mail al encargado de medir arsénico que evalue la posibilidad de hacer la medición lo antes posible, dado que hay muestras que superan 10 ppb. Recuerde que se optimiza el proceso si contamos con 10 muestras.
2- Buscará en la base de datos los emails de las personas que trajeron las muestras de agua cuyos valores de arsénico fueron 51.36 y 51.59 y les avisará que dejen de consumir el agua, por supuesta contaminación con arsénico y que a la brevedad se confirmará ese dato.

Modo optimo de trabajo

Con el fin de optimizar los procesos, se recomienda incluir todas las instrucciones en un script de manera que resulte sencillo el cambio de parámetros. Podemos juntar todas las instrucciones que involucran generación de los grupos de entrenamiento y prueba, asi como el proceso de entrenamiento y prueba en un script que llamamos scriptnn3, que se adjunta a la clase como un archivo de texto y que tiene las siguientes instrucciones. En el proceso de entrenamiento habitualmente debemos fijar diferentes valores de semilla o tomar porcentajes diferentes del grupo de entrenamiento y prueba.

```
set.seed(100)

split<-sample.split(datosnn3$arsenico,SplitRatio=0.594)

entrenamientonn3<-subset(datosnn3,split==1)

pruebann3<-subset(datosnn3,split==0)

set.seed(200)

nn3<-
neuralnet(arsenico~fluoruro+alcalinidadtotal,data=entrenamientonn3,stepmax=500000,rep=1,threshold=0.001,hidden=10,linear.output=TRUE)

plot(nn3)

prediccionnn3=predict(nn3,pruebann3)

pruebann3<-cbind(pruebann3,prediccion=prediccionnn3)

plot(pruebann3$arsenico,pruebann3$prediccion)

print(cor.test(pruebann3$arsenico,pruebann3$prediccion))
```

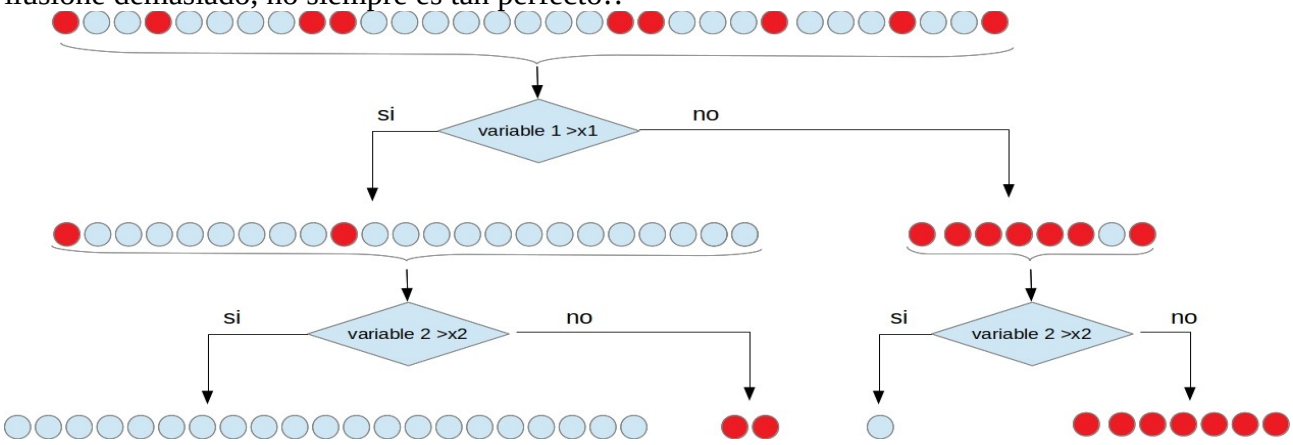
Módulo 8. Clase 4

Arboles de decisión

Los arboles de decisión son un mecanismo de aprendizaje automatizado que pueden ser utilizados en procesos de clasificación y regresión.
El funcionamiento básico de los arboles de decisión podemos explicarlo con un simple ejemplo de tres variables, una dependiente que es color de la unidad experimental en estudio y otras dos variables que consideraremos explicativas y que llamamos variable1 y variable2. Tenemos así una tabla de la siguiente forma

color	variable1	variable2
rojo	1,2	10
celeste	1,5	10,5
celeste	0,9	11
rojo	1	10,9
celeste	0,8	9,3
etc	.....	....

La idea es poder predecir el color en base a los valores de las variables 1 y 2. Los datos de la tabla son representados en la figura siguiente. Como podemos ver, hay 31 unidades experimentales, dentro de las cuales hay 9 rojas. El algoritmo toma la variable 1 y busca un valor x1 que permite dividir al grupo original en dos grupos. Si la variable1>x1 genera el grupo de la izquierda con 23 unidades y solo dos rojos. Si la variable1 es menor o igual a x1, genera el grupo de la derecha con 8 unidades donde salvo una de ellas, todas son rojas. Esto indica que la variable1 es útil para clasificar. Luego ambos grupos son separados en base a un valor de la variable2. Si la variable 2 supera un valor x2, divide al primer grupo en dos, uno de ellos todos celestes y el otro todos rojos. El mismo valor de la variable2 permite dividir al grupo de la derecha en dos: uno formado totalmente por rojos y el otro por celestes. El algoritmo funcionó de manera perfecta!. Pero no se ilusione demasiado, no siempre es tan perfecto!!



Llamamos nodo al grupo obtenido luego de cada división y regla o criterio o ley a la condición utilizada para realizar la división. Supongamos que ahora nos tapamos los ojos y tomamos una unidad experimental cualquiera. Por supuesto no podemos saber su color, pero si sabemos que la variable1<x1 y la variable2>x2, con seguridad podremos decir que el color es celeste. Se llaman nodos terminales a los nodos donde ya no se aplica una nueva regla.

Sin duda los árboles de decisión son mecanismos naturales de aprendizaje de nuestro sistema nervioso. Sabemos a que nos enfrentaremos por sus propiedades. Supongamos que un amigo nos invitó a tomar algo, pero no nos dice qué. Le hacemos una serie de preguntas a las que hallamos respuestas:
1- se toma fría: sí
2- tiene burbujas: sí
3- tiene color: sí, negro
4- es habitual consumirla en el desayuno: no
5- se prepara a partir de algún cereal: sí
6- se destila: no
Entonces, prepará dos botellas, porque la cerveza negra me encanta!!!
Preparación de los datos

Utilizaremos los datos que se hallan en el archivo datosR84.RData. Son los mismos datos de las clases anteriores que utilizaremos adecuadamente para comprender este tema. En primer lugar los introducimos al espacio de trabajo. Para ello el archivo datosR84.RData, debe estar en el mismo directorio que el espacio de trabajo que está utilizando. De ser así, lo introducimos con

```
> load("datosR84.RData")
```

y nos debería quedar el espacio de trabajo de la siguiente manera

```
> ls()
```

```
[1] "datosaguas"
```

veamos los campos de esta base de datos

```
> names(datosaguas)
```

```
 [1] "codigo"            "fecha"             "localidad"
 [4] "provincia"         "tipoagua"          "responsble"
 [7] "ph"                "cvph"              "conductividad"
[10] "cvconductividad"   "cloruro"           "cvcloruro"
[13] "carbonato"         "cvcarb"            "bicarbonato"
[16] "cvbicar"           "solidostotales"    "cvsolidostotales"
[19] "fosfato"           "cvfosfato"         "nitrato"
[22] "cvnitrato"         "nitrito"           "cvnitrito"
[25] "fluoruro"          "cvfluoruro"        "arsenico"
[28] "cvarsenico"        "amonio"            "cvamonio"
[31] "observaciones"     "alcalinidadtotal"  "CValcalinidadtotal"
[34] "consumo"           "sulfato"           "cvsulfato"
[37] "informado"         "sodio"             "cvsodio"
[40] "dqo"               "cvdqo"             "tkn"
[43] "cvtkn"             "calcio"            "cvcalcio"
[46] "durezatotal"       "cvdurezatotal"     "duda"
[49] "numeropersonas"    "orp"               "cvorp"
[52] "magnesio"          "cvmagnesio"        "servicio"
[55] "materiaorganica"   "litio"             "cvlitio"
[58] "arsenicoIII"       "cvarsenicoIII"     "iodo"
[61] "cviodo"            "especiescloro"     "cvespeciescloro"
[64] "muestraagotada"    "manganeso"         "cvmanganeso"
[67] "estroncio"         "cvestroncio"       "plomo"
[70] "cvplomo"           "turbidez"          "cvturbidez"
```

simplificaremos la tabla para no dispersar la atención y la colocaremos en el dataframe datosad1. Eliminaremos variables que tienen un gran número de datos faltantes y aquellas que son redundantes.

```
> datosad1<-datosaguas[,c(5,7,9,11,17,19,25,27,29,32,35,38,46)]
```

vemos los campos seleccionados

```
> names(datosad1)
```

```
 [1] "tipoagua"        "ph"              "conductividad"   "cloruro"
 [5] "solidostotales"  "fosfato"         "fluoruro"        "arsenico"
 [9] "amonio"          "alcalinidadtotal" "sulfato"        "sodio"
[13] "durezatotal"
```

Así nos quedamos con 13 variables, donde tipoagua será nuestra variable dependiente o variable objetivo y el resto serán variables predictoras o independientes. Veamos una descripción de la base de datos datosad1.

```
> summary(datosad1)
```

```
          tipoagua           ph                conductividad        cloruro
 red             :216   Min.   : 5.110   Min.   : 0.0000   Min.   :   0.0538
 pozo            :151   1st Qu.: 7.100   1st Qu.: 0.4150   1st Qu.:  24.2213
 envasada        : 17   Median : 7.600   Median : 0.8175   Median :  43.9884
 osmosisinversa: 14     Mean   : 7.535   Mean   : 1.3896   Mean   : 138.9622
 filtro          : 13   3rd Qu.: 8.060   3rd Qu.: 1.8200   3rd Qu.: 112.9372
 otras           :  9   Max.   :10.370   Max.   :10.5800   Max.   :2066.8424
 (Other)         :  5                                      NA's   :3
 solidostotales      fosfato             fluoruro             arsenico
 Min.       :   0    Min.   :0.00000   Min.   :0.04688   Min.   :  0.000
 1st Qu.    : 500    1st Qu.:0.03699   1st Qu.:0.21615   1st Qu.:  2.905
 Median     : 935    Median :0.35553   Median :0.54744   Median : 11.752
 Mean       :1131    Mean   :0.61369   Mean   :0.79580   Mean   : 19.927
 3rd Qu.    :1411    3rd Qu.:0.94250   3rd Qu.:1.11908   3rd Qu.: 27.296
 Max.       :7650    Max.   :4.92391   Max.   :5.44843   Max.   :220.741
 NA's       :2       NA's   :1         NA's   :1         NA's   :4
    amonio          alcalinidadtotal      sulfato            sodio
 Min.   :0.0000   Min.   :  20.0   Min.   :   0.00   Min.   :   0.00
 1st Qu .:0.0000  1st Qu  .: 108.2  1st Qu.:  21.92   1st Qu.:  33.22
 Median :0.0000   Median : 307.1   Median :  50.12   Median : 136.86
 Mean   :0.5331   Mean   : 328.2   Mean   : 134.96   Mean   : 215.41
 3rd Qu .:0.6493  3rd Qu.: 467.7   3rd Qu.: 151.33   3rd Qu.: 295.93
 Max.   :7.3751   Max.   :1828.7   Max.   :2066.34   Max.   :1871.42
                                   NA's   :2         NA's   :1
  durezatotal
 Min.   :   0.00
 1st Qu. :  86.25
 Median : 128.00
 Mean   : 186.49
 3rd Qu. : 203.55
 Max.    :1711.08
 NA's    :3
```

La base nos quedó con 425 registros

```
> nrow(datosad1)
```

[1] 425

La idea que tenemos es clasificar a las muestras de agua por el campo "tipoagua" y poder predecir el tipo de agua en función de las mediciones realizadas. Si bien este procedimiento no es muy útil en un laboratorio especializado en trabajo con agua, será un buen ejemplo para comprender el funcionamiento de esta herramienta del aprendizaje automatizado. Depuraremos la base de datos eliminando filas que tengan datos NA.

```
> datosad1<-datosad1[is.na(datosad1$cloruro)==FALSE & is.na(datosad1$solidostotales)==FALSE & is.na(datosad1$fosfato)==FALSE & is.na(datosad1$fluoruro)==FALSE & is.na(datosad1$arsenico)==FALSE & is.na(datosad1$sulfato)==FALSE & is.na(datosad1$sodio)==FALSE & is.na(datosad1$durezatotal)==FALSE,]
```

y para facilitar aun más la interpretación del tema, nos quedaremos solo con los registros que tienen tipoagua: red o pozo.

```
> datosad1<-datosad1[datosad1$tipoagua=="pozo"| datosad1$tipoagua=="red",]

> summary(datosad1$tipoagua)
```

pozo	red	osmosisinversa	envasada	superficial	filtro	otras
149	212	0	0	0	0	0

Como podemos ver nos han quedado varios niveles sin registros, entonces eliminamos los niveles sin registros

```
> datosad1$tipoagua<-droplevels(datosad1$tipoagua)

> summary(datosad1$tipoagua)
```

pozo red
149 212

Veamos el número de registros que nos quedó en la base

```
> nrow(datosad1)
```

[1]361

La variable tipoagua debe ser de tipo factor, verificamos si es correcto esto

```
> is.factor(datosad1$tipoagua)
```

[1] TRUE

Todo está en orden para comenzar a preparar los grupos de entrenamiento y prueba.

Preparación set de entrenamiento y prueba

Utilizaremos la biblioteca caTools, ya conocida en clases anteriores. Dejaremos un 70% de datos en el grupo de entrenamiento y el resto en el grupo de prueba.

```
> library(caTools)
```

sembramos una semilla

```
> set.seed(135)
```

y separamos los datos en dos grupos: entrenamiento y prueba con el porcentaje acordado recientemente

```
> split<-sample.split(datosad1$tipoagua,SplitRatio=0.7)

> entrenamiento<-subset(datosad1,split==1)

> prueba<-subset(datosad1,split==0)
```

verificamos la distribución de los datos segun el porcentaje establecido

```
> nrow(entrenamiento)/nrow(datosad1)
```

```
[1] 0.6980609
```

```
> nrow(prueba)/nrow(datosad1)
```

```
[1]  0.3019391
```

Comprobemos que el proceso de división nos generó dos grupos con porcentajes similares de cada tipo de agua

```
> table(prueba$tipoagua)/nrow(prueba)*100
```

```
   pozo     red
41.2844 58.7156
```

```
>  table(entrenamiento$tipoagua)/nrow(entrenamiento)*100
```

```
    pozo      red
41.26984 58.73016
```

Claramente el proceso fue exitoso. No hace falta perder tiempo en demostrar estadísticamente algo que es evidente! Pero si lo desea puede organizar los datos para hacer un test de proporciones.

Entrenamiento del árbol de decisión

Para este caso requerimos de la biblioteca rpart. Descárguela de los repositotios y luego cargamos la biblioteca en nuestro espacio de trabajo

```
>library(rpart)
```

sembramos la semilla. Utilizaremos la misma que para la partición de los datos, pero podría haber sido diferente. Volveremos en clases posteriores sobre este tema

```
> set.seed(135)
```

entrenamos el árbol y guardamos su resultado en el objeto ad1

```
> ad1<-rpart(tipoagua~conductividad +solidostotales + arsenico, data=entrenamiento)
```

podemos obtener información del objeto ad1

```
> ad1
```

```
n= 252

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 252 104 red (0.4126984 0.5873016)
    2) conductividad>=0.7975 139  53 pozo (0.6187050 0.3812950)
      4) conductividad>=3.2825 28   3 pozo (0.8928571 0.1071429) *
      5) conductividad< 3.2825 111  50 pozo (0.5495495 0.4504505)
       10) conductividad< 2.75 100  42 pozo (0.5800000 0.4200000)
         20) conductividad>=2.28 10   0 pozo (1.0000000 0.0000000) *
         21) conductividad< 2.28 90  42 pozo (0.5333333 0.4666667)
           42) solidostotales< 1687.5 79  34 pozo (0.5696203 0.4303797)
             84) conductividad< 0.845 7   1 pozo (0.8571429 0.1428571) *
             85) conductividad>=0.845 72  33 pozo (0.5416667 0.4583333)
              170) conductividad>=1.025 65  27 pozo (0.5846154 0.4153846)
                340) arsenico< 37.40577 51  18 pozo (0.6470588 0.3529412)
                  680) conductividad< 1.9925 40  11 pozo (0.7250000 0.2750000) *
                  681) conductividad>=1.9925 11   4 red (0.3636364 0.6363636) *
```

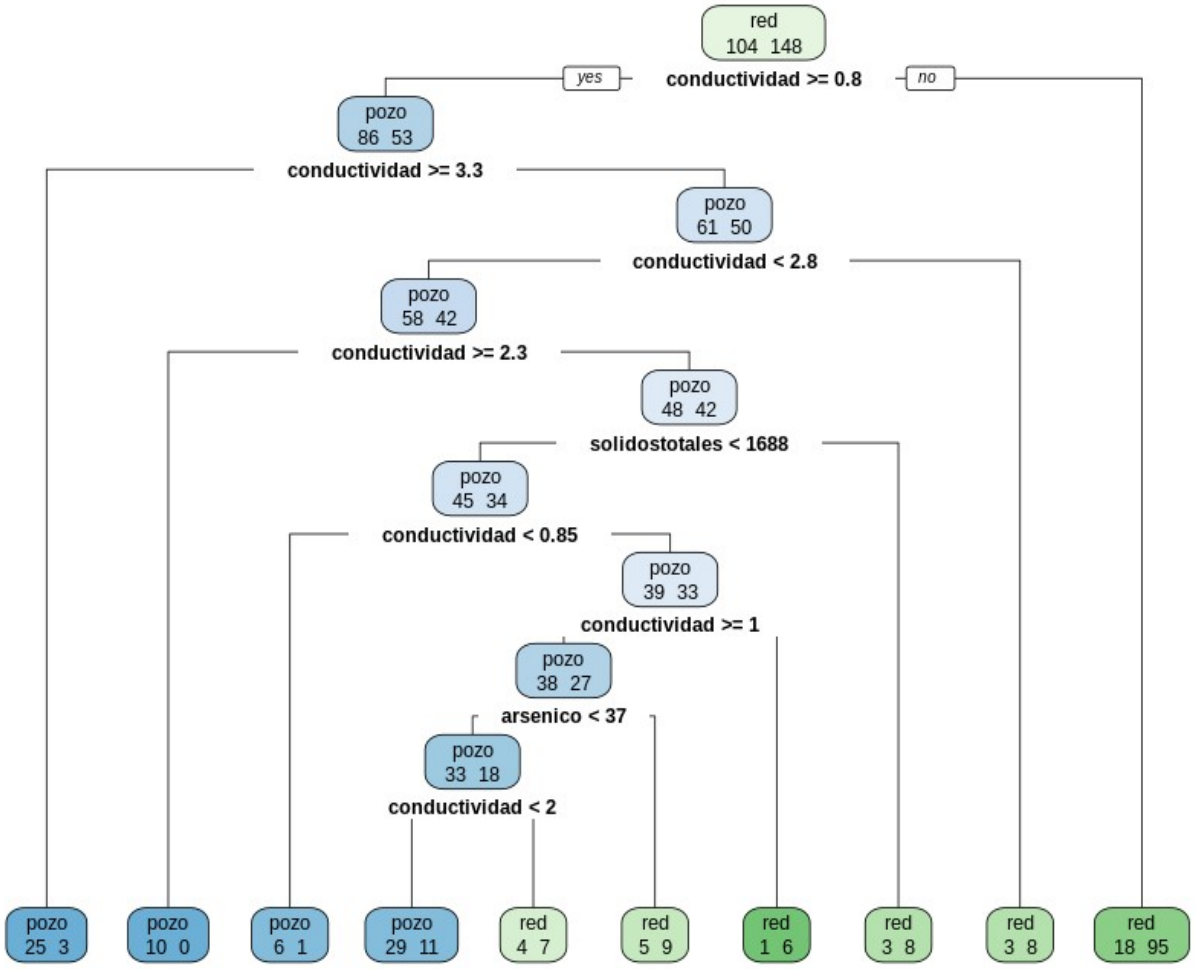
```
      341) arsenico>=37.40577 14   5 red (0.3571429 0.6428571) *
    171) conductividad< 1.025 7    1 red (0.1428571 0.8571429) *
   43) solidostotales>=1687.5 11   3 red (0.2727273 0.7272727) *
  11) conductividad>=2.75 11    3 red (0.2727273 0.7272727) *
 3) conductividad< 0.7975 113  18 red (0.1592920 0.8407080) *
```

Volveremos en la clase siguiente a ver la interpretación de la salida anterior. Ahora analizaremos la gráfica que nos proporciona el algoritmo. Para graficar un árbol de decisión requerimos la biblioteca rpart.plot. Descárguela e instálela

```
>library(rpart.plot)
```

graficamos el arbol ad1, con argumentos mínimos. Es imprescindible darle el objeto generado por rpart(), en este caso ad1 y agregaremos un argumento: extra=1 que permite ver el número de muestras pertenecientes a aguas de pozo o de red. Veremos en la clase siguiente otros argumentos que permiten ver los datos de otra manera.

```
> rpart.plot(ad1,extra=1)
```

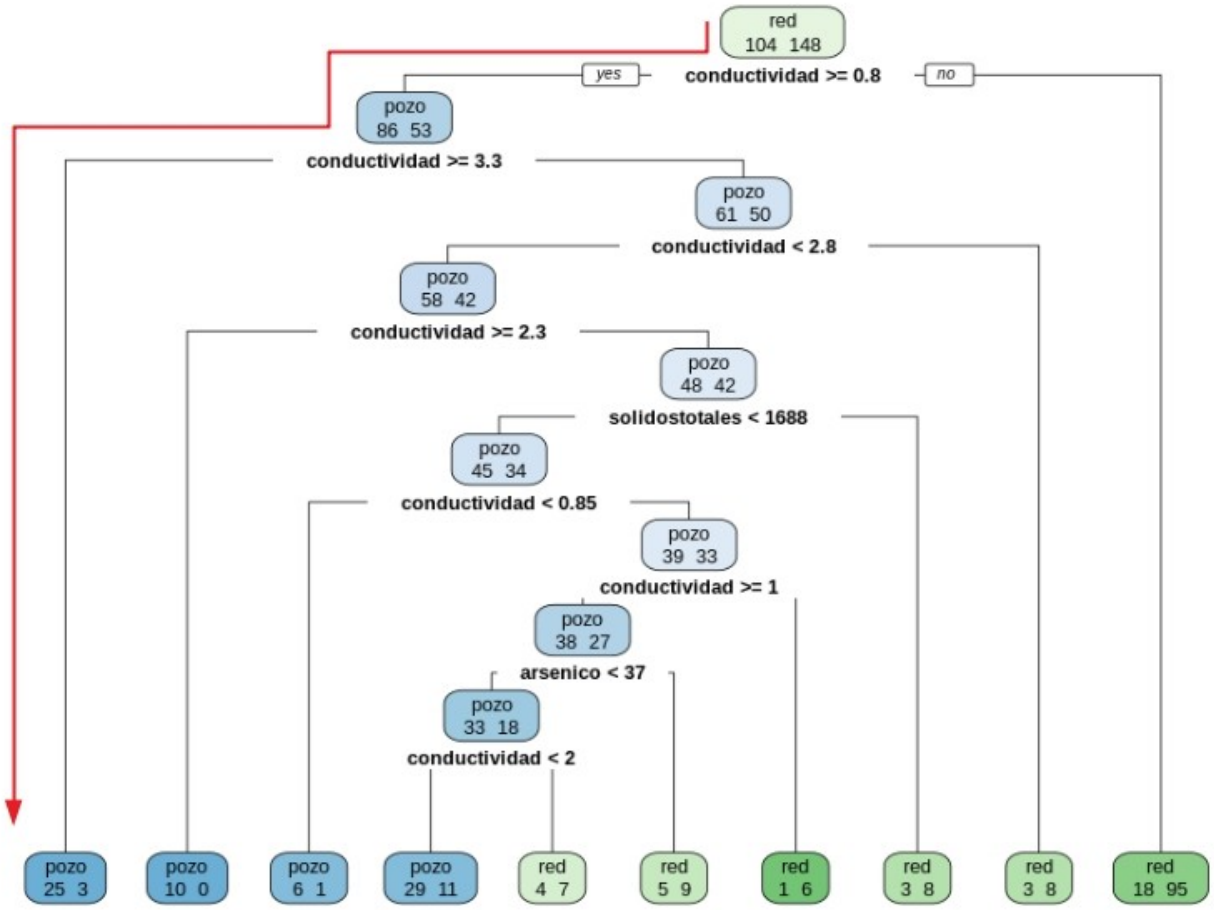


Los rectángulos de bordes redondeados se denominan nodos y contienen un determinado número de las unidades experimentales. En primer lugar resaltemos el color, en este caso hay dos colores: celeste y verde que se asocian a las categorías: pozo y red, respectivamente. Un nodo aparece del color de la categoría que predomina en el nodo y además queda identificado por el nombre en la primer línea del nodo. Como podemos ver, en todos los nodos celestes dice pozo y en los verdes

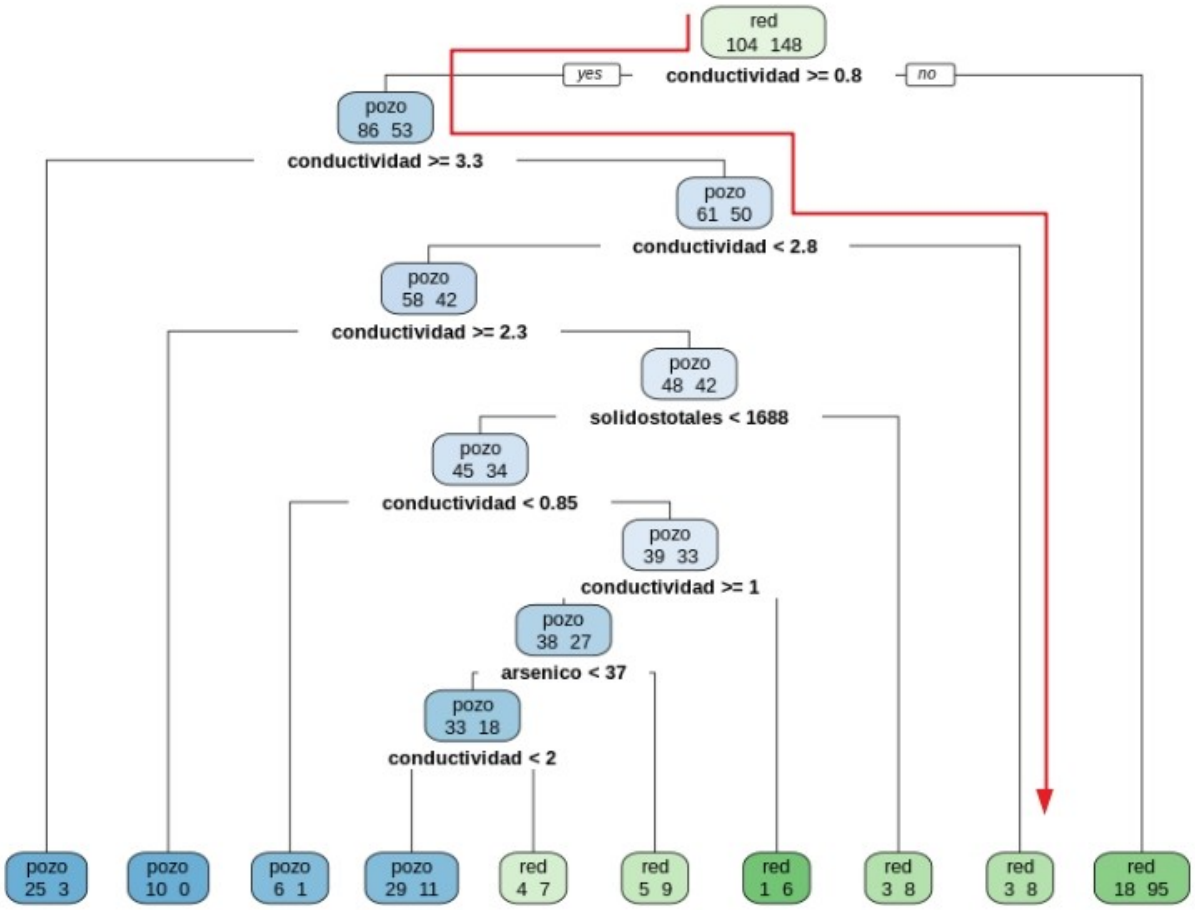
dice red. En la segunda línea vemos dos números, que es la cantidad de unidades experimentales de cada categoría. Como en este caso hay dos categorías, en cada nodo habrá dos números y estos números están ordenados alfabéticamente, según la categoría. En este caso, siempre el primer número corresponde a aguas de pozo y el segundo de red. Si nos posicionamos en el nodo de la parte inferior derecha, verá que éste es verde, indicando que en ese grupo predominan las muestras provenientes de agua de red, a su vez verá este nombre en la primer línea. El predominio de aguas de red se evidencia también en los número inferiores: 18 aguas de pozo y 95 aguas de red. Yendo al otro extremo: inferior izquierdo, tenemos un rectángulo azul con el nombre pozo, porque predominan las aguas de este origen, lo que se evidencia por los números: 25 aguas de pozo y 3 de red.
Ahora analicemos otra parte. Al inicio del árbol tenemos un recuadro de color verde, indicando el predominio de aguas de red, con los números: 104 y 148, que indican la cantidad de aguas de pozo y red, respectivamente. Es decir que partimos de 252 muestras en el grupo de entrenamiento, cosa que podemos comprobar

```
> nrow(entrenamiento)

[1] 252
> nrow(entrenamiento[entrenamiento$tipoagua=="pozo",])

[1] 104
> nrow(entrenamiento[entrenamiento$tipoagua=="red",])

[1] 148
```

El grupo completo es sometido a una primer regla de clasificación: conductividad>=0.8. De esta regla surgen dos direcciones: derecha que corresponde a **no** e izquierda que corresponde a **si**. En cada regla siempre la derecha será **no** y la izquierda será **si.** Es decir que en este punto el grupo total de 252 unidades se divide en dos grandes grupos, los que no cumplen con conductividad>=0.8 que está compuesto predominantes por aguas de red: 95 de este origen y solo 18 de pozo y ese nodo ya constituye un nodo terminal, porque como ve en el esquema, no es sometido a una próxima regla de clasificación. El árbol de decisión formó un grupo de aguas de red, donde la cantidad de falsos positivos es 18,9%, que surge de los 18 aguas de pozo que quedaron dentro del grupo de red. Por otra parte en la misma regla y dirigiéndonos hacia la izquierda tenemos las muestras que tienen conductividad >=0.8, nodo en que predominan las aguas de pozo, aunque 53, son de red. En este nodo los falsos positivos, es decir aguas que no son de pozo, pero que quedaron en este grupo llega al 38,1 % (53/(53+86)). Este nodo no es terminal y será separado con otra regla, en este ejemplo, conductividad>=3.3, que genera dos grupos con predominio de pozo. Uno de ellos un nodo terminal con 25 aguas de pozo y solo 3 red, es decir 11% de falsos positivos. En cambio el nodo que no tuvo conductividad>= 3.3, tiene un porcentaje de falsos positivos que asciende a 45%. Este nodo no es terminal y es sometido a una nueva regla: Conductividad<2.8.
Detengamosnó un minuto y hagamos el siguiente razonamiento. Si tenemos un agua cuya conductividad supera o es igual a 3.3, en la primer regla (conductividad>=0.8) se dirigirá al nodo de la izquierda y si su conducitividad supera 3.3, cae en el nodo terminal del extremo inferior izquierdo. Es decir que en este primer análisis podemos decir que si una muestra de agua tiene conductividad >=3.3, tenemos una probabilidad de 89,2% (25/28*100) de que sea de pozo. Este trazado queda representado en la figura siguiente.



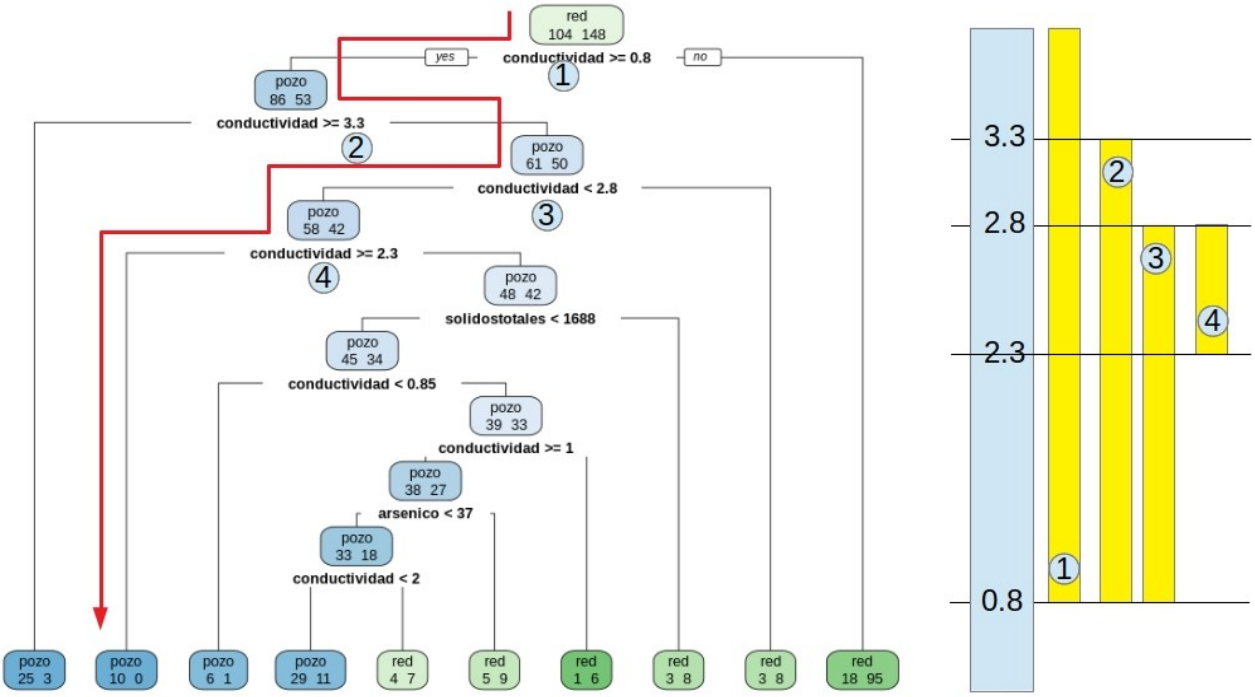
Por otra parte si una muestra supera 0.8 y no supera 3.3, cae en el nodo con predominio de pozo, de la tercer linea (pozo:61-50). Acá se evalúa si es menor a 2.8. Si no lo es se sigue hacia la derecha y cae en el nodo terminal con predominio de aguas de red (8) donde 3 son de pozo, indicando un 27.3% de falsos positivos. Este razonamiento queda plasmado en la siguiente figura. Es decir que estas muestras tienen conductividad entre 2.8 y 3.3.



Como puede ver se van constituyendo reglas, algunas con mayor y otras con menor cantidad de falsos positivos.

Pongamos ahora el foco en el segundo nodo terminal contando desde la izquierda. Verá un predominio total de aguas de pozo, con ausencia de aguas de red. Acá la clasificación ha sido perfecta: 0% de falsos positivos. ¿Qué característica tienen estas aguas?.

Analicemos el caso siguiendo el gráfico. A la derecha en barra celeste representamos las conductividades de todas las muestras. En la primer regla las aguas del nodo en cuestión superan 0.8, lo que queda identificado por el numero 1, sobre la línea 0.8. Luego este nodo es sometido a la regla 2, no superando 3.3, con lo que ya podríamos decir que las aguas tienen conductividad en el intervalo 0.8-3.3. Sin embargo la aplicación de la regla 3, nos indica que es menor de 2.8, restringiendo el rango a 0.8-2.8. La cuarta regla nos indica que supera 2.3 y así llegamos al nodo terminal con 100% de aguas de pozo. A priori y arriesgando, podríamos decir que si una muestra tiene conductividad en el intervalo 2.3-2.8 tenemos un 100% de probabilidad de que sea de pozo. Quede claro que este es un ejercicio para entender las gráficas. Aun no hemos testeado nuestro árbol de decisión con el grupo de prueba y no hemos analizado la posibilidad de mejores resultados.



Prueba del modelo

Realizaremos ahora la prueba del árbol de decisión hallado en el proceso de entrenamiento: ad1. Para ello utilizamos la función predict() y los datos de prueba, que hemos llamado prueba. El resultado la adjudicamos al objeto pruebaad1

```
> pruebaad1<-predict(ad1,newdata=prueba,type="class")
```

vemos el contenido del objeto, donde comprobamos que a cada dato del objeto prueba se asignó un tipo de agua red o pozo.

```
> pruebaad1
```

```
   6    8   10   12   15   18   19   22   26   27   32   35   41   50   54   57
 red  red  red  red pozo  red  red  red  red  red pozo  red  red pozo  red pozo
  59   67   73   83   84   85   89   91  100  103  104  108  112  114  119  121
 red  red pozo  red pozo pozo  red pozo  red  red  red  red  red pozo pozo  red
 124  126  133  134  137  145  153  156  158  161  162  167  171  173  177  195
 red pozo pozo  red pozo  red  red  red pozo pozo  red  red pozo  red  red  red
 201  210  212  215  217  219  220  231  232  234  239  240  245  246  255  256
 red pozo  red pozo  red pozo pozo pozo  red pozo  red  red  red pozo pozo  red
 257  260  264  267  268  269  274  278  282  288  290  294  295  296  298  300
 red pozo  red pozo  red pozo  red pozo pozo  red pozo  red pozo pozo  red  red
 302  305  307  308  310  321  325  331  342  344  347  350  351  357  365  371
pozo pozo pozo pozo pozo pozo  red  red pozo pozo pozo  red  red  red  red  red
 373  374  375  376  380  383  400  404  405  411  412  415  418
pozo  red pozo pozo pozo  red  red pozo pozo pozo  red  red  red
Levels: pozo red
```

La longitud de este objeto es

```
> length(pruebaad1)
```

```
[1] 109
```

que coincide con las líneas del data.frame prueba

```
> nrow(prueba)
```

```
[1] 109
```

Hacemos una primer interpretación. El data.frame prueba tiene los datos reales y medidos, el objeto pruebaad1, tiene los datos de tipoagua predichos por el modelo. Coinciden? Veamos los primeros 10 datos de cada grupo

```
>  pruebaad1[c(1:10)]
```

```
   6    8   10   12   15   18   19   22   26   27
 red  red  red  red pozo  red  red  red  red  red
Levels: pozo red
```

```
> head(prueba$tipoagua,10)
```

```
 [1] red  red  red  red  red  pozo red  red  red  red
Levels: pozo red
```

hemos marcado en verde donde hubo coincidencia entre el valor medido y la predicción. Mientras que marcamos en rojo la discordancia. Coincidencia en 8 de cada 10 no está nada mal, aunque podría ser mejor!

Ahora haremos una matriz de confusión para evaluar la exactitud del modelo, así como falso y verdaderos positivos y negativos. Utilizaremos la biblioteca caret,que permite obtener la matriz de confusión y a su vez obtener muchos valores de importancia.

```
> library(caret)

> matrizconfusion<-confusionMatrix(pruebaad1, prueba[["tipoagua"]])
```

veamos que contiene el objeto, que de hecho es muy rico en información y analizaremos a continuación, resaltando en amarillo las partes más importantes que analizaremos en esta clase

```
> matrizconfusion
```

```
Confusion Matrix and Statistics

          Reference
Prediction pozo red
     pozo    32  16
     red     13  48

               Accuracy : 0.7339
                 95% CI : (0.6407, 0.814)
    No Information Rate : 0.5872
    P-Value [Acc > NIR] : 0.001018

                  Kappa : 0.4566
 Mcnemar's Test P-Value : 0.710347

            Sensitivity : 0.7111
            Specificity : 0.7500
         Pos Pred Value : 0.6667
         Neg Pred Value : 0.7869
             Prevalence : 0.4128
         Detection Rate : 0.2936
   Detection Prevalence : 0.4404
      Balanced Accuracy : 0.7306
```

'Positive' Class : pozo

La matriz de confusión nos indica que la prueba del modelo ha sido buena. Existen 32 y 48 muestras en que coinciden la referencia con la predicción, es decir el valor real conocido del tipo de agua y el estimado. Son los verdaderos positivos. Con un número bajo de inconsistencia en la clasificación. Vemos 16 aguas que siendo de red, el modelo los predijo como pozo. Por otro lado 13 muestras de pozo fueron predichas por el modelo como aguas de red. A pesar de ello la exactitud es elevada: 0.7339, así como la sensibilidad y especificidad (0.7111 y 0.75, respectivamente). Volveremos varias veces sobre la interpretación de la matriz de confusión en el futuro.

Estimación del origen del agua en función de predictores

Supongamos que tenemos cuatro muestras donde conocemos los predictores utilizados: conductividad, solidostotales y arsenico. Podríamos estimar su origen y lo haríamos con una exactitud del 73.39%, si utilizamos el árbol desarrollado: ad1

Introducimos los datos de la tablaR841 de la planilla tablaR8-4.xls/ods

```
> tablaR841<-read.table("clipboard",header=TRUE, sep="\t",dec=",")
```

verificamos el ingreso de los datos

```
> tablaR841
```

```
  numero conductividad solidostotales arsenico
1     A1        0.30            300        10
2     A2        1.75           2500        75
3     A3        0.80           1100        25
4     A4        0.01             50         2
```

hacemos la predicción

```
> prediccionad1<-predict(ad1,newdata=tablaR841,type="class")
```

veamos la clasificación realizada

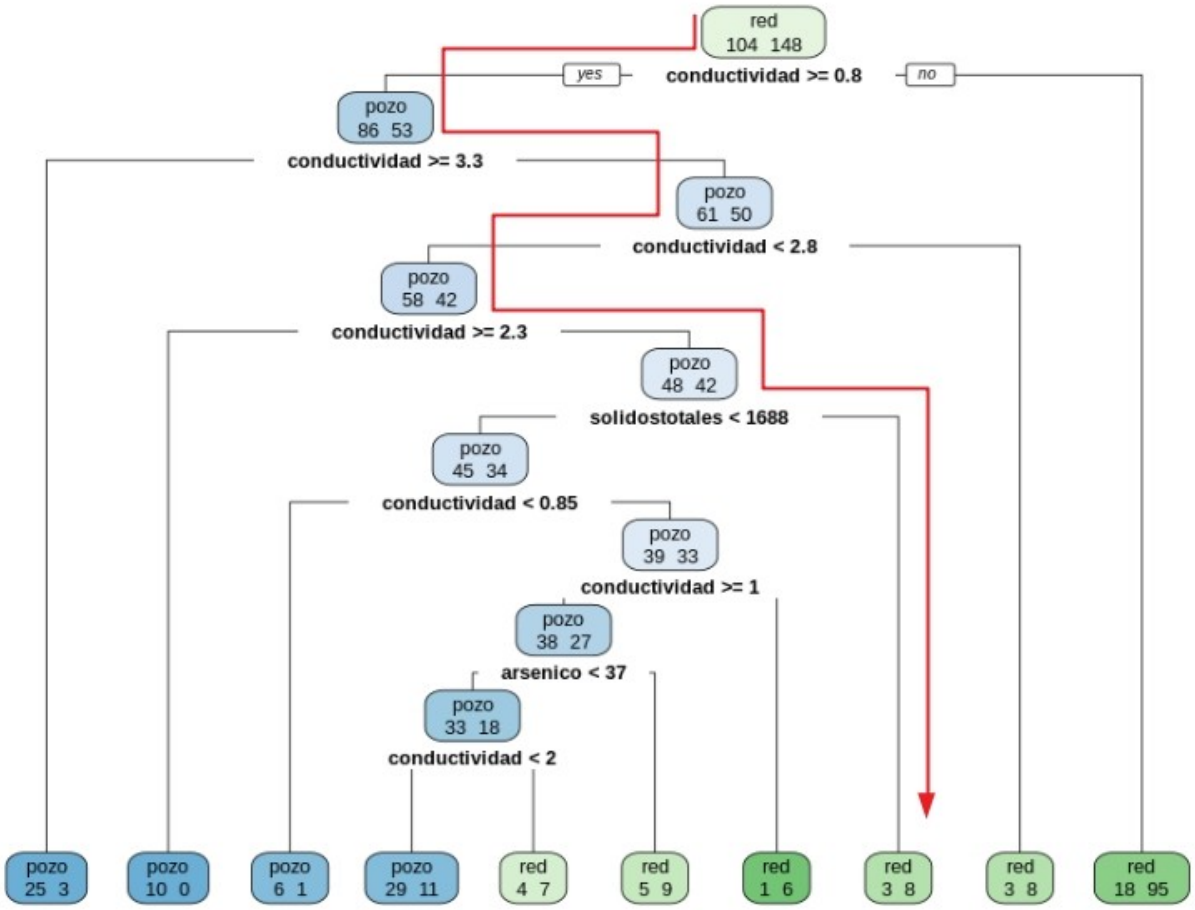
```
> prediccionad1
```

```
   1    2    3    4
 red  red pozo  red
Levels: pozo red
```

Hagamos una comprobación, siguiendo el esquema del árbol obtenido en el entrenamiento. Tomamos la muestra A2 con los siguientes valores

```
     numero  conductividad  solidostotales  arsenico
2     A2        1.75             2500          75
```

marcamos con una línea roja el recorrido de la muestra por los diferentes nodos al aplicar cada regla.



Por supuesto no es una aplicación muy útil tener un sistema de aprendizaje automatizado para saber el origen del agua, siendo que esto es lo primero que nos dice el interesado al traer la muestra para su análisis. Pero a los fines de comprender el funcionamiento de estos modelos ha sido satisfactorio. Imagine otros escenarios. Suponga que ha realizado entrenamiento y prueba de su modelo, obteniendo una exactitud del 85%. Su variable dependiente han sido patologías que pueden pasar inadvertidas por algún tiempo, además de contar con pacientes sanos, tiene pacientes con hipertensión, diabetes, Alzheimer, etc. En el entrenamiento y prueba de su modelo tiene valores de la variable dependiente medida en diversos pacientes y además numerosas pruebas de laboratorio, como podrían ser análisis de diferentes componentes de sangre, orina, marcadores genéticos, signos clínicos, etc. En sus pacientes "supuestamente sanos", si tuviera los valores de las variables predictoras y la función predict() podría diagnosticar su patología.

Otra aplicación podría ser para detectar errores en un laboratorio. Suponga que durante muchos años usted registra el trabajo de sus operarios, registrando todos los datos que surgen del proceso de medición y si este terminó o no en un error, fraude o en una equivocación. Si su árbol tuvo una buena exactitud, cada vez que un operario mida podrá estar alertado si todo anda bien o hay errores, fraude o equivocaciones posibles.

Módulo 8. Clase 5

Arboles de decisión continuación

En esta clase profundizaremos en algunos aspectos básicos de los árboles de decisión.
Los datos que utilizaremos son los mismos de la clase 4, y que quedaron dentro del data.frame datosad1. Usted puede continuar trabajando en el espacio de trabajo de la clase 4, pero a los fines organizativos sugerimos crear un nuevo espacio de trabajo. Si decide por esta última opción utilice los datos que se halla en el archivo datosR85.RData. A continuación y de una manera más resumida realizamos el procesamiento de los datos para aplicar luego las funciones correspondientes.
Introduzca los datos en su espacio de trabajo. Para ello utilice el código siguiente y recuerde que el archivo datosR85.RData, debe estar en el mismo directorio

```
> load("datosR85.RData")
```

Selección de las variables de interés

```
> datosad5<-datosaguas[,c(5,7,9,11,17,19,25,27,29,32,35,38,46)]
```

que resulta en los siguientes campos

```
> names(datosad5)
```

```
 [1] "tipoagua"         "ph"               "conductividad"    "cloruro"
 [5] "solidostotales"   "fosfato"          "fluoruro"         "arsenico"
 [9] "amonio"           "alcalinidadtotal" "sulfato"          "sodio"
[13] "durezatotal"
```

Elimine filas que tengan datos NA.

```
> datosad5<-datosad5[is.na(datosad5$cloruro)==FALSE & is.na(datosad5$solidostotales)==FALSE & is.na(datosad5$fosfato)==FALSE & is.na(datosad5$fluoruro)==FALSE & is.na(datosad5$arsenico)==FALSE & is.na(datosad5$sulfato)==FALSE & is.na(datosad5$sodio)==FALSE & is.na(datosad5$durezatotal)==FALSE,]
```

y se seleccione solo aquellos registros en el tipo de agua tiene el valor: red o pozo. Esta selección tiene el fin de hacer más fácil la comprensión de la salida de los análisis

```
> datosad5<-datosad5[datosad5$tipoagua=="pozo"| datosad5$tipoagua=="red",]
```

Elimine niveles sin datos del campo tipo agua

```
> datosad5$tipoagua<-droplevels(datosad5$tipoagua)
```

Deben quedar 361 registros, verifíquelo

```
> nrow(datosad5)
```

```
[1]361
```

A continuación preparamos los grupos de entrenamiento y prueba, dejando un 70% de datos en el grupo de entrenamiento. Para este proceso utilizamos la biblioteca caTools.

```
> library(caTools)
```

sembramos una semilla

```
> set.seed(135)
```

y separamos los datos en dos grupos: entrenamiento y prueba con el porcentaje acordado recientemente

```
> split<-sample.split(datosad5$tipoagua,SplitRatio=0.7)

> entrenamiento<-subset(datosad5,split==1)
```

```
> prueba<-subset(datosad5,split==0)
```

Realizamos las verificaciones correspondientes para evaluar que todo esté en orden. Salvo la variable tipoagua que es un factor con dos niveles, el resto de las variables son numéricas continuas. Verifique esto con la función summary(), en los grupos de entrenamiento y prueba. La función str() también es muy útil para ver esto. Veamos para el grupo de datos de entrenamiento

```
> str(entrenamiento)
```

```
'data.frame':    252 obs. of  13 variables:
 $ tipoagua         : Factor w/ 2 levels "pozo","red": 1 2 1 2 2 1 2 1 1 2 ...
 $ ph               : num  7.76 7.38 7.21 6.58 7.07 7.14 6.2 6.54 6.8 6.25 ...
 $ conductividad : num  1.305 1.81 1.777 0.637 5.61 ...
 $ cloruro          : num  27.87 57.03 55.61 7.86 241.84 ...
 $ solidostotales  : num  400 1550 1685 640 3921 ...
 $ fosfato          : num  0 1.262 1.522 0.68 0.506 ...
 $ fluoruro         : num  0.725 0.968 0.307 1.216 0.985 ...
 $ arsenico         : num  34 27.1 66.4 58.2 109.9 ...
 $ amonio           : num  0.655 0 1.866 0.482 1.232 ...
 $ alcalinidadtotal: num  531 599 757 328 716 ...
 $ sulfato          : num  102.9 224.4 89.1 17.7 1247.7 ...
 $ sodio            : num  248 424 438 130 1048 ...
 $ durezatotal      : num  83.7 63 54 51.2 252.3 ...
```

y para prueba

```
> str(prueba)
```

```
'data.frame':    109 obs. of  13 variables:
 $ tipoagua         : Factor w/ 2 levels "pozo","red": 2 2 2 2 2 1 2 2 2 2 ...
 $ ph               : num  6.64 6.77 7.45 6.49 6.75 6.84 7.34 7.32 7.5 7.62 ...
 $ conductividad  : num  0.285 0.877 1.545 1.478 1.725 ...
 $ cloruro          : num  18.3 105.1 51.5 39.2 108.6 ...
 $ solidostotales   : num  380 685 1915 1570 1250 ...
 $ fosfato          : num  0 0 0 0.933 0.533 ...
 $ fluoruro         : num  0.522 0.318 1.182 1.844 0.782 ...
 $ arsenico         : num  3.66 13.04 65.74 101.73 31.61 ...
 $ amonio           : num  2.299 2.443 3.797 3.251 0.308 ...
 $ alcalinidadtotal: num  65.3 161.6 689.3 457.1 621.9 ...
 $ sulfato          : num  27.5 145.4 152.4 106.8 354.5 ...
 $ sodio            : num  38.1 67.3 370.1 387.9 496.6 ...
 $ durezatotal      : num  71.1 108.1 52.9 66.5 67.3 ...
```

Cargamos la biblioteca rpart, para entrenar el árbol

```
>library(rpart)
```

creamos una semilla

```
> set.seed(135)
```

entrenamos el árbol

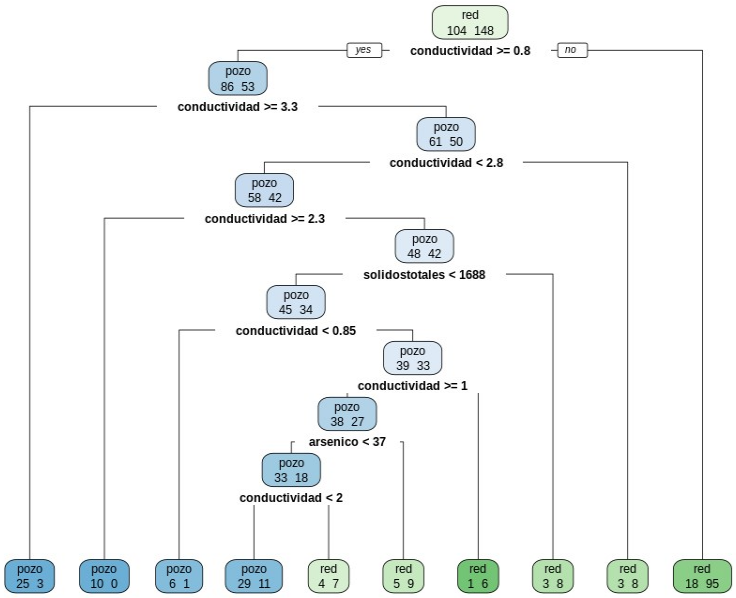
```
> ad5<-rpart(tipoagua~conductividad +solidostotales + arsenico, data=entrenamiento)
```

cargamos la biblioteca rpart.plot

```
>library(rpart.plot)
```

para graficar el árbol

```
> rpart.plot(ad5,extra=1)
```



Realizamos la prueba del árbol entrenado,

```
> pruebaad5<-predict(ad5,newdata=prueba,type="class")
```

y finalmente hallamos la matriz de confusión para lo cual utilizamos la biblioteca caret

```
> library(caret)

> matrizconfusion<-confusionMatrix(pruebaad5, prueba[["tipoagua"]])

> matrizconfusion
```

Con la que hallamos diversos valores de importancia, donde la Accuracy es nuestra principal información.

```
Confusion Matrix and Statistics

          Reference
Prediction pozo red
     pozo   32  16
     red    13  48

               Accuracy : 0.7339
```

podríamos pedir solo el valor de la exactitud con el comando

```
> as.numeric(matrizconfusion$overall[1])
```

[1] 0.733945

La importancia de la semilla

Como hemos visto, los resultados de los procedimientos de aprendizaje automatizado dependen en gran medida de la semilla establecida con la función set.seed(). La búsqueda de la mejor semilla a la hora de la preparación de los datos de prueba y entrenamiento es importante para mejorar la capacidad predictiva del modelo. Para optimizar este proceso podemos generar un script y correrlo con diferentes semillas. Si partimos de los datos pretratados a los que llamamos datosad5, podemos hacer diferentes particiones tomando diferentes semillas, luego entrenamos y probamos el modelo, para finalmente calcular la exactitud del mismo.

Recuerde que siempre necesitara las bibliotecas ya utilizadas, así que siempre para comenzar a trabajar cárguelas en su espacio de trabajo

```
> library(caTools)

> library(rpart)

> library(caret)

> library(rpart.plot)
```

Todos los pasos mencionados en el inciso anterior quedan resumidos en los siguientes pasos

```
> set.seed(135)

> split<-sample.split(datosad5$tipoagua,SplitRatio=0.7)

> entrenamiento<-subset(datosad5,split==1)

> prueba<-subset(datosad5,split==0)

> set.seed(135)

> ad5<-rpart(tipoagua~conductividad +solidostotales + arsenico, data=entrenamiento)

> pruebaad5<-predict(ad5,newdata=prueba,type="class")

> matrizconfusion<-confusionMatrix(pruebaad5, prueba[["tipoagua"]])

>as.numeric(matrizconfusion$overall[1])
```

transformamos las instrucciones en un script agregando algunos comandos que lo hagan más operativo. Mostramos el script en la columna de la izquierda, en la derecha las aclaraciones y en la primer fila el nombre del archivo

scriptsemilla	aclaración
print("introduzca la semilla que sea un número entero y oprima enter") semilla<-as.numeric(scan(file="",what="",nmax=1)) set.seed(semilla) writeLines(paste("semilla: ", semilla)) split<-sample.split(datosad5$tipoagua,SplitRatio=0.7) entrenamiento<-subset(datosad5,split==1) prueba<-subset(datosad5,split==0) ad5<-rpart(tipoagua~conductividad +solidostotales + arsenico, data=entrenamiento) pruebaad5<-predict(ad5,newdata=prueba,type="class") matrizconfusion<-confusionMatrix(pruebaad5, prueba[["tipoagua"]]) writeLines(paste("Accuracy: ",as.numeric(matrizconfusion$overall[1])))	pide en pantalla el valor de la semilla introducimos el valor y oprimimos enter fija el valor de la semilla con el valor introducido muestra en pantalla: semilla: el valor introducido fija el porcentaje de datos a cada grupo crea el set de entrenamiento crea el set de prueba entrena el árbol hace la prueba del arbol entrenado con el set de prueba crea la matriz de confusion nos muestra la exactitud como exactitud: valor hallado

Si bien a esta altura del curso ya debemos manejar scripts, no está de más recordarlo. Copie el script de la celda [2,1] de la tabla anterior y péguelo en un procesador de texto. Grabe el archivo con formato .txt o .R, en el mismo directorio en que se halla su espacio de trabajo y asígnele el nombre scriptsemilla

Luego desde el espacio de trabajo de R ejecute

```
> source('scriptsemilla')
```

le pedirá el valor de la semilla y le mostrará la exactitud. A modo de prueba mostramos algunos intentos. En el primer caso mostramos completa la salida del script

```
> source('scriptsemilla')
```

[1] "introduzca la semilla. Un número entero"
1: 135
Read 1 item
semilla: 135
Accuracy: 0.73394495412844
Como podemos observar obtuvimos la exactitud, con el mismo valor hallado anteriormente, ya que sembramos la semilla con el mismo valor. En los siguientes caso mostramos solo el valor de la semilla y la exactitud arrojada por el cálculo

```
> source('scriptsemilla')
```

semilla: 136
Accuracy: 0.761467889908257

```
> source('scriptsemilla')
```

semilla: 137
Accuracy: 0.743119266055046

```
> source('scriptsemilla')
```

semilla: 1245
Accuracy: 0.623853211009174

```
> source('scriptsemilla')
```

semilla: 68
Accuracy: 0.73394495412844

```
> source('scriptsemilla')
```

semilla: 140
Accuracy: 0.651376146788991

```
> source('scriptsemilla')
```

semilla: -135
Accuracy: 0.752293577981651
Como podemos ver, si bien en todos los casos la exactitud es buena, en algunos casos es mucho mejor, como por ejemplo con la semilla 136.
No se quede con el primer resultado, haga varias pruebas. Por supuesto, podría hacer un script con un bucle que barra valores de semillas desde un valor hasta un máximo, como se muestra a continuación

scriptsemillas	explicacion

print("introduzca el valor mínimo de la semilla. Un número entero")
minimo<-as.numeric(scan(file="",what="",nmax=1))
print("introduzca el valor máximo de la semilla. Un número entero")
maximo<-as.numeric(scan(file="",what="",nmax=1))
for(i in minimo:maximo){
set.seed(i)
writeLines(paste("semilla: ", i))
split<-sample.split(datosad5$tipoagua,SplitRatio=0.7)
entrenamiento<-subset(datosad5,split==1)
prueba<-subset(datosad5,split==0)
ad5<-rpart(tipoagua~conductividad +solidostotales + arsenico, data=entrenamiento)
pruebaad5<-predict(ad5,newdata=prueba,type="class")
matrizconfusion<-confusionMatrix(pruebaad5, prueba[["tipoagua"]])
writeLines(paste("Accuracy: ",as.numeric(matrizconfusion$overall[1])))
}
	nos pide el mínimo
	lo introducimos por teclado y oprimimos enter
	nos pide el máximo
	lo introducimos y oprimimos enter
	repite el proceso de entrenamiento y prueba desde el minimo hasta el máximo, mostrandonos al final de cada ciclo el valor de la semilla y la accuracy

Guardelo como scriptsemillas y ejecutelo. En el ejemplo utilizamos semillas entre 20 y 30

```
> source('scriptsemillas')
```

[1] "introduzca el valor mínimo de la semilla. Un número entero"
1: 20
Read 1 item
[1] "introduzca el valor máximo de la semilla. Un número entero"
1: 30
Read 1 item
semilla: 20
Accuracy: 0.715596330275229
semilla: 21
Accuracy: 0.642201834862385
semilla: 22
Accuracy: 0.669724770642202
semilla: 23
Accuracy: 0.651376146788991
semilla: 24
Accuracy: 0.743119266055046
semilla: 25
Accuracy: 0.63302752293578
semilla: 26
Accuracy: 0.669724770642202
semilla: 27
Accuracy: 0.724770642201835
semilla: 28
Accuracy: 0.724770642201835
semilla: 29
Accuracy: 0.715596330275229
semilla: 30
Accuracy: 0.669724770642202
Con la semilla 24 obtuvimos el mejor entrenamiento del modelo.
Podríamos ser más eficientes, haciendo que guarde semilla y exactitud en un data.frame y al finalizar nos muestre el trabajo realizado, donde podremos ver las diferentes semillas y exactitudes del proceso de entrenamiento-prueba

scriptmejorsemilla	

print("introduzca el valor mínimo de la semilla. Un número entero")
minimo<-as.numeric(scan(file="",what="",nmax=1))
print("introduzca el valor máximo de la semilla. Un número entero")
maximo<-as.numeric(scan(file="",what="",nmax=1))
mejordato<-data.frame(semilla=NA,exactitud=NA)
for(i in minimo:maximo){
set.seed(i)
split<-sample.split(datosad5$tipoagua,SplitRatio=0.7)
entrenamiento<-subset(datosad5,split==1)
prueba<-subset(datosad5,split==0)
ad5<-rpart(tipoagua~conductividad +solidostotales + arsenico, data=entrenamiento)
pruebaad5<-predict(ad5,newdata=prueba,type="class")
matrizconfusion<-confusionMatrix(pruebaad5, prueba[["tipoagua"]])
mejordato[i,1]=i
mejordato[i,2]=as.numeric(matrizconfusion$overall[1])
}
print(na.omit(mejordato))
	pide valor mínimo de semilla
	lo introducimos
	pide el valor máximo de la semilla
	lo introducimos
	crea un data.frame con dos columnas: semilla y exactitud
	realiza un bucle de entrenamiento-prueba desde el valor mínimo al máximo establecido
	
	carga la semilla en la primer columna del data.frame
	carga la exactitud en la segunda columna del data.frame
	
	elimina filas con NA y nos muestra el resultado

Cree el archivo de texto con el script de la tabla anterior y grábelo como scriptmejorsemilla. Luego ejecute

```
> source('scriptmejorsemilla')
```

a modo de ejemplo mostramos lo que obtendría si utiliza semillas: 10 a 15

	semilla	exactitud
10	10	0.6697248
11	11	0.7155963
12	12	0.7247706
13	13	0.6972477
14	14	0.5871560
15	15	0.6788991

Queda claro que la mejor elección sería la 12.
Por último optimicemos más el proceso de manera que nos muestre solo el mejor valor de semilla y la exactitud alcanzada.

scriptmejorsemillaoptimizado	explicación

print("introduzca el valor mínimo de la semilla. Un número entero")
minimo<-as.numeric(scan(file="",what="",nmax=1))
print("introduzca el valor máximo de la semilla. Un número entero")
maximo<-as.numeric(scan(file="",what="",nmax=1))
mejordato<-data.frame(semilla=NA,exactitud=NA)
for(i in minimo:maximo){
set.seed(i)
split<-sample.split(datosad5$tipoagua,SplitRatio=0.7)
entrenamiento<-subset(datosad5,split==1)
prueba<-subset(datosad5,split==0)
ad5<-rpart(tipoagua~conductividad +solidostotales + arsenico, data=entrenamiento)
pruebaad5<-predict(ad5,newdata=prueba,type="class")
matrizconfusion<-confusionMatrix(pruebaad5, prueba[["tipoagua"]])
mejordato[i,1]=i
mejordato[i,2]=as.numeric(matrizconfusion$overall[1])
}
mejordato<-na.omit(mejordato)
print(mejordato[mejordato$exactitud==max(mejordato$exactitud),])
	muestra el valor de la semilla para la exactitud máxima

Cree un archivo de texto con el código de la tabla anterior, grábelo en el directorio de trabajo donde se halla su espacio de trabajo con el nombre scriptmejorsemillaoptimizado y luego ejecute

```
> source('scriptmejorsemillaoptimizado')
```

[1] "introduzca el valor mínimo de la semilla. Un número entero"
Se muestra resultado para semillas entre 10 y 15
1: 10
Read 1 item
[1] "introduzca el valor máximo de la semilla. Un número entero"
1: 15
Read 1 item

	semilla	exactitud
12	12	0.7247706

Viendo la rapidez con que se realiza el proceso. Por qué no probar con semillas entre 1 y 20000. Le llevará un poco más de tiempo, quizás el necesario para preparar algo para tomar y descansar!

```
> source('scriptmejorsemillaoptimizado')
```

	semilla	exactitud
580	580	0.8440367

Una exactitud del 84.4% no está nada mal.
Se podrá preguntar en este punto: ¿Pero el aprendizaje es diferente, según de donde parto?. La respuesta es SI. Acaso no aprendemos de manera diferente, según sea el profesor que nos tocó en la clase, o los libros que seleccionamos para comenzar a estudiar un tema?

Recuerde que el sistema aprende aunque usted no esté presente. Puede diseñar un script exigente y dejarlo funcionando cuando usted comienza a descansar. A su regreso tendrá el análisis completo y como si fuera poco, el sistema no le recriminará nada!!! Seguramente si fuese al revés pondríamos el grito en el cielo y seguramente organizaríamos una protesta!

Otras interpretaciones de rpart.plot

Realizaremos la grafica del objeto ad5 obtenido con el último entrenamiento. Utilizaremos la semilla 580 hallada. Hacemos nuevamente la partición y el entrenamiento, por las dudas haya cambiado algo durante el estudio

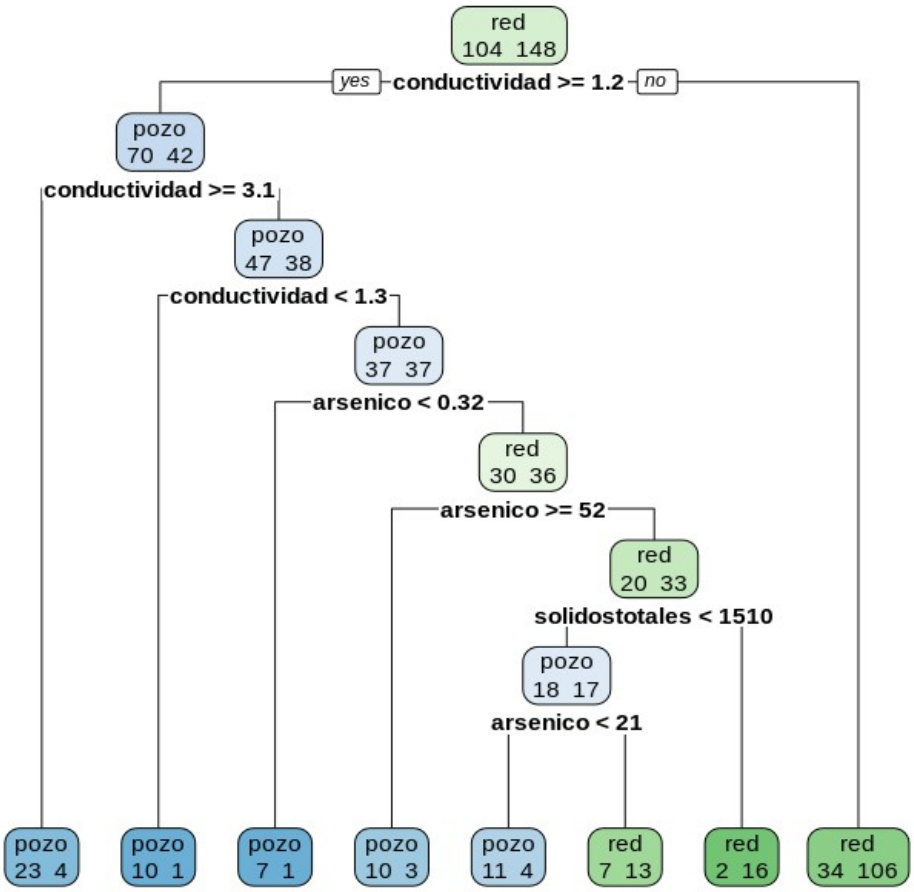
```
> set.seed(580)

> split<-sample.split(datosad5$tipoagua,SplitRatio=0.7)

> entrenamiento<-subset(datosad5,split==1)

> prueba<-subset(datosad5,split==0)

> ad5<-rpart(tipoagua~conductividad +solidostotales + arsenico, data=entrenamiento)

> pruebaad5<-predict(ad5,newdata=prueba,type="class")

> matrizconfusion<-confusionMatrix(pruebaad5, prueba[["tipoagua"]])

> as.numeric(matrizconfusion$overall[1])
```

verificamos que todo está funcionando como esperamos ya que la exactitud arrojada fue de
[1] 0.8440367

Todos los análisis posteriores se harán sobre el objeto ad5. Recuerde que si en el transcurso de su estudio hiciera algún entrenamiento con otra semilla, no hallaría los mismos gráficos que se muestran a continuación. Para realizar los gráficos utilizamos la función rpart.plot() con los siguientes argumentos

```
> rpart.plot(ad5,type=2,extra=1)
```

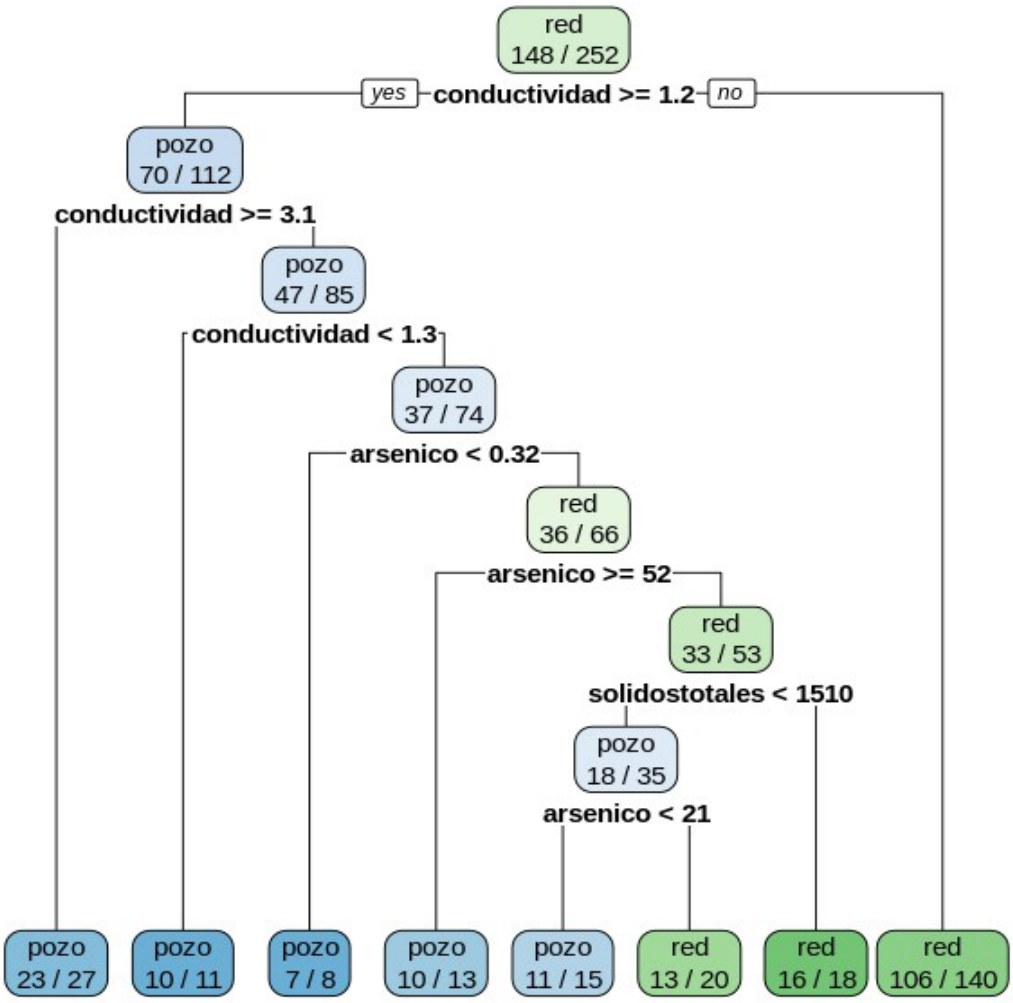
obtenemos



Donde el color azul representa nodos con predominio de agua de pozo y los de color verde con predominio de red. A su vez en la segunda línea de cada nodo vemos el número de muestras de pozo a la izquierda y de red a la derecha.
El argumento type, que por defecto toma el valor 2, nos permite ver el árbol de diferentes maneras. type puede variar entre 0 y 5. Puede probar diferentes opciones para elegir la que le permita una mejor interpretación. En general cambian en la forma de presentar los criterios, pero verá que no cambian los nodos terminales.
El argumento extra, tiene 11 valores posibles que nos mostrarán diferente información. La gráfica obtenido fue obtenida con extra=1, que quizás es la forma más directa de interpretar.
si utilizamos el valor extra=2

```
> rpart.plot(ad5,type=2,extra=2)
```

muestra en cada nodo una razón entre el número de unidades de la clase predominante sobre el número total de unidades del nodo. Para el caso que nos compete la gráfica se presenta de la siguiente manera



Si analizamos él último nodo de la derecha, dentro del recuadro figura red 106/140, es decir predominan aguas de red siendo 106 de esta característica de un total de 140 que forman el nodo. Puede comprobar que en la anterior presentación decía 34 106, indicando aguas de pozo y red, respectivamente.

Si utilizamos el argumento extra=3

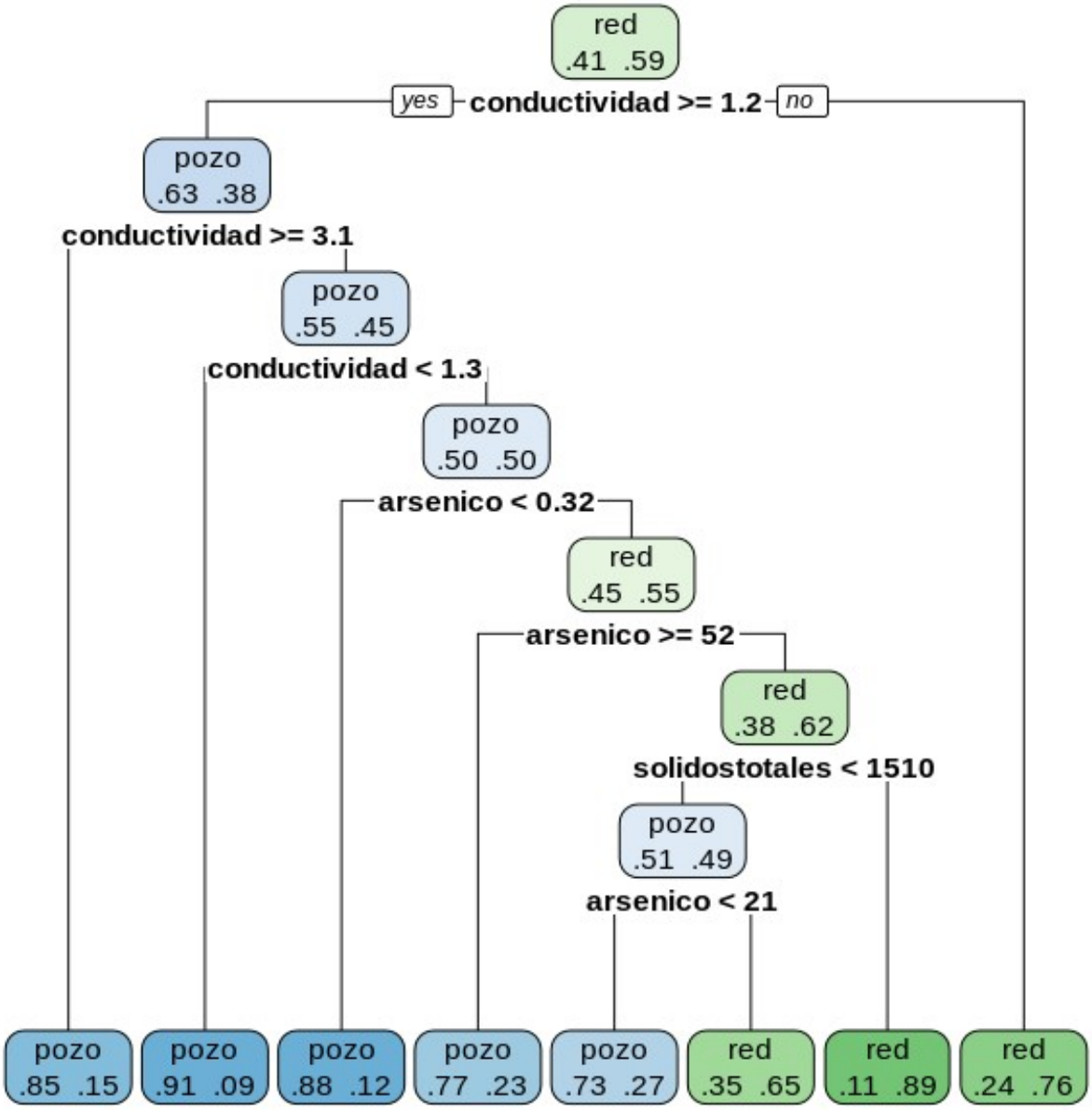
```
> rpart.plot(ad5,type=2,extra=3)
```

verá que es similar, salvo que en lugar de mostrar la razón entre el número de datos predominantes y datos totales del nodo, muestra la razón entre las unidades que no predominan y el total. El último nodo de la derecha tiene la razón 34/140: 34 de pozo de un total de 140 muestras.

Con

```
> rpart.plot(ad5,type=2,extra=4)
```

muestra la fracción de cada tipo de agua, o el porcentaje si lo multiplicamos por 100. Para el último nodo terminal de la derecha observa que 24% son aguas de pozo y 76% aguas de red.

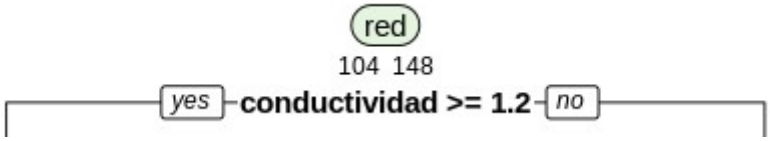


puede probar otros valores del argumento extra, pero quizás con los analizados es más que suficiente.

El argumento under, que puede ser TRUE o FALSE, solo es una cuestión estética de ubicación de los números. Puede comparar los dos códigos

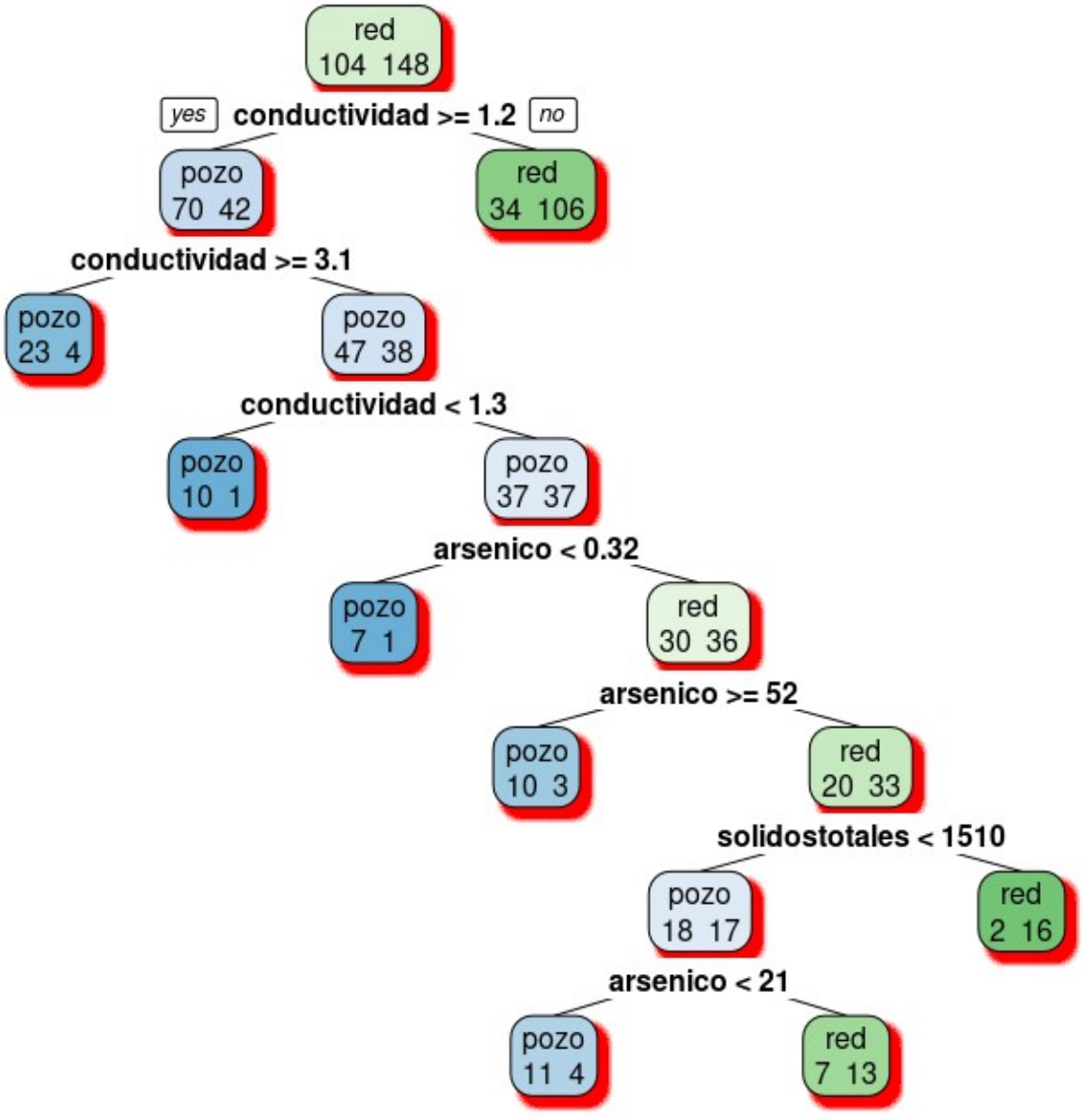
```
> rpart.plot(ad5,type=2,extra=1,under=FALSE)
```


```
> rpart.plot(ad5,type=2,extra=1,under=TRUE)
```



otros argumentos menos importantes como shadow.col y fallen.leaves, permiten algunas modificaciones estéticas.

```
> rpart.plot(ad5,type=2,extra=1,under=FALSE,fallen.leaves=FALSE,shadow.col="red")
```



Otra forma de visualizar los datos, que es menos intuitiva, pero que da la misma información es llamando el objeto creado en el entrenamiento, en este caso ad5.

En esta visualización primero vemos n=252, es el número de unidades de entrenamiento. Luego vemos diferentes líneas, que comienzan con el número del nodo, sigue con el criterio, el número total de unidades del nodo y luego el número de unidades de la categoría minoritaria. Luego la categoría predominantes y entre paréntesis los porcentajes de las categorías en orden alfabético.

```
> ad5
```

```
n= 252

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 252 104 red (0.41269841 0.58730159)
    2) conductividad>=1.22125 112  42 pozo (0.62500000 0.37500000)
      4) conductividad>=3.095 27   4 pozo (0.85185185 0.14814815) *
      5) conductividad< 3.095 85  38 pozo (0.55294118 0.44705882)
       10) conductividad< 1.3225 11   1 pozo (0.90909091 0.09090909) *
       11) conductividad>=1.3225 74  37 pozo (0.50000000 0.50000000)
         22) arsenico< 0.3186419 8   1 pozo (0.87500000 0.12500000) *
         23) arsenico>=0.3186419 66  30 red (0.45454545 0.54545455)
           46) arsenico>=52.35493 13   3 pozo (0.76923077 0.23076923) *
           47) arsenico< 52.35493 53  20 red (0.37735849 0.62264151)
             94) solidostotales< 1510 35  17 pozo (0.51428571 0.48571429)
              188) arsenico< 20.55911 15   4 pozo (0.73333333 0.26666667) *
              189) arsenico>=20.55911 20   7 red (0.35000000 0.65000000) *
             95) solidostotales>=1510 18   2 red (0.11111111 0.88888889) *
    3) conductividad< 1.22125 140  34 red (0.24285714 0.75714286) *
```

Como puede ver las líneas se van indentando. La tabulación indica a partir de que nodo se produjo la división. Por ejemplo partiendo del nodo 1, es la raíz del árbol con 252, se partió en dos nodos (2 y 3). El nodo 2, corresponde a aquellos valores de conductividad mayor o igual a 1.22, tiene 112 unidades en total, con 70 unidades de pozo. El otro nodo, el 3 son las unidades de conductividad menor a 1.22. El nodo 2, se separa en los módulo 4 y 5, utilizando como criterio el valor conductividad, generando un nodo intermedio (nº 5) y uno terminal (nº 4). Los nodos terminales queda identificados por un asterisco al final de la línea.

Módulo 8. Clase 6

Random Forest

La metodología Random Forest dentro del aprendizaje automatizado forma parte de lo que se conoce como el aprendizaje ensamblado, en que más de un predictor es tenido en cuenta para el proceso de entrenamiento. Básicamente, random forest consiste en entrenar un gran número de árboles de decisión y luego hacer la elección por aquellos de mejor rendimiento.
En estas metodologías, se pueden realizar estudios de clasificación o regresión. El primer caso es cuando la variable respuesta es cualitativa y tiene diversos niveles de un factor. En el segundo caso la variable respuesta es una variable numérica continua.
Para la resolución de estos modelos utilizamos la biblioteca randomForest, que puede descargar e instalar como lo ha hecho a lo largo de todo el curso. Utilizaremos también otras bibliotecas que indicaremos durante el desarrollo del módulo. Como en todos los procesos de aprendizaje automatizado tenemos un grupo de entrenamiento y un grupo de prueba.
La fórmula general para entrenar Random Forest es

```
> RandomForest(formula, data, ntree=n, mtry=m, importance=TRUE/FALSE, maxnodes=NULL)
```

Detalle de los argumentos:
- formula: es el modelo a ajustar. Veremos que podemos seleccionar algunas variables predictoras o bien utilizar todas las variables independientes.
- ntree: es el número de árboles utilizados.
- mtry: el número de variables independientes seleccionadas por ciclo de entrenamiento. Por defecto, para procesos de clasificación este número es la raíz cuadrada del número de variable predictoras y para regresión, es el número de variables predictoras/3. Si este parámetro se deja con el valor NULL, la función RandomForest elige automáticamente uno u otro valor.
- maxnodes: número máximo de nodos terminales. Por defecto no tiene valor.
- importance=TRUE. Si la importancia de cada variable independiente debe ser evaluada

Necesitaremos la biblioteca randomForest, instálela en su computadora y cárguela al espacio de trabajo

```
> library(randomForest)
```

Preparación de los datos

Los datos que utilizaremos son los mismos de las clases de los módulos anteriores. Por cuestiones de orden sugerimos iniciar un nuevo espacio de trabajo para esta clase e introducir los datos nuevamente. Los datos se halla en el archivo datosR86.RData. A continuación y de una manera más resumida realizamos el procesamiento de los datos para aplicar luego las funciones correspondientes.
Introduzca los datos en su espacio de trabajo.

```
> load("datosR86.RData")
```

Los datos ingresan como un data.frame llamado datosaguas. Compruébelo

```
> ls()
```

[1] "datosaguas"
Estos datos tienen una gran cantidad de variables de componentes medidos y sus errores

```
> names(datosaguas)
```

```
 [1] "codigo"        "fecha"         "localidad"
 [4] "provincia"     "tipoagua"      "responsble"
 [7] "ph"            "cvph"          "conductividad"
```

```
[10] "cvconductividad"   "cloruro"            "cvcloruro"
[13] "carbonato"         "cvcarb"             "bicarbonato"
[16] "cvbicar"           "solidostotales"     "cvsolidostotales"
[19] "fosfato"           "cvfosfato"          "nitrato"
[22] "cvnitrato"         "nitrito"            "cvnitrito"
[25] "fluoruro"          "cvfluoruro"         "arsenico"
[28] "cvarsenico"        "amonio"             "cvamonio"
[31] "observaciones"     "alcalinidadtotal"   "CValcalinidadtotal"
[34] "consumo"           "sulfato"            "cvsulfato"
[37] "informado"         "sodio"              "cvsodio"
[40] "dqo"               "cvdqo"              "tkn"
[43] "cvtkn"             "calcio"             "cvcalcio"
[46] "durezatotal"       "cvdurezatotal"      "duda"
[49] "numeropersonas"    "orp"                "cvorp"
[52] "magnesio"          "cvmagnesio"         "servicio"
[55] "materiaorganica"   "litio"              "cvlitio"
[58] "arsenicoIII"       "cvarsenicoIII"      "iodo"
[61] "cviodo"            "especiescloro"      "cvespeciescloro"
[64] "muestraagotada"    "manganeso"          "cvmanganeso"
[67] "estroncio"         "cvestroncio"        "plomo"
[70] "cvplomo"           "turbidez"           "cvturbidez"
```

Selección de las variables de interés. Como explicamos en la clase anterior nos quedamos con algunas variables independientes, que corresponden a mediciones con buena precisión y exactitud, dejando de lado otras variables que figuran en la base de datos, pero que no están directamente relacionadas a calidad del agua. Asignamos esta selección a un nuevo objeto: datosrf6, nombre que hace referencia al tema: rf-randomforest y 6 por el número de clase. Todo es una cuestión de ordenamiento. Si lo desea puede dar otros nombres a los objetos

```
> datosrf6<-datosaguas[,c(5,7,9,11,17,19,25,27,29,32,35,38,46)]
```

que resulta en los siguientes campos

```
> names(datosrf6)
```

```
 [1] "tipoagua"          "ph"                "conductividad"    "cloruro"
 [5] "solidostotales"    "fosfato"           "fluoruro"         "arsenico"
 [9] "amonio"            "alcalinidadtotal"  "sulfato"          "sodio"
[13] "durezatotal"
```

el data.frame tiene 425 registros, como podemos comprobar

```
> nrow(datosrf6)
```

```
[1] 425
```

Eliminamos filas que tengan datos NA.

```
> datosrf6<-datosrf6[is.na(datosrf6$cloruro)==FALSE & is.na(datosrf6$solidostotales)==FALSE &
is.na(datosrf6$fosfato)==FALSE & is.na(datosrf6$fluoruro)==FALSE &
is.na(datosrf6$arsenico)==FALSE & is.na(datosrf6$sulfato)==FALSE &
is.na(datosrf6$sodio)==FALSE & is.na(datosrf6$durezatotal)==FALSE,]
```

y seleccionamos solo aquellos registros en el tipo de agua tiene el valor: red o pozo. Esta selección tiene el fin de hacer más fácil la comprensión de la salida de los análisis, que en este caso será de clasificación

```
> datosrf6<-datosrf6[datosrf6$tipoagua=="pozo"| datosrf6$tipoagua=="red",]
```

Eliminamos niveles sin datos del campo tipo agua

```
> datosrf6$tipoagua<-droplevels(datosrf6$tipoagua)
```

Deben quedar 361 registros, verifíquelo

```
> nrow(datosrf6)
```

[1]361

A continuación preparamos los grupos de entrenamiento y prueba dejando un 90% de datos en el grupo de entrenamiento y para el proceso utilizamos la biblioteca caTools.

```
> library(caTools)
```

sembramos una semilla. Recuerde que pueden hallarse valores de semillas más favorables para entrenar el modelo

```
> set.seed(580)
```

y separamos los datos en dos grupos: entrenamiento y prueba con el porcentaje acordado recientemente

```
> split<-sample.split(datosrf6$tipoagua,SplitRatio=0.9)

> entrenamiento<-subset(datosrf6,split==1)

> prueba<-subset(datosrf6,split==0)
```

Realizamos las verificaciones correspondientes para evaluar que todo esté en orden. Salvo la variable tipoagua que es un factor con dos niveles, el resto de las variables son numéricas continuas. Verifique esto con la función summary(), en los grupos de entrenamiento y prueba. La función str() también es muy útil para ver esto. Veamos para el grupo de datos de entrenamiento

```
> str(entrenamiento)
```

```
'data.frame':    325 obs. of  13 variables:
 $ tipoagua          : Factor w/ 2 levels "pozo","red": 1 2 1 2 2 2 2 1 2 2 ...
 $ ph                : num  7.76 7.38 7.21 6.58 6.64 7.07 6.77 7.14 7.45 6.2 ...
 $ conductividad     : num  1.305 1.81 1.777 0.637 0.285 ...
 $ cloruro           : num  27.87 57.03 55.61 7.86 18.29 ...
 $ solidostotales    : num  400 1550 1685 640 380 ...
 $ fosfato           : num  0 1.26 1.52 0.68 0 ...
 $ fluoruro          : num  0.725 0.968 0.307 1.216 0.522 ...
 $ arsenico          : num  33.96 27.09 66.42 58.19 3.66 ...
 $ amonio            : num  0.655 0 1.866 0.482 2.299 ...
 $ alcalinidadtotal : num  531.1 599.3 756.8 327.8 65.3 ...
 $ sulfato           : num  102.9 224.4 89.1 17.7 27.5 ...
 $ sodio             : num  247.6 423.5 437.8 130 38.1 ...
 $ durezatotal       : num  83.7 63 54 51.2 71.1 ...
```

y para prueba

```
> str(prueba)
```

```
'data.frame':    36 obs. of  13 variables:
 $ tipoagua          : Factor w/ 2 levels "pozo","red": 2 2 2 2 2 2 1 2 1 2 ...
 $ ph                : num  6.25 7.18 6.98 7.24 7.91 8.07 7.39 8.38 6.76 8.43 ...
 $ conductividad     : num  0.705 0.698 1.105 0.03 0.49 ...
 $ cloruro           : num  19.07 30.09 95.72 5.73 27.93 ...
 $ solidostotales    : num  1040 1340 965 675 270 ...
 $ fosfato           : num  0.492 0 0 0 0 ...
```

```
 $ fluoruro           : num  0.784 1.554 0.494 0.105 0.108 ...
 $ arsenico           : num  18.14 34.57 32.04 0 3.45 ...
 $ amonio             : num  1.06 1.4 2.57 1.54 0 ...
 $ alcalinidadtotal   : num  467.1 398.7 489.2 31.9 66.6 ...
 $ sulfato            : num  9.69 32.87 35.95 9.04 87.69 ...
 $ sodio              : num  123.93 181.49 195.14 0 7.58 ...
 $ durezatotal        : num  87.9 128.2 83.2 32.8 121 ...
```

Entrenamiento del modelo

A continuación realizamos el entrenamiento con ciertos valores de parámetros. En todos los ejemplos la variable dependiente será tipoagua, con sus niveles pozo y red. En este caso en particular utilizaremos todas las variables predictoras. Esto queda representado en la fórmula del modelo, que como se ve a continuación se escribe: tipoagua~. No especificando ninguna variable independiente.

```
> set.seed(1400)

> entrenamientoRF<-randomForest(tipoagua~.,data=entrenamiento,ntree=1000,importance=TRUE)
```

podemos ver diferentes datos del proceso de entrenamiento, que han quedado en el objeto entrenamientoRF. El siguiente comando nos da la función utilizada y sus argumentos.

```
> entrenamientoRF$call
```

randomForest(formula = tipoagua ~ ., data = entrenamiento, ntree = 1000, importance = TRUE)

Podemos ver también si el entrenamiento se hizo como regresión o clasificación. La función randomForest hace esta elección automáticamente. Si la variable dependiente es un factor, será clasificación.

```
> entrenamientoRF$type
```

[1] "classification"

Si deseamos ver los valores medidos de la variable dependiente en los datos de entrenmiento aplicamos el código.

```
> entrenamientoRF$y
```

```
   2     3     4     5     6     7     8     9     10   11    12   13    14    15    17    18
pozo  red pozo  red  red  red  red pozo  red  red  red pozo pozo  red pozo pozo
  19   20    21   22   25   26   27   29   30   31   34   35   36   39   40   41
 red  pozo  red  .....
```

Como puede observar, los números de las unidades no son correlativos, debido a que el grupo de entrenamiento no tiene todas las muestras y la selección de las unidades del grupo se hizo al azar utilizando la función sample.split().

Si deseamos ver los valores que el modelo ha predicho para la variable dependiente utilizamos el código

```
> entrenamientoRF$predicted
```

```
   2     3     4     5     6     7     8     9     10     11    12   13   14    15     17     18
pozo pozo pozo  red  red pozo  red pozo pozo  red  red  red pozo pozo pozo  red
  19   20   21   22   25   26   27   29   30   31   34   35   36   39   40   41
 red  red  red  ..................................................
```

Si desea conocer el número de árboles utilizados en el proceso de entrenamiento ejecute el código

```
> entrenamientoRF$ntree
```

[1] 1000

Da el número de árboles utilizados, parámetro que hemos fijado al aplicar la función randomForest(). Si no se fija este valor por defecto toma 500. Con sus datos podrá hacer pruebas buscando el mejor valor de este parámetro. La mejor elección la hará mirando siempre la exactitud que surge de la matriz de confusión con los datos de prueba.

El código siguiente da el número de variables independientes considerados para cada proceso de entrenamiento.

```
> entrenamientoRF$mtry
```

[1] 3

da el número de variables que se toman por vez, como se mencionó anteriormente, si no se especifica este valor entre los argumentos de la función randomForest(), la función adjudica automáticamente la raíz cuadrada del número de variables independientes para el caso de clasificación. Como estás son 12 en el grupo de entrenamiento, se han tomando 3 de ellas. Esto no implica que solo se utilizan 3, sino que el algoritmo va tomando combinación de 3 de ellas.

Podemos conocer las clases, es decir los niveles de la variable dependiente. Para ello utilizamos el siguiente código.

```
> entrenamientoRF$classes
```

[1] "pozo" "red"

Por defecto se llama clase positiva a la primer, que surge del ordenamiento alfabético. red: sería la clase negativa. Estos términos son importantes para analizar un tema que trataremos inmediatamente.

Para obtener la matriz de confusión generada por randomForest, utilice el código siguiente. También podemos utilizar la función de la biblioteca caret, que da además la exactitud. Pero es más útil para el proceso de prueba.

```
> entrenamientoRF$confusion
```

	pozo	red	class.error
pozo	95	39	0.2910448
red	35	156	0.1832461

Esta es la matriz de confusión utilizando los valores reales de tipoagua en el grupo de entrenamiento y los valores predichos por el modelo. Básicamente sale de los datos que hemos visto anteriormente. Es importante recordar que la matriz de confusión es una tabla que indica el número de unidades experimentales clasificada en base a dos criterios, en este caso los valores medidos y los predichos. Otro punto a remarcar es que la matriz de confusión de más valor es la que hacemos con los valores medidos y los predichos por el modelo, pero utilizando el grupo de prueba. En este caso estamos utilizando el grupo de entrenamiento. Sin embargo observar esta matriz, no es una mala práctica y nos irá mostrando la bondad de nuestro modelo.

Vemos también una columna class.error. En la primer línea observamos el valor 0.2910448. Es decir un 29,1% de error. Lo interpretamos así: el modelo predijo que de hay (95+39)= 134 aguas de pozo. Pero como podemos ver, en 39 de ellas se equivocó, ya que esas 39 aguas en realidad son de red. Entonces se equivocó en 39/134= 0.2910448. De la misma manera, el modelo predijo que (35+156)=191 aguas son de red, pero se equivocó en 35. Asi, el error de la clase fue 35/191= 0.1832461, es decir en 18.3% de los casos.

A fines de consolidar más aun el concepto de matriz de confusión, veamos como construirla a partir de nuestros datos. Este ejercicio además me refrescará algunas funciones muy útiles en cualquier tema. Como mencionamos los datos medidos de tipo agua se encuentran en

```
> entrenamientoRF$y
```

y los predichos en

```
> entrenamientoRF$predicted
```

Con los dos datos anteriores, creamos un objeto mc, forzando los datos de los objetos anteriores a que sean data.frame

```
> mc<-cbind(as.data.frame(entrenamientoRF$y),as.data.frame(entrenamientoRF$predicted))
```

para más claridad, cambiamos los nombres de las dos columnas del data.frame mc

```
> names(mc)<-c("referencia","predicho")
```

podemos ver los datos

```
> head(mc)
```

```
  referencia predicho
2      pozo     pozo
3       red     pozo
4      pozo     pozo
5       red      red
6       red      red
7       red     pozo
```

Analicemos: la primer línea corresponde al agua número 2, es una muestra de pozo (clase positiva) y el modelo predijo que es de pozo (clase positiva). Es decir es un verdadero positivo. La línea siguiente, agua número 3 es una muestra de red (clase negativa), sin embargo el modelo predijo que es de pozo (clase positiva). Entonces lo computamos como un falso positivo. En este caso en particular, no tiene mucho interés ni claridad hablar de clase positiva y negativa. Pero imagine otra situación donde la variable dependiente es el diagnóstico de una enfermedad, con dos niveles: enfermo - sano. Enfermo sería la clase positiva y sano la negativa y allí tomaría mucho más sentido.
Ahora hagamos la matriz de confusión y para ello pedimos que nos clasifique a las unidades según los valores de ambas columnas

```
> table(mc$referencia,mc$predicho)
```

```
      pozo red
  pozo   95  39
  red    35 156
```

que podemos ver es el mismo resultado que el hallado con el código utilizado anterioremente

```
> entrenamientoRF$confusion
```

podemos calcular la exactitud

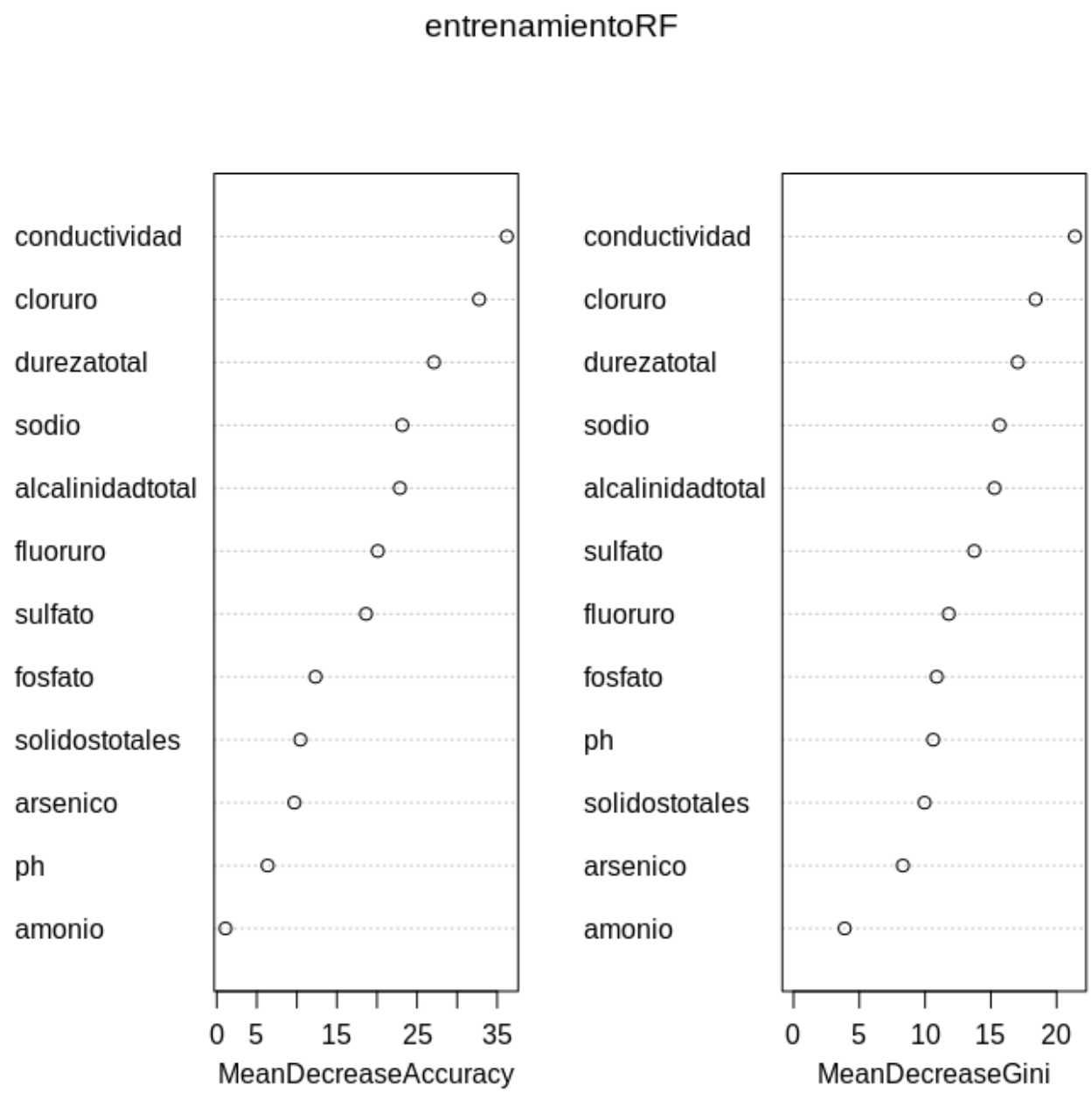
```
> (95+156)*100/(95+39+35+156)
```

```
[1] 77.23077
```

Para ver la importancia que tiene cada variable en el proceso de entrenamiento, podemos utilizar el siguiente código.

```
> varImpPlot(entrenamientoRF)
```

Esta función nos muestra la importancia de cada variable sobre el modelo ordenado de mayor a menor. Se observan dos índices MeanDecreaseAccuracy y MeanDecreaseGini. En un modo práctico interpretamos que cuanto mayor es el valor de MeanDecreaseAccuracy o MeanDecreaseGini, más importancia tiene la variable en el modelo. En el caso que estamos analizando podemos decir que la variable que más importancia tiene en el modelo es la conductividad, seguida por el cloruro y la durezatotal. Estos datos los podemos ver de forma gráfica o analítica. El código anterior lo hace forma gráfica.



Vemos que los dos índices coinciden en las variables de más peso, sin embargo alguna discrepancia aparece en variables de menos peso. Volveremos sobre el tema en la clase siguiente.
Si asignamos el valor de la función anterior a un objeto

```
> importanciavariables<-varImpPlot(entrenamientoRF)
```

en el objeto creado tenemos los valores de ambos índices

```
> importanciavariables
```

	MeanDecreaseAccuracy	MeanDecreaseGini
ph	6.343631	10.617853
conductividad	36.214410	21.397593
cloruro	32.752928	18.405749
solidostotales	10.437762	9.960798
fosfato	12.326473	10.883832
fluoruro	20.101680	11.803994
arsenico	9.694042	8.312983

amonio	1.086588	3.887648
alcalinidadtotal	22.859427	15.277805
sulfato	18.633528	13.738825
sodio	23.157702	15.663890
durezatotal	27.112653	17.047498

Testeo del modelo

Finalmente podemos testear el modelo con el grupo prueba

```
> pruebaRF<-predict(entrenamientoRF,prueba)
```

para analizar la matriz de confusión utilizamos la biblioteca caret

```
> library(caret)

> matrizconfusion<-confusionMatrix(pruebaRF,prueba$tipoagua)

> matrizconfusion
```

```
Confusion Matrix and Statistics
          Reference
Prediction pozo red
     pozo   13   1
     red     2  20

               Accuracy : 0.9167
                 95% CI : (0.7753, 0.9825)
    No Information Rate : 0.5833
    P-Value [Acc > NIR] : 1.104e-05

                  Kappa : 0.8269
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.8667
            Specificity : 0.9524
         Pos Pred Value : 0.9286
         Neg Pred Value : 0.9091
             Prevalence : 0.4167
         Detection Rate : 0.3611
   Detection Prevalence : 0.3889
      Balanced Accuracy : 0.9095

       'Positive' Class : pozo
```

Analicemos algunos términos que nos muestra esta salida. Si lo recuerda, muchos de los términos fueron analizados al tratar el tema de curvas ROC, en el módulo 4.
Accuracy, es la exactitud y es el cociente entre la suma de verdaderos positivos y negativos por el número total de casos. Es decir en que porcentaje el modelo predijo valores correctos. En este caso 91.67%. Podríamos decir que es muy bueno.
95% CI: nos da el intervalo de confianza de la exactitud. El límite superior muy cercano a 1 y el inferior lejos de 0.5, son buenos indicadores.
Sensitivity o sensibilidad: son cuantos positivos detecta el modelo (13) sobre el total de realmente positivos (13+2). Esto nos da: 13/(13+2) = 0.8667

Specificity o especificidad, nos indica que capacidad tiene el test de solo dar positivos con la clase positivo. Es decir que la especificidad será alta si el número de falsos positivos es bajo. En este caso los falsos positivos toma valor: 1. La especificidad la calculamos como la cantidad de verdaderos negativos predichos por el modelo (20), dividido todos los negativos (1+20). Así obtenemos; 20/(1+20)= 0.9523.
Pos Pred Value o valor predictivo positivo: indica que capacidad tienen nuestro modelo de predecir un valor positivo. Se calcula como el número de verdaderos positivos detectados por el modelo: (13), sobre el total de positivos predichos (13+1). Obtenemos el valor: 0.9286. Es decir que cuando el modelo predice que un agua es de pozo (la clase positiva) tenemos un 92.86% de probabilidad que así lo sea!
Neg pred Value o valor predictivo negativo: indica que capacidad tiene el modelo de predecir un valor de la clase negativa. Se calcula como el cociente de los verdaderos negativos (recuerde que la clase negativa es red), en este caso 20, dividido el total de valores que predijo en la clase negativa, en nuestro caso (2 + 20). Entonces obtenemos: 0.9091.
Prevalence o prevalencia: es el cociente el número de unidades que realmente pertenecen a la clase positiva (13+2) y el numero total de datos (36). Esto nos da el valor de 0.4166. Es decir que en las muestras la prevalencia de aguas de pozo (la clase positiva) es del 41.66%.
Deteccion Prevalence o prevalencia predicha por el modelo, es el cociente entre el número de clase positiva predicha por el modelo (13+1), sobre el total de datos: 36, que arroja el valor: 0.3889, es decir un 38,89%.
Deteccion Rate o razón de detección de positivos: es el número de verdaderos positivos (13) dividido por el total de datos de prueba (36). Obtenemos el valor 13/36=0.3611
Influencia del parámetro ntree en el entrenamiento

A continuación evaluaremos la influencia del valor asignado al parámetro ntree, sobre el proceso de entrenamiento.
ntree=500

Utilizaremos los mismos grupos de entrenamiento y prueba, pero modificaremos el parámetro ntree. Comenzaremos por el valor por defecto, que es 500

```
>set.seed(1400)

> entrenamientoRF<-randomForest(tipoagua~.,data=entrenamiento,ntree=500)

> pruebaRF<-predict(entrenamientoRF,prueba)

> matrizconfusion<-confusionMatrix(pruebaRF,prueba$tipoagua)

> matrizconfusion
```

Confusion Matrix and Statistics

```
          Reference
Prediction pozo red
     pozo    13   1
     red      2  20

               Accuracy : 0.9167
                 95% CI : (0.7753, 0.9825)
    No Information Rate : 0.5833
    P-Value [Acc > NIR] : 1.104e-05
```

```
                    Kappa : 0.8269
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.8667
            Specificity : 0.9524
         Pos Pred Value : 0.9286
         Neg Pred Value : 0.9091
             Prevalence : 0.4167
         Detection Rate : 0.3611
   Detection Prevalence : 0.3889
      Balanced Accuracy : 0.9095

       'Positive' Class : pozo
```

Obtuvimos una exactitud de 91.67%. Probemos ahora otro valor de ntree

ntree=200

Veamos ahora con menos cantidad. Utilizamos 200, aunque no es recomendable. Se recomienda que este número no sea muy pequeño

```
> set.seed(1400)

> entrenamientoRF<-randomForest(tipoagua~.,data=entrenamiento,ntree=200)

> pruebaRF<-predict(entrenamientoRF,prueba)

> matrizconfusion<-confusionMatrix(pruebaRF,prueba$tipoagua)

> matrizconfusion
```

Confusion Matrix and Statistics

```
          Reference
Prediction pozo red
     pozo    13   2
     red      2  19

          Accuracy : 0.8889
```

Aunque la exactitud no ha sido mala, es peor que con ntree=500. Veamos con ntree=2000

ntree=2000

```
> set.seed(1400)

> entrenamientoRF<-randomForest(tipoagua~.,data=entrenamiento,ntree=2000)

> pruebaRF<-predict(entrenamientoRF,prueba)

> matrizconfusion<-confusionMatrix(pruebaRF,prueba$tipoagua)

> matrizconfusion
```

Confusion Matrix and Statistics

```
          Reference
Prediction pozo red
```

```
      pozo      13    1
      red        2   20
```

Accuracy : 0.9167

Obtuvimos el mismo valor que con ntree=500. Veamos con ntree=100

ntree=100

```
> set.seed(1400)
> entrenamientoRF<-randomForest(tipoagua~.,data=entrenamiento,ntree=100)
> pruebaRF<-predict(entrenamientoRF,prueba)
> matrizconfusion<-confusionMatrix(pruebaRF,prueba$tipoagua)
> matrizconfusion
```

Confusion Matrix and Statistics

```
          Reference
Prediction  pozo red
     pozo     14   1
     red       1  20
```

Accuracy : 0.9444

Como se puede observar el parámetro ntree influye mucho en la exactitud del modelo y no siempre un valor más grande es mejor. Para elegir el mejor parámetro se puede hacer un script y si el número de datos lo permite barrer un cierto rango del parámetro ntree, eligiendo el mejor. Como recordaremos también es importante la semilla. Por supuesto que con el tiempo verá otros parámetros que le permitirán aun más optimizar el modelo.

Veamos un script que permita fijar máximo y mínimo para semilla y ntree y nos muestre el mejor valor de los parámetro dentro de los rangos elegidos. Recuerde que debe tener cargadas las bibliotecas randomForest y caret

script861	explicacion
print("introduzca el valor mínimo de la semilla. Un número entero")	pide valor minimo de semilla
minimo<-as.numeric(scan(file="",what="",nmax=1))	asigna el valor a la variable minimo
print("introduzca el valor máximo de la semilla. Un número entero")	pide el valor máximo de la variable
maximo<-as.numeric(scan(file="",what="",nmax=1))	lo asigna a la variable maximo
mejorsemilla<-data.frame(semilla=NA,exactitud=NA)	crea un data.frame con dos columnas vacias: semilla, exactitud
for(i in minimo:maximo){	inicia un bucle for desde la valor mínimo al máximo de semilla
set.seed(i)	inicializa semilla con el mínimo
entrenamientoRF<- randomForest(tipoagua~.,data=entrenamiento,ntree=500)	realiza el entrenamiento de randomForest con ntree=500 que es el valor por defecto
pruebaRF<-predict(entrenamientoRF,prueba)	realiza la prueba con el grupo de datos de prueba
matrizconfusion<-confusionMatrix(pruebaRF,prueba$tipoagua)	crea la matriz de confusion con la biblioteca caret
mejorsemilla[i,1]=i	asigna al data.frame mejor semilla en columna 1, el valor de la semilla
mejorsemilla[i,2]=as.numeric(matrizconfusion$overall[1])	asigna a la segunda columna el valor de la exactitud con la semilla i
}	cierra el bucle for
mejorsemilla<-na.omit(mejorsemilla)	elimina valores NA
mejoressemilla<- mejorsemilla[mejorsemilla$exactitud==max(mejorsemilla$exactitud),1]	declara e inicializa un vector mejoressemillas con los valores de las semillas que dieron la exactitud máxima (puede ser más de uno)
writeLines(paste('mejor semilla =',mejoressemilla[1]))	
print("introduzca el valor mínimo de ntree. Un número entero")	pide valor minimo de ntree
minimo<-as.numeric(scan(file="",what="",nmax=1))	asigna el valor a mínimo
print("introduzca el valor máximo de ntree. Un número entero")	pide valor máximo de ntree

maximo<-as.numeric(scan(file="",what="",nmax=1))	lo asigna a máximo
mejorntree<-data.frame(ntree=NA,exactitud=NA,semilla=NA)	crea data.frame mejorntree con columnas: ntree, exactitud y semilla
for(i in minimo:maximo){	inicia ciclo for desde minimo a máximo de ntree
set.seed(mejoressemilla[1])	fija como semilla el valor del primer elemento del vector mejoressemillas.
entrenamientoRF<-	
randomForest(tipoagua~.,data=entrenamiento,ntree=i)	realiza el entrenamiento variando ntree desdee minimo a máximo
pruebaRF<-predict(entrenamientoRF,prueba)	realiza la prueba de cada modelo
matrizconfusion<-confusionMatrix(pruebaRF,prueba$tipoagua)	crea la matriz de confusion con caret
mejorntree[i,1]=i	asigna a primer columna de data.frame mejrontree el valor de ntree
mejorntree[i,2]=as.numeric(matrizconfusion$overall[1])	asigna a la segunda columna la exactitud
mejorntree[i,3]=mejoressemilla[1]	asigna a la columna 3 el valor de la semilla
}	cierra el bucle
mejorntree<-na.omit(mejorntree)	elimina valores NA de mejor semilla
mejorexactitud<-	declarae inicializa el vector mejorexactitud con los valores de exactitud
mejorntree[mejorntree$exactitud==max(mejorntree$exactitud),2]	
mejoresntree<-	declara e inicializa un vector mejores ntree, aquellos con máxima exactitud
mejorntree[mejorntree$exactitud==max(mejorntree$exactitud),1]	
writeLines(paste('mejor semilla =',mejoressemilla[1]))	muestra valor de mejor semilla
writeLines(paste('mejor ntree =',mejoresntree[1]))	muestra valor de mejor ntree
writeLines(paste('Exactitud= ', mejorexactitud[1]))	muestra valor de exactutyd para la mejor semilla y ntree

Cree un archivo de texto con los códigos de la columna de la izquierda. Grábelo con el nombre scritp861 y ejecute el siguiente comando

```
> source('script861')
```

en este ejemplo fijamos semillas entre 10 y 20
y ntree entre 450 y 550
debería hallar
mejor semilla = 10
mejor ntree = 450
Exactitud= 0.916666666666667

Entrenamiento y prueba con variables acotadas.

En función de los índices MeanDecreaseGini y MeanDecreaseAccuracy, las variables con más peso en el modelo son : conductividad, cloruro y dureza. Podríamos entrenar un modelo solo con esas variables y ver si el modelo mejora
vemos los datos de entrenamiento

```
> names(entrenamiento)
```

```
 [1] "tipoagua"         "ph"               "conductividad"    "cloruro"
 [5] "solidostotales"   "fosfato"          "fluoruro"         "arsenico"
 [9] "amonio"           "alcalinidadtotal" "sulfato"          "sodio"
[13] "durezatotal"
```

Nos quedaremos entonces con: conductividad, cloruro y durezatotal
plantemos el entrenamiento, utilizando el parámetro ntree en 450 y la semilla en 10, que como vimos nos dió con todas las variables un valor de exactitud de 0.9167.

```
> set.seed(10)

>entrenamientoRF<-
randomForest(tipoagua~conductividad+cloruro+durezatotal,data=entrenamiento,ntree=450)
```

realizamos la prueba

```
> pruebaRF<-predict(entrenamientoRF,prueba)
```

realizamos la matriz de confusion

```
> matrizconfusion<-confusionMatrix(pruebaRF,prueba$tipoagua)

> matrizconfusion
```

Confusion Matrix and Statistics

```
          Reference
Prediction pozo red
     pozo   13   2
     red     2  19

               Accuracy : 0.8889
                 95% CI : (0.7394, 0.9689)
    No Information Rate : 0.5833
    P-Value [Acc > NIR] : 6.84e-05

                  Kappa : 0.7714
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.8667
            Specificity : 0.9048
         Pos Pred Value : 0.8667
         Neg Pred Value : 0.9048
             Prevalence : 0.4167
         Detection Rate : 0.3611
   Detection Prevalence : 0.4167
      Balanced Accuracy : 0.8857

       'Positive' Class : pozo
```

como puede ver, no necesariamente la exactitud mejorará. En este caso restringir las variables a las de mayor peso no ha sido buena idea. Pero cuesta tan poco hacer pruebas que seguramente, contribuirán a hallar un modelo que mejor ajuste.

Módulo 8. Clase 7

Random Forest

La metodología Random Forest dentro del aprendizaje automatizado puede ser utilizada en análisis de clasificación o regresión. El primer caso, visto en la clase anterior es cuando la variable dependiente es cualitativa y tiene diversos niveles de un factor. En el caso de regresión, la variable respuesta es una variable numérica continua.
Veremos a continuación un caso de regresión utilizando datos ya conocidos cuyo procesamiento ya ha sido realizado, pero se repite por cuestión de organización e independencias de las clases. Se sugiere crear un nuevo directorio y un nuevo espacio de trabajo.

Preparación de los datos

Los datos se halla en el archivo datosR87.RData. A continuación y de una manera más resumida realizamos el procesamiento de los datos para aplicar luego las funciones correspondientes.
Introduzca los datos en su espacio de trabajo.

```
> load("datosR87.RData")
```

Los datos ingresan como un data.frame llamado datosaguas. Compruébelo

```
> ls()
```

```
[1] "datosaguas"
```

Estos datos tienen una gran cantidad de variables que corresponden a la concentración de componentes químicos del agua y sus errores de medición, junto a datos descriptivos de origen

```
> names(datosaguas)
```

```
 [1] "codigo"           "fecha"            "localidad"
 [4] "provincia"        "tipoagua"         "responsble"
 [7] "ph"               "cvph"             "conductividad"
[10] "cvconductividad"  "cloruro"          "cvcloruro"
[13] "carbonato"        "cvcarb"           "bicarbonato"
[16] "cvbicar"          "solidostotales"   "cvsolidostotales"
[19] "fosfato"          "cvfosfato"        "nitrato"
[22] "cvnitrato"        "nitrito"          "cvnitrito"
[25] "fluoruro"         "cvfluoruro"       "arsenico"
[28] "cvarsenico"       "amonio"           "cvamonio"
[31] "observaciones"    "alcalinidadtotal" "CValcalinidadtotal"
[34] "consumo"          "sulfato"          "cvsulfato"
[37] "informado"        "sodio"            "cvsodio"
[40] "dqo"              "cvdqo"            "tkn"
[43] "cvtkn"            "calcio"           "cvcalcio"
[46] "durezatotal"      "cvdurezatotal"    "duda"
[49] "numeropersonas"   "orp"              "cvorp"
[52] "magnesio"         "cvmagnesio"       "servicio"
[55] "materiaorganica"  "litio"            "cvlitio"
[58] "arsenicoIII"      "cvarsenicoIII"    "iodo"
[61] "cviodo"           "especiescloro"    "cvespeciescloro"
[64] "muestraagotada"   "manganeso"        "cvmanganeso"
[67] "estroncio"        "cvestroncio"      "plomo"
[70] "cvplomo"          "turbidez"         "cvturbidez"
```

En este caso analizaremos la concentración de arsénico y su relación con otros componentes del agua. Por esta razón seleccionamos las variables de interés y las asignamos a un data.frame que llamamos datosrf7

```
> datosrf7<-datosaguas[,c(5,7,9,11,17,19,25,27,29,32,35,38,46)]
```

que resulta en los siguientes campos

```
> names(datosrf7)
```

```
 [1] "tipoagua"          "ph"                "conductividad"    "cloruro"
 [5] "solidostotales"    "fosfato"           "fluoruro"         "arsenico"
 [9] "amonio"            "alcalinidadtotal" "sulfato"          "sodio"
[13] "durezatotal"
```

Eliminamos filas que tengan datos NA.

```
> datosrf7<-datosrf7[is.na(datosrf7$cloruro)==FALSE & is.na(datosrf7$solidostotales)==FALSE &
is.na(datosrf7$fosfato)==FALSE & is.na(datosrf7$fluoruro)==FALSE &
is.na(datosrf7$arsenico)==FALSE & is.na(datosrf7$sulfato)==FALSE &
is.na(datosrf7$sodio)==FALSE & is.na(datosrf7$durezatotal)==FALSE,]
```

Deben quedar 418 registros, verifíquelo

```
> nrow(datosrf7)
```

```
[1] 418
```

En este caso, la variable respuesta será arsenico. Intentaremos modelizar la concentración de arsénico en agua, en función de las demás variables, utilizando Random Forest.

A continuación preparamos los grupos de entrenamiento y prueba dejando un 90% de datos en el grupo de entrenamiento y para el proceso utilizamos la biblioteca caTools, también conocida.

```
> library(caTools)
```

sembramos una semilla. Recuerde que pueden hallarse valores de semillas más favorables para entrenar el modelo

```
> set.seed(630)
```

y separamos los datos en dos grupos: entrenamiento y prueba con el porcentaje acordado recientemente

```
> split<-sample.split(datosrf7$arsenico,SplitRatio=0.9)

> entrenamiento<-subset(datosrf7,split==1)

> prueba<-subset(datosrf7,split==0)
```

Realizamos las verificaciones correspondientes para evaluar que todo esté en orden. Salvo la variable tipoagua que es un factor, el resto de las variables son numéricas continuas. La función str() es muy útil para ver esto. Realizamos la comprobación para el grupo de datos de entrenamiento

```
> str(entrenamiento)
```

```
'data.frame':    376 obs. of  13 variables:
 $ tipoagua          : Factor w/ 8 levels "","pozo","red",..: 2 3 2 3 3 3 2 3 3 2 ...
 $ ph                : num  7.76 7.38 7.21 6.58 6.64 6.77 7.14 7.45 6.49 6.54 ...
 $ conductividad     : num  1.305 1.81 1.777 0.637 0.285 ...
 $ cloruro           : num  27.87 57.03 55.61 7.86 18.29 ...
 $ solidostotales    : num  400 1550 1685 640 380 ...
 $ fosfato           : num  0 1.26 1.52 0.68 0 ...
```

```
 $ fluoruro            : num  0.725 0.968 0.307 1.216 0.522 ...
 $ arsenico            : num  33.96 27.09 66.42 58.19 3.66 ...
 $ amonio              : num  0.655 0 1.866 0.482 2.299 ...
 $ alcalinidadtotal    : num  531.1 599.3 756.8 327.8 65.3 ...
 $ sulfato             : num  102.9 224.4 89.1 17.7 27.5 ...
 $ sodio               : num  247.6 423.5 437.8 130 38.1 ...
 $ durezatotal         : num  83.7 63 54 51.2 71.1 ...
```

y realizamos la misma comprobación para el set de datos de prueba

```
> str(prueba)
```

```
'data.frame':    42 obs. of  13 variables:
 $ tipoagua            : Factor w/ 8 levels "","pozo","red",..: 3 3 3 5 4 3 3 3 3 2 ...
 $ ph                  : num  7.07 6.2 6.88 7.21 7.3 7.23 8.37 7.78 8.25 7.84 ...
 $ conductividad       : num  5.61 0.292 0.76 0.818 0.383 ...
 $ cloruro             : num  241.8 47.1 32.5 52.2 62.6 ...
 $ solidostotales      : num  3921 250 1265 728 505 ...
 $ fosfato             : num  0.5063 0.0919 0 0.1413 0.3515 ...
 $ fluoruro            : num  0.9852 0.0989 2.0486 0.1008 0.123 ...
 $ arsenico            : num  109.95 9.41 63.78 0 0 ...
 $ amonio              : num  1.232 2.645 0.28 1.708 0.247 ...
 $ alcalinidadtotal    : num  715.7 56.7 389.3 396.5 58.7 ...
 $ sulfato             : num  1247.74 44.78 19.73 2.04 42.59 ...
 $ sodio               : num  1047.6 35.4 192.2 196.4 53.9 ...
 $ durezatotal         : num  252.3 66.3 189.9 56.3 47.4 ...
```

Entrenamiento del modelo

A continuación realizamos el entrenamiento con ciertos valores de argumentos, utilizando como variable dependiente a arsenico y el resto como variables independientes. Incluimos todas las variables independientes que se hallan en el data.frame entrenamiento

En primer lugar cargamos la biblioteca randomForest

```
> library(randomForest)
```

y fijamos el valor de la semilla

```
> set.seed(1310)

> entrenamientoRF<-randomForest(arsenico~.,data=entrenamiento,ntree=500,importance=TRUE)
```

revisemos algunos datos del proceso de entrenamiento, que hallamos en el objeto: entrenamientoRF

```
> entrenamientoRF$call
```

randomForest(formula = arsenico ~ ., data = entrenamiento, ntree = 500, importance = TRUE)
nos da la función utilizada y sus argumentos.

Podemos ver también si el entrenamiento se hizo como regresión o clasificación.

```
> entrenamientoRF$type
```

[1] "regression"

Si deseamos ver los valores medidos de la variable dependiente en los datos de entrenmiento aplicamos el código.

```
> entrenamientoRF$y
```

```
         2          3          4          5          6          8
33.9552704 27.0932981 66.4164029 58.1882861  3.6612575 13.0401788
```

```
        9        10        12        13        14        15
71.7253760 65.7402826 101.7309724 22.0216490 72.7453442 31.6096001
```

.....

Si deseamos ver los valores que el modelo ha predicho para la variable dependiente utilizamos el código

```
> entrenamientoRF$predicted
```

```
        2        3        4        5        6        8
26.4386853 45.9224446 49.7880140 36.4447667 3.0390490 7.7134483
        9        10        12        13        14        15
44.8090461 67.6981553 49.4606172 34.0673907 52.7447579 29.1035158
```

.....

El código siguiente da el número de variables independientes considerados para cada proceso de entrenamiento.

```
> entrenamientoRF$mtry
```

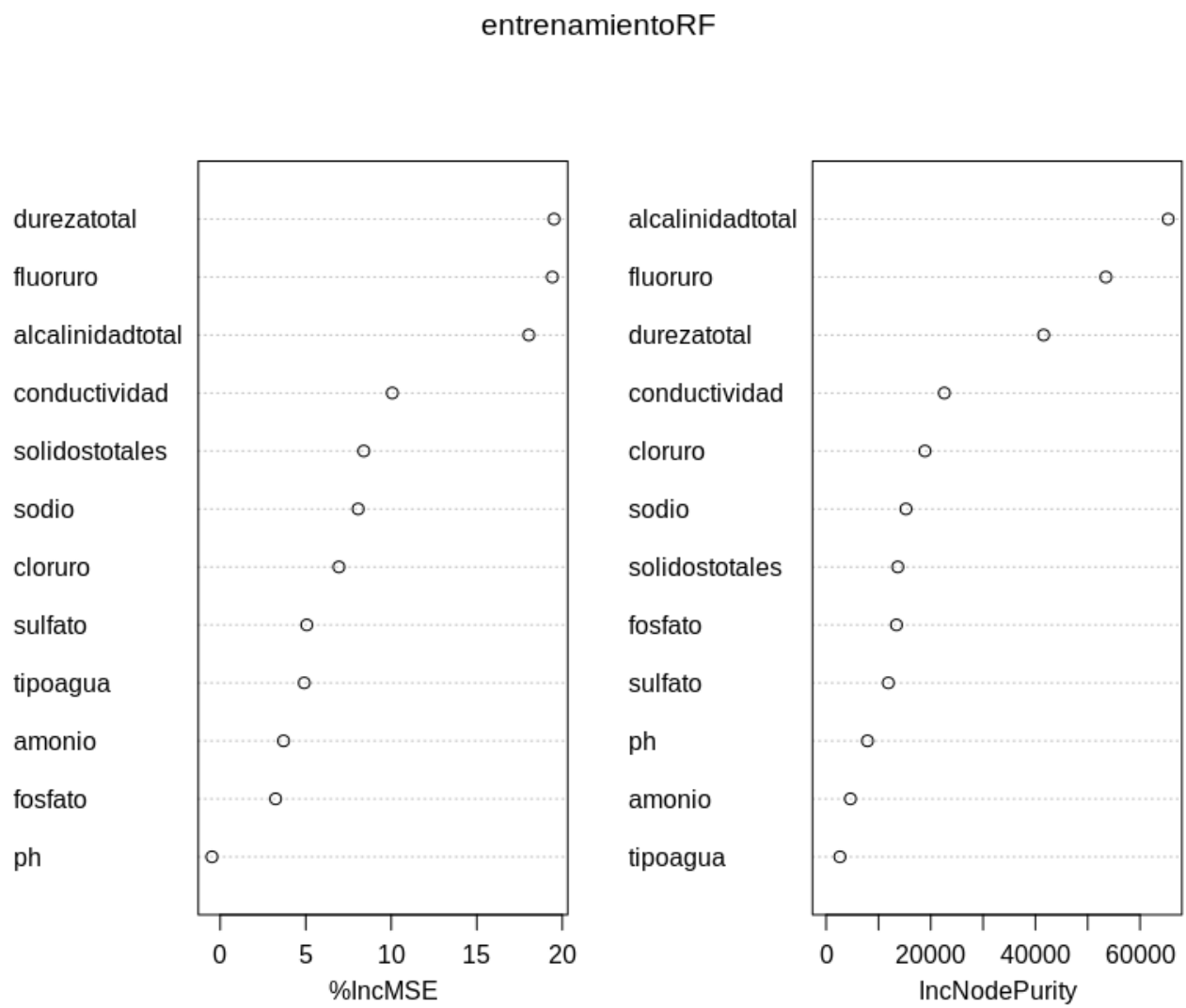
```
[1] 4
```

da el número de variables que se toman por vez, como se mencionó anteriormente, si no se especifica este valor entre los argumentos de la función randomForest(), la función adjudica automáticamente a este número el número de variables independientes dividido 3. En nuestro caso tenemos 12 variables independientes y el valor hallado con $mtry fue 4.

Podemos ver la importancia de cada variable en el modelo

```
varImpPlot(entrenamientoRF)
```

Este código nos muestra el siguiente grafico



Los gráficos son de igual interpretación que los analizados en el caso de clasificación, aunque el nombre de los índices cambia. A mayor $incMSE e IncNodePurity, mayor la importancia de la variable en la regresión.

Podemos poner estos datos también en un objeto, ante la situación que deseemos graficar los mismos con algún otro tipo de presentación, a nuestra medida

```
> importanciavariables<-varImpPlot(entrenamientoRF)
```

en el objeto creado tenemos los valores de ambos índices

```
> importanciavariables
```

	%IncMSE	IncNodePurity
tipoagua	1.9602960	2271.099
ph	-0.4425176	6312.726
conductividad	6.9907655	17551.249
cloruro	4.7318901	16806.421
solidostotales	6.4040727	11235.443
fosfato	3.0920884	12602.209
fluoruro	16.8695850	43361.216
amonio	-1.4682895	4928.164
alcalinidadtotal	13.4426595	46828.719
sulfato	5.2788172	9128.366

sodio	8.6775467	16248.563
durezatotal	16.0203047	37568.930

Testeo del modelo

Finalmente podemos testear el modelo con el grupo prueba

```
> pruebaRF<-predict(entrenamientoRF,prueba)
```

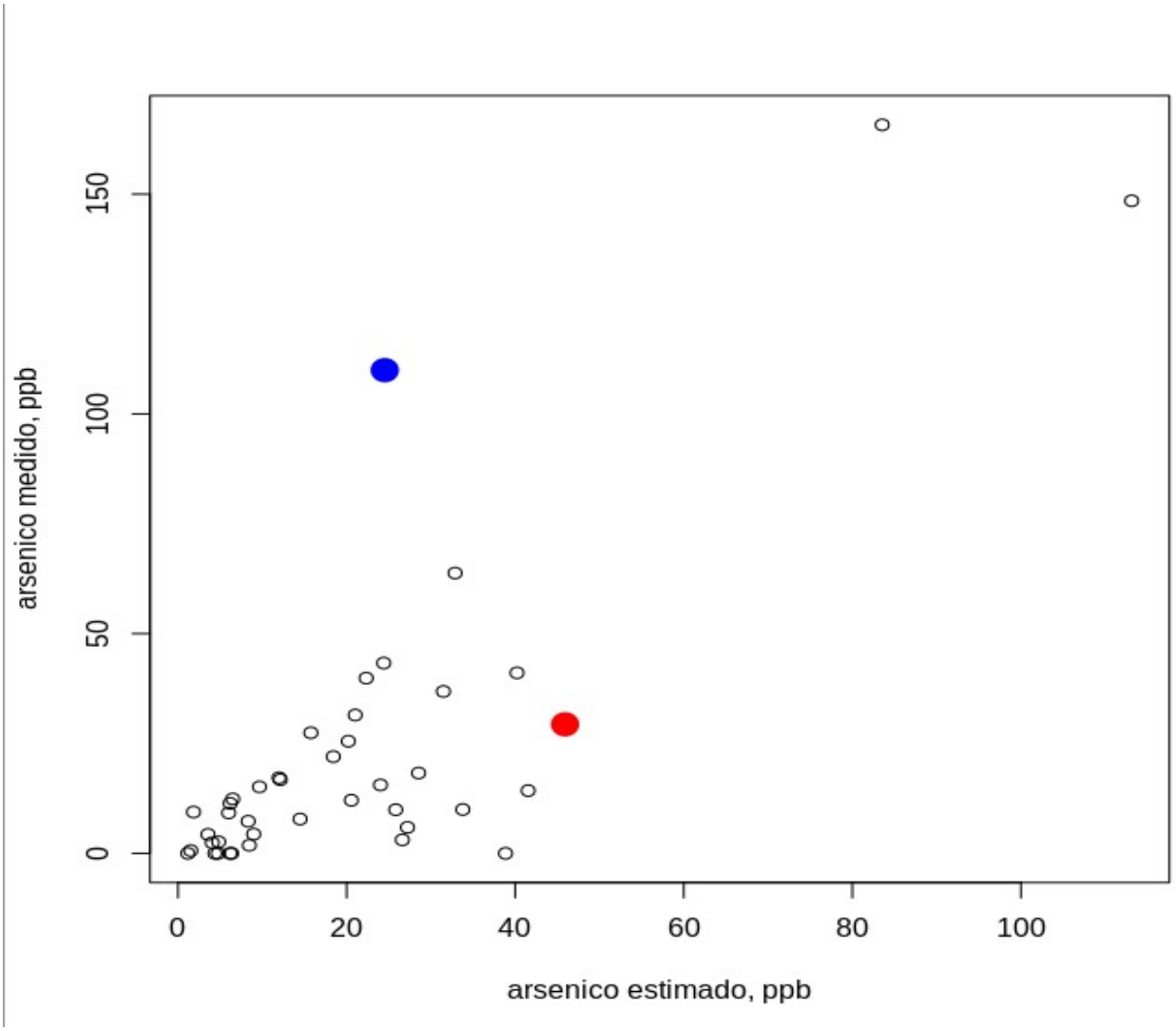
En este caso al tratarse de una regresión, no tiene sentido una matriz de confusión, ni siquiera podría calcularse. Lo que haremos es una análisis de correlación entre los valores estimados de la variable arsenico, que se hallan en el objeto pruebaRF y los medidos que se halla en el objeto prueba$arsenico.

```
> cor.test(pruebaRF,prueba$arsenico)
```

```
        Pearson's product-moment correlation

data:  pruebaRF and prueba$arsenico
t = 8.4848, df = 40, p-value = 1.762e-10
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.6582360 0.8890522
sample estimates:
     cor
0.801767
```

El coeficiente de correlación fue de 0,80 con p-value<0.05, indicando que el modelo ha hallado una buena regresión entre las variable medida y estimada. Podemos ver esto en la gráfica, siguiente

```
> plot(pruebaRF,prueba$arsenico,xlab="arsenico estimado, ppb",ylab="arsenico medido, ppb")
```



Por supuesto que a pesar del adecuado valor de p-value y buen valor del coeficiente de correlación, podemos ver que para un valor medido de aproximadamente 25 ppb, el modelo predijo un valor cercano a 50 ppb (punto rojo). Este podríamos considerarlo un falso positivo, indicando alto arsénico para un agua que no tiene ese nivel. Contrariamente, para un valor de aproximadamente 110, el valor estimado sería cercano 20 (punto azul), claramente representando un falso negativo.
Los puntos elegidos por conveniencia se hicieron utilizando los siguientes códigos, a partir de los objetospruebaRF y prueba$arsenico. Primero creamos un data.frame con datos estimados y medidos de arsénico

```
> arsenicos<-as.data.frame(cbind(estimado=pruebaRF,medido=prueba$arsenico))

> head(arsenicos)
```

```
    estimado    medido
7  24.530644 109.94834
11  1.842095   9.40600
29 32.870668  63.77949
33 38.856526   0.00000
37  4.736725   0.00000
43 31.485716  36.85020
```

Vemos que la tabla tiene los números de filas de los data.frame originales, por lo tanto renumeramos el data.frame arsenicos

```
> row.names(arsenicos)<-seq(1,nrow(arsenicos))
```

```
> head(arsenicos)
```

```
   estimado    medido
1 24.530644 109.94834
2  1.842095   9.40600
3 32.870668  63.77949
4 38.856526   0.00000
5  4.736725   0.00000
6 31.485716  36.85020
```

seleccionamos el punto rojo fijando adecuadamente los valores de las variables, que hacemos de observar la gráfica. Claramente nuestro punto es el de indice 8

```
> arsenicos[arsenicos$estimado>40 & arsenicos$medido>25,]
```

```
    estimado    medido
8   45.91641  29.35341
9   83.54716 165.76542
10 113.13653 148.48187
31  40.21231  41.07647
```

seleccionamos el punto azul. Claramente es el punto indice 1

```
> arsenicos[arsenicos$estimado>20 & arsenicos$medido>100,]
```

```
    estimado   medido
1   24.53064 109.9483
9   83.54716 165.7654
10 113.13653 148.4819
```

marcamos los puntos sobre el gráfico

```
>  points(arsenicos[8,1],arsenicos[8,2],col="red",cex=2,pch=19)

>  points(arsenicos[1,1],arsenicos[1,2],col="blue",cex=2,pch=19)
```

Una comprobación más que podemos hacer, el hallar una regresión lineal entre valores medidos y estimados. Utilizamos medido como variable dependiente y estimado, como independiente.

```
> rlRF<-lm(medido~estimado,data=arsenicos)
```

pedimos información sobre el análisis

```
> summary(rlRF)
```

```
Call:
lm(formula = medido ~ estimado, data = arsenicos)

Residuals:
      Min       1Q   Median       3Q      Max
-47.622  -7.979   1.118    6.857   81.755

Coefficients:
             Estimate  Std. Error  t value  Pr(>|t|)
(Intercept)   -5.0744      4.8160   -1.054     0.298
estimado       1.3562      0.1598    8.485 1.76e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Residual standard error: 22.14 on 40 degrees of freedom
Multiple R-squared: 0.6428, Adjusted R-squared: 0.6339
F-statistic: 71.99 on 1 and 40 DF, p-value: 1.762e-10

Interpretación de la salida anterior:
1- La ordenada al origen (intercept) no discrepa de cero, en base al valor de p-value. Lo que está indicando que para un arsénico estimado de cero, el medido sería cero.
2- la pendiente de la recta (estimado), tiene un valor de 1,3562, con p-value<0.05. Esto nos está indicando que diferentes valores estimados, hacen referencia a diferentes valores medidos. Es decir, el modelo estima muy bien los valores medidos.
La regresión lineal sigue un modelo
medido = -5.0744 + 1.3562*valor estimado
Es decir que para cada valor estimado se corresponde un medido un 35% más grande. Por ende, el modelo estaría subestimando los valores medidos.

Módulo 8. Clase 8

Funciones en R

A lo largo del curso hemos utilizado funciones de R. Una función es un objeto que realiza una determinada acción y para su funcionamiento requiere de variables, parámetros y argumentos. Se hace tan común su uso, que no tenemos en mente que estamos utilizando funciones. Tan importante es el tema, que consideramos necesarios, comenzar a cerrar este curso con una recopilación de lo aprendido a lo largo de los 8 módulos y agregaremos el mecanismo de modificación de funciones.
Cuando se utiliza una función se escribe el nombre de la función y entre paréntesis se incluyen las variables y los argumentos.
De manera general, cuando utilizamos una función, en R la escribimos de la siguiente forma
nombrefunción(variables y argumentos)
Por ejemplo si utilizamos la función sum()
sum(AA, na.rm=TRUE)
donde AA puede ser un vector que contenga los números a sumar, por ejemplo: AA= (3,5) y na.rm= es un argumento.
y cuando se define una función se utiliza el siguiente código general
nombrefuncion<-function(variables, argumentos){código de las acciones que ejecuta}

Funciones definidas por el usuario

Hemos aprendido a definir funciones, para realizar rutinas. A modo de recordatorio, escribamos una función que nos permita multiplicar un número por una constante, al resultado dividirlo por 3 y finalmente redondear el resultado con un decimal. No existe una única forma de escribir esta función, pero tampoco tendremos muchas variantes. Una forma de hacerlo sería escribir el siguiente código (copielo y péguelo en su consola)

```
mifuncion<-function(x,c){

print('introduzca el número')

x<-as.numeric(scan(file="", what="",nmax=1))

print('introduzca la constante')

c<-as.numeric(scan(file="", what="",nmax=1))

resultado<-x*c/3

print(round(resultado,digits=1))

}
```

Luego de pegarlo hallará

```
> mifuncion<-function(x,c){

+ print('introduzca el número')

+ x<-as.numeric(scan(file="", what="",nmax=1))

+ print('introduzca la constante')

+ c<-as.numeric(scan(file="", what="",nmax=1))

+ resultado<-x*c/3

+ print(round(resultado,digits=1))
```

```
+ }
```

Para ejecutarla simple, ejecute

```
> mifuncion()
```

y le pedirá el valor del número y luego la constante, arrojando al final el resultado. Por supuesto se puede embellecer mucho más la presentación.
Si usamos número=4 y constante=2, resultará

```
> mifuncion()
```

```
[1] "introduzca el número"
1: 4
Read 1 item
[1] "introduzca la constante"
1: 2
Read 1 item
[1] 2.7
```

Modificación de una función definida por el usuario

Para modificar la función, si bien podríamos ingeniarnos de varias maneras, una forma es llamar la función de R, en este caso no utilice los paréntesis

```
> mifuncion
```

Con lo que obtendrá

```
function(x,c){
print('introduzca el número')
x<-as.numeric(scan(file="", what="",nmax=1))
print('introduzca la constante')
c<-as.numeric(scan(file="", what="",nmax=1))
resultado<-x*c/3
print(round(resultado,digits=1))
}
```

La forma quizás más sencilla de modificar una función es llamarla, como acabamos de hacer, copiarla y pegarla en un procesador de texto, modificarla y dejar el mismo nombre o cambiarla de acuerdo a las necesidades. Supongamos que deseo introducir un cambio, de manera que en lugar de mostrarme el resultado final solamente, lo haga con un mensaje "El resultado es: ....". Además la sobreescribiré a la función existente, ya que hace la misma rutina. Modificaría mifunción en su primera y última línea, como muestra el código siguiente, resaltado en amarillo

```
> mifuncion<-
function(x,c){
print('introduzca el número')
x<-as.numeric(scan(file="", what="",nmax=1))
print('introduzca la constante')
c<-as.numeric(scan(file="", what="",nmax=1))
resultado<-x*c/3
writeLines(paste('El resultado es: ',round(resultado,digits=1)))
}
```

Copio al portapapeles la función modificada y la pego en la consola. Si ahora ejecuto nuevamente la función con los mismos números

```
> mifuncion()
```

```
[1] "introduzca el número"
1: 4
Read 1 item
[1] "introduzca la constante"
1: 2
Read 1 item
El resultado es:  2.7
```

También podríamos definir la función con argumentos. El argumento es en realidad una variable que tiene un valor por defecto. Supongamos que definimos una nueva función, con las variables x y c, como la anterior, pero le agregamos el argumento digits=1. Es decir que por defecto redondeara con 1 decimal

```
funcionargumento<-

function(x,c,digits=1){

resultado<-x*c/3

writeLines(paste('El resultado es: ',round(resultado,digits=digits)))

}
```

En esta función vemos en su cuerpo que no se introduce x y c, entonces debe hacerse al llamar la función, veamos

```
> funcionargumento(4,2)
```

```
El resultado es:  2.7
```

Si desearamos que la función redondee con dos decimales, al llamar la función modificamos su argumento.

```
> funcionargumento(4,2,digits=2)
```

```
El resultado es:  2.67
```

También podría ejecutarse como

```
> funcionargumento(4,2,2)
```

La función sobreentiende el orden de los números y argumentos.

La modificación de esta función sigue el mismo mecanismo mostrado más arriba. Si quisiera que por defecto tenga 2 decimales. Llamo la función, sin paréntesis y la pego en un procesador

```
> funcionargumento
```

```
function(x,c,digits=1){
resultado<-x*c/3
writeLines(paste('El resultado es: ',round(resultado,digits=digits)))
}
```

e introduzco los cambios en amarillo

```
> funcionargumento<-
function(x,c,digits=2){
resultado<-x*c/3
writeLines(paste('El resultado es: ',round(resultado,digits=digits)))
}
```

Copio la función modificada y la pego en la consola

```
> funcionargumento<-
```

```
+ function(x,c,digits=2){
+ resultado<-x*c/3
+ writeLines(paste('El resultado es: ',round(resultado,digits=digits)))
+ }
```

si ejecuto ahora el mismo cálculo

```
> funcionargumento(4,2)
```

El resultado es: 2.67

vemos que ya trabajó con dos decimales. Por supuesto puede ser modificado el número de decimales, en tiempo de ejecución. Si ejecutamos

```
> funcionargumento(4,2,digits=1)
```

El resultado es: 2.7

Pero si ejecutamos

```
> funcionargumento(4,2)
```

El resultado es: 2.67

vemos que la modificación del argumento fue solo por esa ejecución. El valor por defecto permaneció en este caso en 2.

Funciones no definidas por el usuario

Cada biblioteca de R nos proporciona funciones más o menos complejas. Algunas de ellas pueden tener centenas de líneas, habitualmente de difícil comprensión. En este y los módulos anteriores utilizamos un incontable número de funciones de R definidas y desarrolladas por usuarios. Veremos ahora como modificarlas

A modo de ejemplo, recordemos algunas funciones clásicas:

La función sum()

```
> sum(aditivos$precio)
```

esta función está sumando los valores de la columna precio del data.frame aditivos. Como podemos ver solo hay variable, pero no argumentos. Cuando lo escribimos de esta manera es porque estamos aceptando los argumentos por defecto de la función.

Otra función que hemos utilizado a menudo fue aov(), para el análisis de la variancia

```
> aov(presionarterial~edad+peso+dosisA,data=datosPA)
```

esta función está realizando un ANOVA utilizando como variable respuesta: presionarterial y como factores o variables determinantes: edad, peso y dosisA, datos que se hallan en el data.frame datosPA. Podríamos escribir el código

```
> PAANOVA<-aov(presionarterial~edad+peso+dosisA,data=datosPA)
```

En este caso utilizamos la misma función, pero el resultado de la misma lo adjudicamos a un objeto que llamamos PAANOVA, que podremos utilizar en análisis posteriores, como por ejemplo comparaciones múltiples.

Veamos algunos ejemplos para recordar algunos términos. Introduzcamos los datos de la tablaR881 de la planilla de cálculo tablaR8-8.ods/xls

```
> tablaR881<-read.table("clipboard",header=TRUE,sep="\t",dec=",")
```

y verifiquemos su contenido y correcta introducción de datos

```
> tablaR881
```

```
  var1 var2
1  1.2  2.3
2  2.0  3.0
3  3.4  5.0
4  5.0  5.6
5  5.0  9.7
6  5.1   NA
7  2.9 25.0
8  2.5  0.1
9  4.5  0.3
```

```
> summary(tablaR881)
```

```
      var1            var2       
 Min.   :1.200   Min.   : 0.100  
 1st Qu.:2.500   1st Qu.: 1.800  
 Median :3.400   Median : 4.000  
 Mean   :3.511   Mean   : 6.375  
 3rd Qu.:5.000   3rd Qu.: 6.625  
 Max.   :5.100   Max.   :25.000  
                 NA's   :1
```

Bien, los datos han ingresado de manera numérica, como nos indicó la función summary().
Aplicaremos la función mean() para obtener la media de todos los datos de cada columna por separado. Vamos con la primer columna var1.

```
> mean(tablaR881$var1)
```

```
[1] 3.511111
```

La función mean(), promedió los 8 valores de la columna var1 y nos devolvió el resultado, en este caso 3.511111
Podemos adjudicar el valor de la función mean() a un objeto, por ejemplo al objeto meanvar1

```
> meanvar1<-mean(tablaR881$var1)

> meanvar1
```

```
[1] 3.511111
```

La ventaja de adjudicar el valor a un objeto es que nos permite otras operaciones posteriores. Por ejemplo multiplicar su valor, aunque en realidad cuando lo adjudicamos a una variable, podremos sobre él aplicar prácticamente cualquier función, con las restricciones que en cada caso pudieran existir. Por ejemplo la funcion log() no podrá aplicarse sobre un objeto que almacena un número real negativo.

```
> meanvar1*2
```

```
[1] 7.022222
```

Promediemos ahora los datos de la variable var2 del data.frame tablaR881

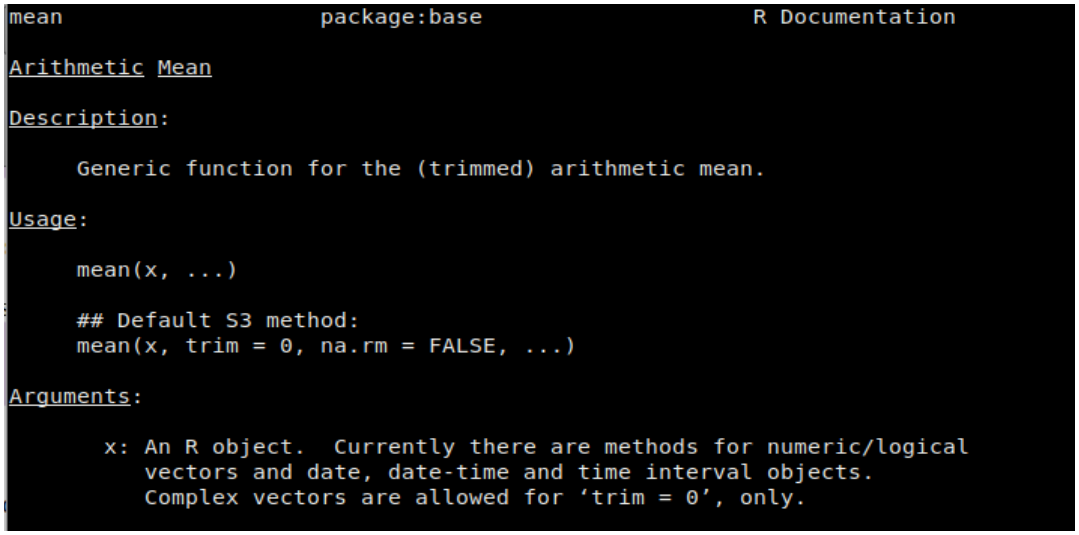
```
> mean(tablaR881$var2)
```

```
[1] NA
```

vemos que el resultado que nos arrojó fue NA. Ya hemos enfrentado este problema y lo hemos solucionado. El problema acá es que estamos utilizando los argumentos propuestos por R por defecto. Por defecto, R incluye todos los valores en el proceso de promediar datos. Como vemos en los datos uno de los valores es NA y como consecuencia el promedio arroja un resultado NA, ya que R no entiende cómo promediar varios números reales a "NA"

La función help() nos ayudará siempre en este caso y es recomendable siempre y cada vez con más frecuencia analizar los help() de cada función. Veamos el help() para la función mean()

```
> help(mean)
```



```
mean                    package:base                    R Documentation

Arithmetic Mean

Description:

     Generic function for the (trimmed) arithmetic mean.

Usage:

     mean(x, ...)

     ## Default S3 method:
     mean(x, trim = 0, na.rm = FALSE, ...)

Arguments:

       x: An R object.  Currently there are methods for numeric/logical
          vectors and date, date-time and time interval objects.
          Complex vectors are allowed for 'trim = 0', only.
```

El archivo de ayuda, verá que continua y tiene información que a primera vista nos resultará extraña e inentendible en su mayoría. La práctica cotidiana del uso de help(), nos irá facilitando el entendimiento del idioma y el aprovechamiento de cada función y por sobretodo evitar errores por mala utilización.

Volviendo a nuestro problema, vemos que la función que se utilizan son

mean(x,.....)

o la función default

mean(x,trim=0,na.rm=FALSE)

En la segunda vemos el argumento na.rm, que podríamos traducir "remover o eliminar valores NA". Esto nos está indicando que por defecto, la función mean() no elimina los valores NA del cálculo y por ende si intenta promediar valores numéricos con valores NA, no sabrá qué hacer y no devolverá ningún valor. Por esa razón el resultado fue NA.

Por supuesto que ya lo sabemos, podemos modificar esto al escribir la función, cambiando el argumento

```
> mean(tablaR881$var2,na.rm=TRUE)
```

[1] 6.375

Con lo que obtenemos el resultado deseado, habiendo sido excluido de la suma el valor NA que teníamos en la variable var2. Si volvemos a ejecutar la función

```
> mean(tablaR881$var2)
```

[1] NA

comprobamos que el cambio en el argumento na.rm, fue transitorio y el cambio solo dura una ejecución.

En conclusión, podemos cambiar los argumentos de la función, modificando su valor dentro del paréntesis de la función, pero este cambio es transitorio, durando solo una ejecución. Esto es válido para todos los argumentos y siempre R volverá a sus valores por defecto.

Por el tipo de trabajo que hagamos puede resultar conveniente dejar el argumento na.rm en TRUE, pero en otros casos, puede llegar a ser conveniente dejarlo en FALSE.

Solo a modo de ejemplo podemos plantear que si nosotros tuviéramos que introducir valores de una variable numérica en una base de datos y estos valores luego se promediaran y además siempre

debe existir un valor de la variable para todos los registros, nos convendrá dejar na.rm=FALSE. De esta manera, cuando se ejecute la función mean(), si por error no se hubiera introducido algún valor, el resultado sería NA y esto nos alertaría del error. Por supuesto todo esto lo podremos hacer de manera automática, utilizando scripts y mecanismos de alerta. Contrariamente, si en nuestro trabajo, es habitual promediar los valores de una variable, pero estos pueden tener valores NA, nos convendrá que el argumento na.rm=TRUE. Si el cálculo lo hacemos con un scritp podemos introducir en el mismo la función mean() con el argumento na.rm=TRUE. Pero si la función mean() la ejecutamos por escritura en la línea de comando, cada vez deberemos escribir na.rm=TRUE. En tal caso, cada vez que lo hagamos tendremos que escribir 10 caracteres más. Un buen escritor puede tipear 10 caracteres en 0.5 segundos. Si la función la utiliza 50 veces al día, durante sus 40 años de servicio, al jubilarse podrá decir, perdí 100 horas de mi vida cambiando el argumento de una función! Nadie quiere esto en los tiempos que corren!!!. Más vale invertir esos valiosos minutos viendo una buena serie o compartiendo un momento con personas que ama.
Veamos como modificar de manera permanente un argumento.
Básicamente, nos encontraremos con tres tipos de funciones:
1- definidas por el usuario.
2- pertenecientes a algún paquete específico.
3- funciones generales.
El caso de la función mean() es una función de uso general.
El código de una función lo podemos modificar en el paquete ya instalado o bien en el paquete antes de instalar

Modificación de función instalada

Para ver el código de una función escribimos en la línea de comando el nombre de la función, como se utiliza, pero sin los paréntesis, en este caso, cuando la utilizamos escribimos
entonces escribiremos

```
> mean
```

```
function (x, ...)
UseMethod("mean")
<bytecode: 0x560f63044b58>
<environment: namespace:base>
```

Si tenemos esta salida, escribimos la función y oprimimos dos veces el tabulador

```
> mean <tab> <tab>
```

obtenemos

```
mean            mean.default     mean.POSIXct     meanNA
mean.Date       mean.difftime    mean.POSIXlt     meanvar1
```

Tambien lo podemos hacer escribiendo

```
>methods(mean)
```

```
[1] mean.Date     mean.default      mean.difftime     mean.POSIXct     mean.POSIXlt
```

la función que nos interesa es aquella seguido de .defaut. Entonces escribimos

```
> mean.default
```

con lo que obtenemos es siguiente código, resaltado en amarillo, donde podemos ver el argumento na.rm=FALSE, en rojo

```
function (x, trim = 0, na.rm = FALSE, ...)
{
```

```
    if (!is.numeric(x) && !is.complex(x) && !is.logical(x)) {
        warning("argument is not numeric or logical: returning NA")
        return(NA_real_)
    }
    if (na.rm)
        x <- x[!is.na(x)]
    if (!is.numeric(trim) || length(trim) != 1L)
        stop("'trim' must be numeric of length one")
    n <- length(x)
    if (trim > 0 && n) {
        if (is.complex(x))
            stop("trimmed means are not defined for complex data")
        if (anyNA(x))
            return(NA_real_)
        if (trim >= 0.5)
            return(stats::median(x, na.rm = FALSE))
        lo <- floor(n * trim) + 1
        hi <- n + 1 - lo
        x <- sort.int(x, partial = unique(c(lo, hi)))[lo:hi]
    }
    .Internal(mean(x))
}
<bytecode: 0x560f630215e8>
<environment: namespace:base>
```

Una forma sencilla de modificarla, es copiar todo el código y asignarla a una nueva función, en este caso meanNA, y le cambiaremos el argumento na.rm=TRUE. El resto lo dejamos como está

```
meanNA<-function (x, trim = 0, na.rm = TRUE, ...)
{
    if (!is.numeric(x) && !is.complex(x) && !is.logical(x)) {
        warning("argument is not numeric or logical: returning NA")
        return(NA_real_)
    }
    if (na.rm)
        x <- x[!is.na(x)]
    if (!is.numeric(trim) || length(trim) != 1L)
        stop("'trim' must be numeric of length one")
    n <- length(x)
    if (trim > 0 && n) {
        if (is.complex(x))
            stop("trimmed means are not defined for complex data")
        if (anyNA(x))
            return(NA_real_)
        if (trim >= 0.5)
            return(stats::median(x, na.rm = FALSE))
        lo <- floor(n * trim) + 1
        hi <- n + 1 - lo
        x <- sort.int(x, partial = unique(c(lo, hi)))[lo:hi]
    }
```

```
   .Internal(mean(x))
}
```

El código anterior, lo pegamos en la consola y oprimimos enter. Hemos definido una nueva función que promedia, pero excluye los NA.

Si ejecutamos la función convencional

```
> mean(tablaR881$var2)
```

```
[1] NA
```

como ya hemos visto obtenemos de resultado NA. Pero si utilizamos la nueva función, que por defecto excluye los NA

```
> meanNA(tablaR881$var2)
```

```
[1] 6.375
```

vemos que el resultado obtenido es un número, porque nuestra función ya eliminó los NA.

Modificando t.test

La función t.test() ha sido usada ampliamente en este curso y estamos familiarizados con su salida, pero podemos desear modificarla. Lo primero que haremos es escribir la función sin argumentos

```
> t.test
```

```
function (x, ...)
UseMethod("t.test")
<bytecode: 0x555e3265edb8>
<environment: namespace:stats>
```

como vemos no nos muestra la función. Entonces ejecutamos el siguiente comando, donde vemos que hay dos métodos que involucran a t.test.

```
> methods(t.test)
```

```
[1] t.test.default*     t.test.formula*
see '?methods' for accessing help and source code
```

como ambos métodos están con asterisco, no podemos ver la función escribiendo simplemente su nombre. Si lo hacemos

```
> t.test.default
```

```
Error: object 't.test.default' not found
```

nos da un error.

Cuando se halla con asterisco debemos buscar en el paquete. Este procedimiento lo podemos hacer de varias maneras

1- al escribir t.test en la línea de comando, vemos en la última línea que se halla en el paquete stats

```
> t.test
```

```
function (x, ...)
UseMethod("t.test")
<bytecode: 0x5652d9243ad0>
<environment: namespace:stats>
```

2- pidiendo help de la función t.test, obtenemos

```
>help(t.test)
```

```
t.test                  package:stats                  R Documentation

Student's t-Test

Description:

     Performs one and two sample t-tests on vectors of data.

Usage:

     t.test(x, ...)
```

donde vemos en la primer línea: package:stats

3- escribir

```
> ??t.test
```

obtendrá una larga lista, en casos como este, pero en algún lugar hallará su función para realizar Student-s t-Test

```
stats::fisher.test      Fisher's Exact Test for Count Data
stats::pairwise.t.test
                        Pairwise t tests
  Aliases: pairwise.t.test
stats::power.t.test     Power calculations for one and two sample t
                        tests
  Aliases: power.t.test
stats::t.test           Student's t-Test
  Aliases: t.test, t.test.default, t.test.formula
strucchange::Fstats     F Statistics
:
```

vemos stats::t.test

lo que nos está indicando que se halla en el paquete stats. Puede ocurrir que su función la halle en más de un paquete si son funciones muy comunes.

En este caso para editar la función utilizaremos

```
> stats:::t.test.default
```

con lo que obtenemos lo siguiente, resaltado en amarillo. Como podemos ver, la función es bastante larga y difícil de analizar. Puede llevar días entender el código en todos sus aspectos. Pero de ser necesario hacerlo, seguramente dará sus frutos. Como podemos ver entre los argumentos tenemos paired =FALSE. Es decir que por defecto la prueba t.test está fijada para datos independientes o desapareados. Pero si usted tuviera que realizar siempre un t.test de datos apareados le convendría modificar este argumento, dejando paired=TRUE. De esa manera al aplicar la función t.test() no debería especificar el argumento en cuestión.

```
function (x, y = NULL, alternative = c("two.sided", "less", "greater"),
    mu = 0, paired = FALSE, var.equal = FALSE, conf.level = 0.95,
    ...)
{
    alternative <- match.arg(alternative)
    if (!missing(mu) && (length(mu) != 1 || is.na(mu)))
        stop("'mu' must be a single number")
    if (!missing(conf.level) && (length(conf.level) != 1 || !is.finite(conf.level) ||
```

```
    conf.level < 0 || conf.level > 1))
    stop("'conf.level' must be a single number between 0 and 1")
if (!is.null(y)) {
    dname <- paste(deparse(substitute(x)), "and", deparse(substitute(y)))
    if (paired)
        xok <- yok <- complete.cases(x, y)
    else {
        yok <- !is.na(y)
        xok <- !is.na(x)
    }
    y <- y[yok]
}
else {
    dname <- deparse(substitute(x))
    if (paired)
        stop("'y' is missing for paired test")
    xok <- !is.na(x)
    yok <- NULL
}
x <- x[xok]
if (paired) {
    x <- x - y
    y <- NULL
}
nx <- length(x)
mx <- mean(x)
vx <- var(x)
if (is.null(y)) {
    if (nx < 2)
        stop("not enough 'x' observations")
    df <- nx - 1
    stderr <- sqrt(vx/nx)
    if (stderr < 10 * .Machine$double.eps * abs(mx))
        stop("data are essentially constant")
    tstat <- (mx - mu)/stderr
    method <- if (paired)
        "Paired t-test"
    else "One Sample t-test"
    estimate <- setNames(mx, if (paired)
        "mean of the differences"
    else "mean of x")
}
else {
    ny <- length(y)
    if (nx < 1 || (!var.equal && nx < 2))
        stop("not enough 'x' observations")
    if (ny < 1 || (!var.equal && ny < 2))
        stop("not enough 'y' observations")
    if (var.equal && nx + ny < 3)
        stop("not enough observations")
```

```
        my <- mean(y)
        vy <- var(y)
        method <- paste(if (!var.equal)
            "Welch", "Two Sample t-test")
        estimate <- c(mx, my)
        names(estimate) <- c("mean of x", "mean of y")
        if (var.equal) {
            df <- nx + ny - 2
            v <- 0
            if (nx > 1)
                v <- v + (nx - 1) * vx
            if (ny > 1)
                v <- v + (ny - 1) * vy
            v <- v/df
            stderr <- sqrt(v * (1/nx + 1/ny))
        }
        else {
            stderrx <- sqrt(vx/nx)
            stderry <- sqrt(vy/ny)
            stderr <- sqrt(stderrx^2 + stderry^2)
            df <- stderr^4/(stderrx^4/(nx - 1) + stderry^4/(ny -
                1))
        }
        if (stderr < 10 * .Machine$double.eps * max(abs(mx),
            abs(my)))
            stop("data are essentially constant")
        tstat <- (mx - my - mu)/stderr
    }
    if (alternative == "less") {
        pval <- pt(tstat, df)
        cint <- c(-Inf, tstat + qt(conf.level, df))
    }
    else if (alternative == "greater") {
        pval <- pt(tstat, df, lower.tail = FALSE)
        cint <- c(tstat - qt(conf.level, df), Inf)
    }
    else {
        pval <- 2 * pt(-abs(tstat), df)
        alpha <- 1 - conf.level
        cint <- qt(1 - alpha/2, df)
        cint <- tstat + c(-cint, cint)
    }
    cint <- mu + cint * stderr
    names(tstat) <- "t"
    names(df) <- "df"
    names(mu) <- if (paired || !is.null(y))
        "difference in means"
    else "mean"
    attr(cint, "conf.level") <- conf.level
    rval <- list(statistic = tstat, parameter = df, p.value = pval,
```

```
        conf.int = cint, estimate = estimate, null.value = mu,
        alternative = alternative, method = method, data.name = dname)
    class(rval) <- "htest"
    return(rval)
}
```

Para modificar la función procedemos de la misma manera que lo venimos haciendo, copiamos la función en un procesador de texto e introducimos las modificaciones y luego las pegamos en la consola. En este caso en particular nos conviene cambiarle de nombre, la llamaremos t.testAparado. Las siguientes líneas las copiamos y pegamos en la consola

```
t.testApareado<- function (x, y = NULL, alternative = c("two.sided", "less", "greater"),
    mu = 0, paired = TRUE, var.equal = FALSE, conf.level = 0.95,
    ...)
{
    alternative <- match.arg(alternative)
    if (!missing(mu) && (length(mu) != 1 || is.na(mu)))
        stop("'mu' must be a single number")
    if (!missing(conf.level) && (length(conf.level) != 1 || !is.finite(conf.level) ||
        conf.level < 0 || conf.level > 1))
        stop("'conf.level' must be a single number between 0 and 1")
    if (!is.null(y)) {
        dname <- paste(deparse(substitute(x)), "and", deparse(substitute(y)))
        if (paired)
            xok <- yok <- complete.cases(x, y)
        else {
            yok <- !is.na(y)
            xok <- !is.na(x)
        }
        y <- y[yok]
    }
    else {
        dname <- deparse(substitute(x))
        if (paired)
            stop("'y' is missing for paired test")
```

```
    xok <- !is.na(x)
    yok <- NULL
}
x <- x[xok]
if (paired) {
    x <- x - y
    y <- NULL
}
nx <- length(x)
mx <- mean(x)
vx <- var(x)
if (is.null(y)) {
    if (nx < 2)
        stop("not enough 'x' observations")
    df <- nx - 1
    stderr <- sqrt(vx/nx)
    if (stderr < 10 * .Machine$double.eps * abs(mx))
        stop("data are essentially constant")
    tstat <- (mx - mu)/stderr
    method <- if (paired)
        "Paired t-test"
    else "One Sample t-test"
    estimate <- setNames(mx, if (paired)
        "mean of the differences"
    else "mean of x")
}
else {
    ny <- length(y)
    if (nx < 1 || (!var.equal && nx < 2))
        stop("not enough 'x' observations")
    if (ny < 1 || (!var.equal && ny < 2))
```

```
        stop("not enough 'y' observations")
    if (var.equal && nx + ny < 3)
  names(mu) <- if (paired || !is.null(y))
    "difference in means"
  else "mean"
  attr(cint, "conf.level") <- conf.level
  rval <- list(statistic = tstat, parameter = df, p.value = pval,
    conf.int = cint, estimate = estimate, null.value = mu,
    alternative = alternative, method = method, data.name = dname)
  class(rval) <- "htest"
  return(rval)
}          stop("not enough observations")
    my <- mean(y)
    vy <- var(y)
    method <- paste(if (!var.equal)
        "Welch", "Two Sample t-test")
    estimate <- c(mx, my)
    names(estimate) <- c("mean of x", "mean of y")
    if (var.equal) {
        df <- nx + ny - 2
        v <- 0
        if (nx > 1)
            v <- v + (nx - 1) * vx
        if (ny > 1)
            v <- v + (ny - 1) * vy
        v <- v/df
        stderr <- sqrt(v * (1/nx + 1/ny))
    }
    else {
        stderrx <- sqrt(vx/nx)
        stderry <- sqrt(vy/ny)
```

```
        stderr <- sqrt(stderrx^2 + stderry^2)
        df <- stderr^4/(stderrx^4/(nx - 1) + stderry^4/(ny -
            1))
    }
    if (stderr < 10 * .Machine$double.eps * max(abs(mx),
        abs(my)))
        stop("data are essentially constant")
    tstat <- (mx - my - mu)/stderr
}
if (alternative == "less") {
    pval <- pt(tstat, df)
    cint <- c(-Inf, tstat + qt(conf.level, df))
}
else if (alternative == "greater") {
    pval <- pt(tstat, df, lower.tail = FALSE)
    cint <- c(tstat - qt(conf.level, df), Inf)
}
else {
    pval <- 2 * pt(-abs(tstat), df)
    alpha <- 1 - conf.level
    cint <- qt(1 - alpha/2, df)
    cint <- tstat + c(-cint, cint)
}
cint <- mu + cint * stderr
names(tstat) <- "t"
names(df) <- "df"
names(mu) <- if (paired || !is.null(y))
    "difference in means"
else "mean"
attr(cint, "conf.level") <- conf.level
rval <- list(statistic = tstat, parameter = df, p.value = pval,
```

```
    conf.int = cint, estimate = estimate, null.value = mu,
    alternative = alternative, method = method, data.name = dname)
  class(rval) <- "htest"
  return(rval)
}
```

Es importante remarcar que esta nueva función tiene un argumento cambiado, en referencia a paired. Esta nueva función que hemos introducido al espacio de trabajo, solo estará disponible en ese espacio de trabajo. Si deseamos modificar una función de un dado paquete de R y que la función modificada se halle disponible en todos los espacios de trabajo, deberemos modificar los archivos de instalación, tema que veremos a continuación con un sencillo ejemplo. Otra opción que veremos en la próxima clase, será hacer un nuevo paquete e instalarlo en nuestra computadora y por que no, compartirla con colegas.

Modificando archivos con código fuente

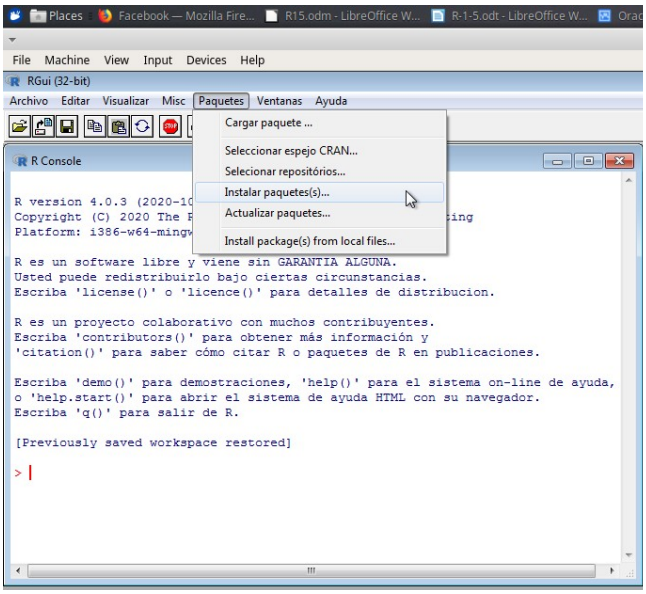
Antes de comenzar con el tema, debemos recordar algunos conceptos básicos introducidos en el módulo 1 de este curso.

Es normal que debamos instalar paquetes o bibliotecas para realizar ciertas tareas, dado que esas bibliotecas contienen funciones que no se instalan por defecto.

Para ilustrar este tema elegiremos la biblioteca pwr(), utilizada en el módulo 3, para los análisis de potencia.

Si deseamos instalar una biblioteca, tenemos varios mecanismos, algunos más sencillos y otros más complejos, siendo estos últimos con mayor potencial. Recordemos algunas formas de instalar paquetes

1- Desde la interfaz gráfica del usuario en entornos Windows, desplegamos el menú
> Paquetes > Instalar paquetes
como muestra la figura



y todo es cuestión de unos pocos clics. Por supuesto puede haber problemas con incompatibilidad de versión de R con la de la última versión del paquete.

2- A través de línea de comando, escribimos

```
> install.packages("pwr",dependencies=TRUE)
```

y habitualmente terminamos con la biblioteca instalada, aunque pueden ocurrir algunos problemas de incompatibilidad de versiones.
3- Descargar los archivos fuente del paquete, conocidos como archivos tar.gz e instalarlos desde la línea de comando. Supongamos que hemos descargado el archivo pwr_1.3.0.tar.gz, el que lo dirigimos en la descarga al directorio Downloads. En este caso podemos utilizar el siguiente código

```
>install.packages("~/Downloads/
pwr_1.3.3.tar.gz",repos=NULL,type="source",dependencies=TRUE)
```

El mecanismo no está exento de incompatibilidades, pero ya vimos que siempre al menos por este camino, la solución al problema siempre existirá.
Este último mecanismo parece ser el más fácil de utilizar cuando deseamos hacer modificaciones permanentes de alguna función. Este trabajo no es sencillo, pero tampoco está solo limitado a expertos en el tema. Como en todos los temas, el uso continuo del recurso, lo va tornando más amigable y seguro. Además verá que para modificar una función debe analizar códigos, que le irán dando nuevos conocimientos sobre programación en R.
Para el desarrollo del tema ejecutaremos ensayos ya conocidos. Supongamos que tenemos datos de la concentración de fluoruro en agua, medida en pozos de dos localidades de la misma zona. Introduzca los datos de la tablaR882

```
> tablaR882<-read.table("clipboard",header=TRUE,dec=",",sep="\t")
```

verificaciones de rutina

```
> tablaR882
```

```
   irigoyen barrancas
1       1.5       0.9
2       1.4       0.8
3       1.4       0.8
4       1.2       1.1
5       1.7       1.2
6       0.9       0.7
7       1.2       0.8
8       1.2       0.8
9       1.1       0.5
10      1.0       0.5
11      1.5       0.6
```

```
> summary(tablaR882)
```

```
    irigoyen            barrancas
 Min.   :0.900      Min.   :0.5000
 1st Qu.:1.150      1st Qu.:0.6500
 Median :1.200      Median :0.8000
 Mean   :1.282      Mean   :0.7909
 3rd Qu.:1.450      3rd Qu.:0.8500
 Max.   :1.700      Max.   :1.2000
```

Deseamos conocer si existen diferencias entre los valores de fluoruro en agua de las dos localidades.
Vemos que podemos aceptar distribución normal para ambas muestras, para lo que aplicamos shapiro.test()

```
> shapiro.test(tablaR882$irigoyen)
```

```
        Shapiro-Wilk normality test
data:  tablaR882$irigoyen
```

W = 0.9676, p-value = 0.8613

```
> shapiro.test(tablaR882$barrancas)
```

Shapiro-Wilk normality test

data: tablaR882$barrancas
W = 0.9262, p-value = 0.3737

Los valores de p-value nos permiten aceptar que ambas muestras tienen distribución normal.
Probamos homogeneidad de variancias, comprobando homocedasticidad por valor de p-value obtenido.

```
> bartlett.test(list(tablaR882$irigoyen,tablaR882$barrancas))
```

Bartlett test of homogeneity of variances

data: list(tablaR882$irigoyen, tablaR882$barrancas)
Bartlett's K-squared = 0.064115, df = 1, p-value = 0.8001

Dado el valor de p-value, aceptamos que las variancias son homogéneas. Con lo cual, para la comparación utilizaremos t.test() para datos independientes

```
> t.test(tablaR882$irigoyen,tablaR882$barrancas,paired=FALSE)
```

Welch Two Sample t-test

data: tablaR882$irigoyen and tablaR882$barrancas
t = 4.988, df = 19.867, p-value = 7.199e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
0.2855260 0.6962922
sample estimates:
mean of x mean of y
1.2818182 0.7909091

En base al p-value<0,05, rechazamos la hipótesis nula: la concentración de fluoruro no difiere entre Irigoyen y Barrancas, inclinándonos porque debe haber diferencia entre los valores de fluoruro de ambas localidades.
Haber rechazado que no son iguales no implica que podamos aceptar alegremente que son diferentes. El test de potencia es la respuesta a este último interrogante. Para ello utilizamos la biblioteca pwr, que cargamos

```
> library(pwr)
```

y utilizaremos la función pwr.t.test() para evaluar la potencia. Recordaremos que necesitamos 3 datos, para poder calcular la potencia: numero de datos de cada grupo, p-value y magnitud de efecto, a la que llamamos ez.
numero de datos: 11
p-value=0.05
ez lo calculamos con el siguiente cálculo, para este caso

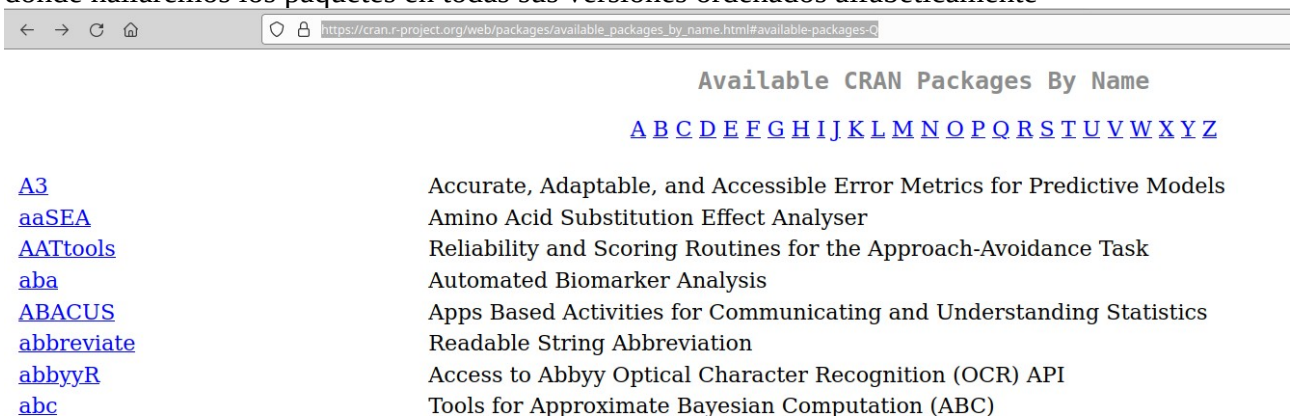
```
> ez=abs(mean(tablaR882$irigoyen)-mean(tablaR882$barrancas))*2/(sd(tablaR882$irigoyen)
+sd(tablaR882$barrancas))

> ez
```

[1] 2.128697

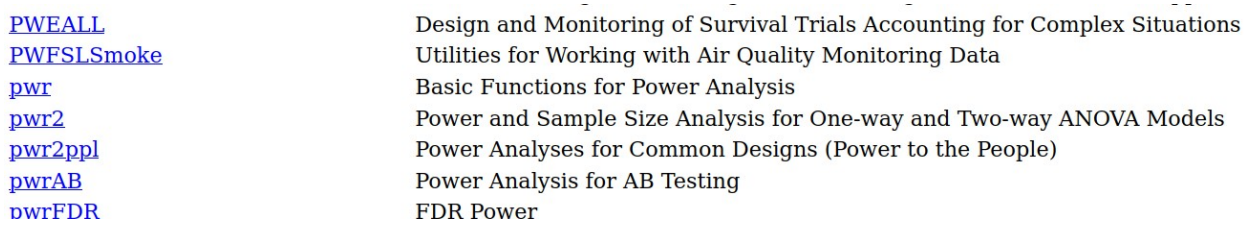
En el valor de ez, número de datos y p-value, calculamos la potencia

```
> pwr.t.test(n = 11, d = ez, sig.level = 0.05, power = NULL, type="two.sample",alternative="two.sided")

     Two-sample t test power calculation

              n = 11
              d = 2.128697
      sig.level = 0.05
          power = 0.9972893
    alternative = two.sided

NOTE: n is number in *each* group
```

Con una potencia de 0.997, podemos decir con baja probabilidad de error que la concentración de fluoruro en el agua de los pozos de Irigoyen difiere de las de Barrancas.
Pero no estamos acá para esto, sino para ver como podemos modificar permanentemente una función perteneciente a un paquete. Manos a la obra!!!
De manera de entender el mecanismo que debemos seguir, solo modificaremos la función en un único aspecto. Solo deseamos que la línea final
NOTE: n is number in *each* group
diga
NOTA: n es el número en cada grupo
Para ello debemos modificar el archivo con el código fuente. Para ello nos dirigimos a la página https://cran.r-project.org/web/packages/available_packages_by_name.html#available-packages-Q donde hallaremos los paquetes en todas sus versiones ordenados alfabéticamente



clickeamos en la letra P y buscamos pwr



hacemos clik en pwr y luego en Package source: pwr_1.3.0.tar.gz

CRAN checks: pwr results

Documentation:

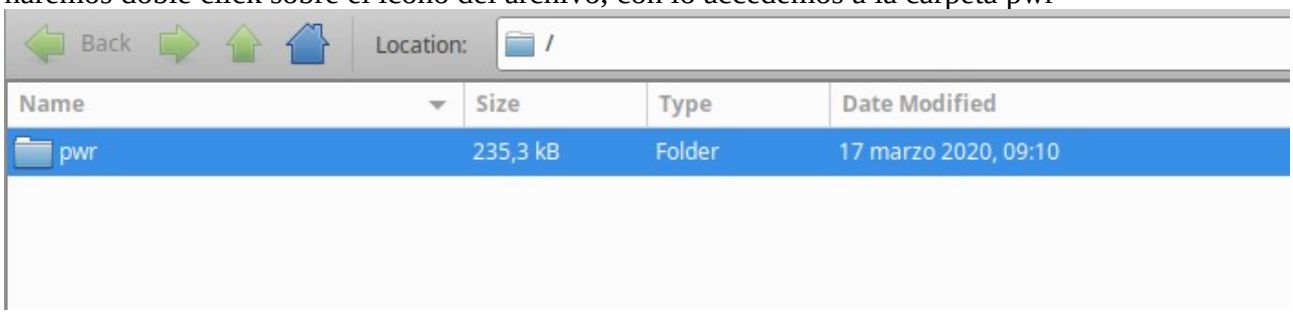
Reference manual: pwr.pdf
Vignettes: Getting started with the pwr package

Downloads:

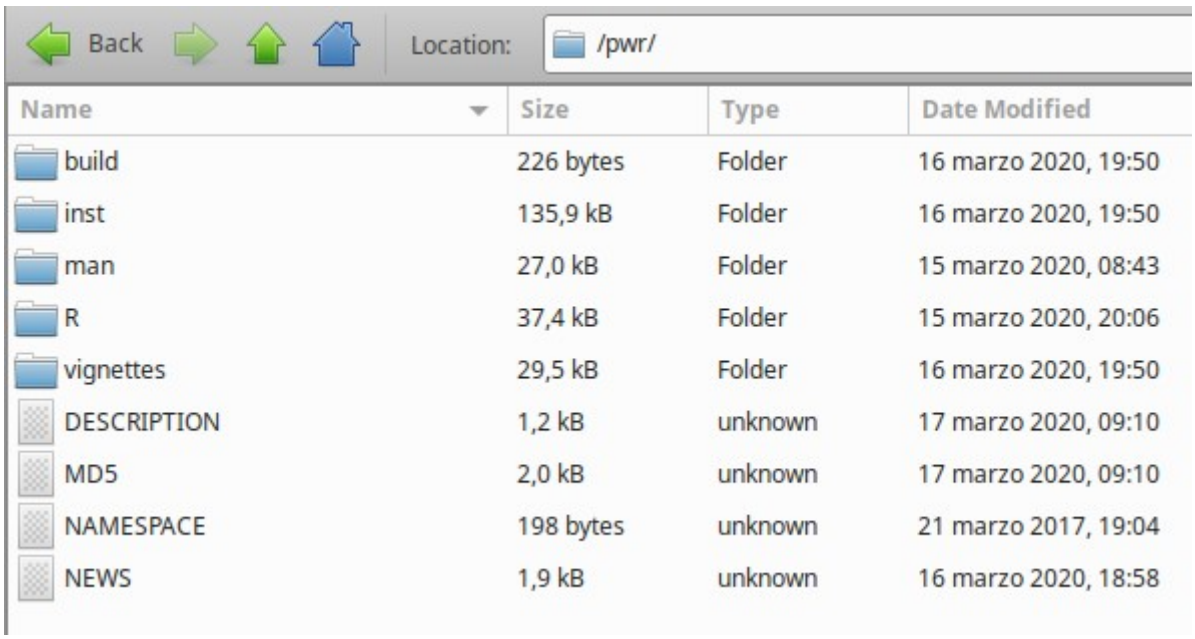
Package source: pwr_1.3-0.tar.gz
Windows binaries: r-devel: pwr_1.3-0.zip, r-release: pwr_1.3-0.zip, r-oldrel: pwr_1.3-0.zip
macOS binaries: r-release (arm64): pwr_1.3-0.tgz, r-release (x86_64): pwr_1.3-0.tgz, r-oldrel: pwr_1.3-0.tgz

nos descargará el archivo en la carpeta Downloads, o donde usted tenga configurado

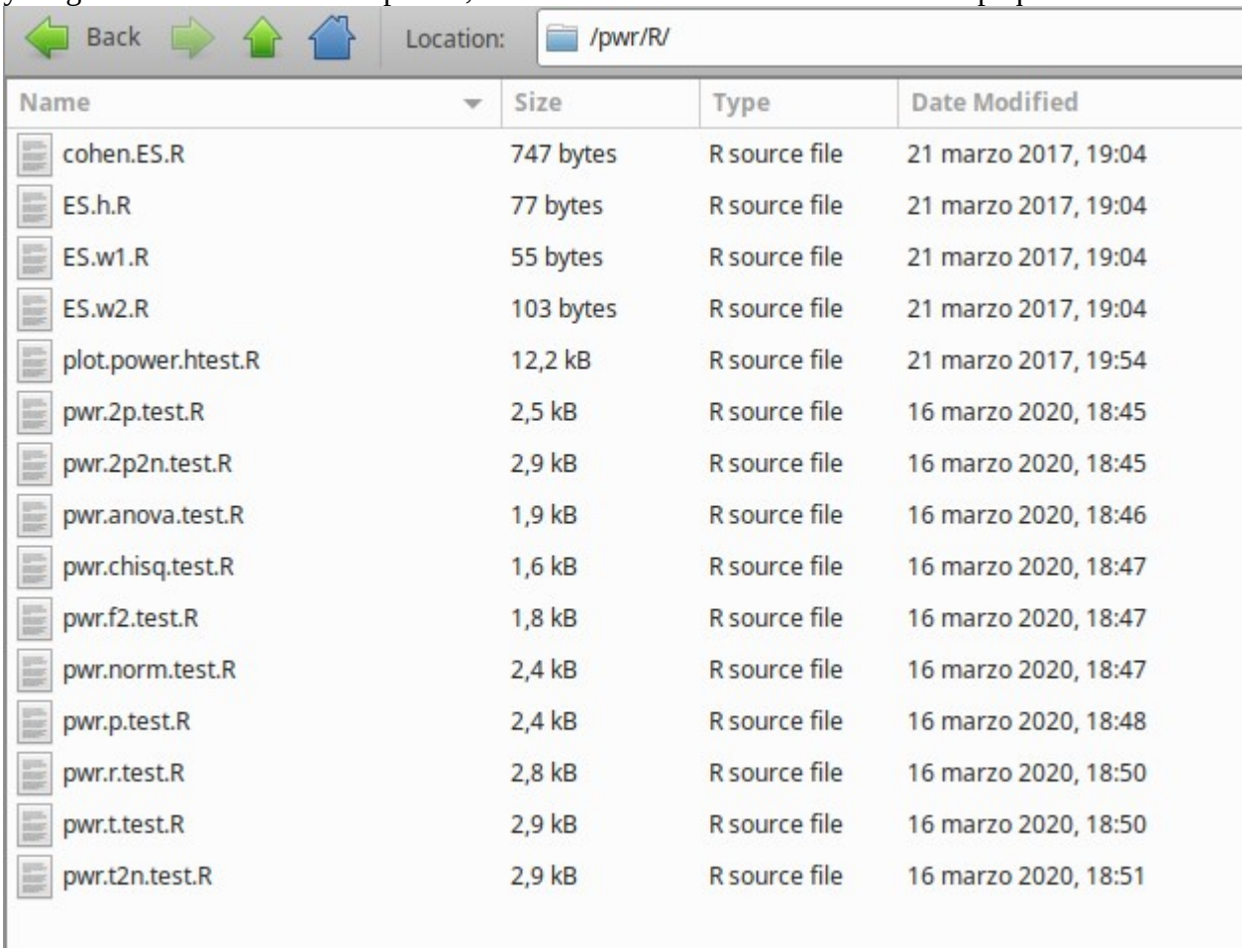
haremos doble click sobre el icono del archivo, con lo accedemos a la carpeta pwr



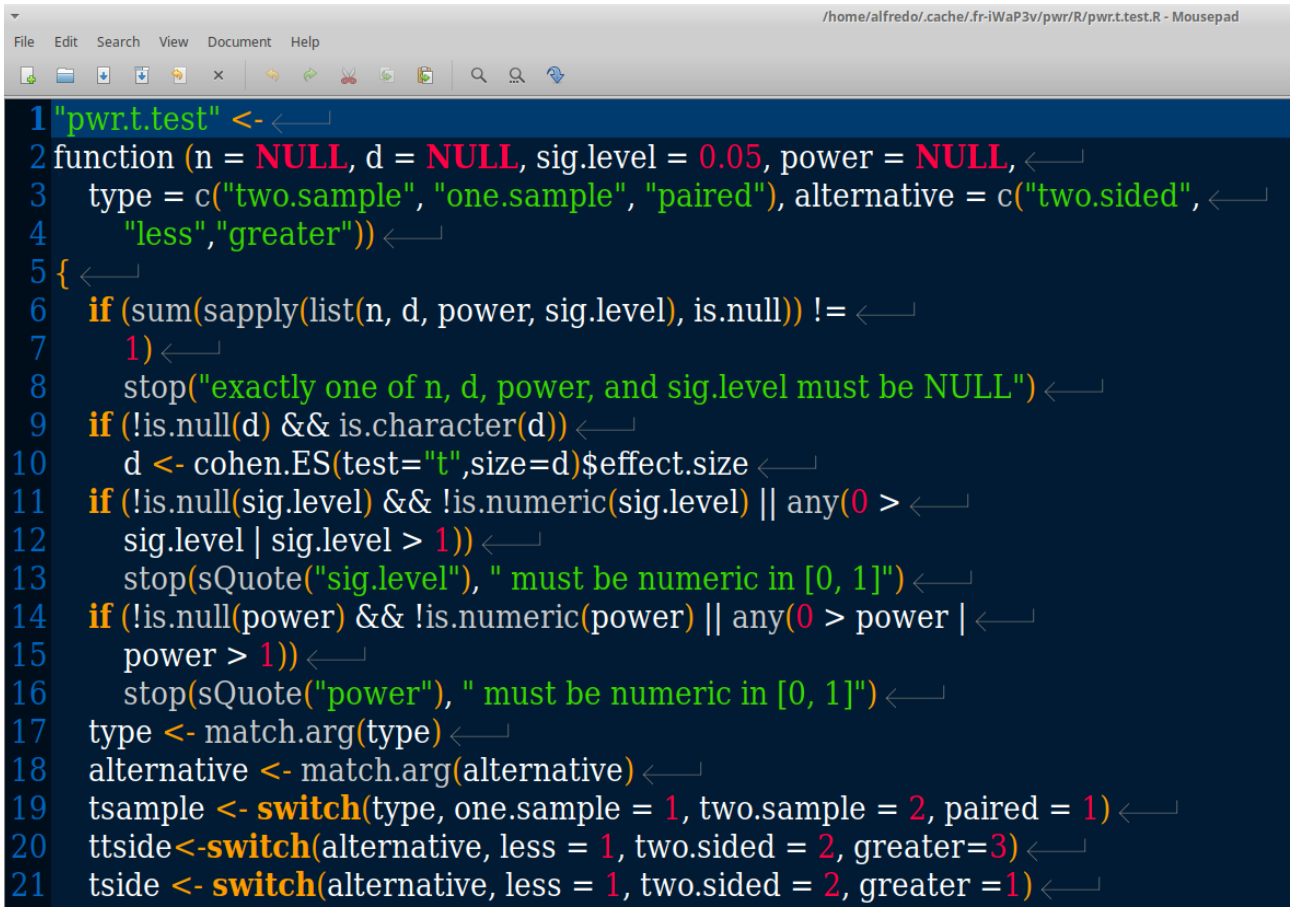
hacemos doble click sobre pwr



y luego doble clic sobre la carpeta R, donde veremos todas las funciones del paquete en cuestión.



La función que deseamos modificar es pwr.t.test.R. Hacemos doble click sobre ella o la abrimos con un editor de texto, que nos mostrará el código fuente de la función. En la figura se muestra solo una parte. A esta altura del curso, aun no es fácil leer de corrido el código, ya que se topará con nuevas estructuras, pero con algo de esfuerzo podrá comprenderlas!



Sin embargo para comprender el procedimiento, nos pusimos un objetivo sencillo: cambiar la nota final

NOTE: n is number in *each* group

por

NOTA: n es el número en cada grupo

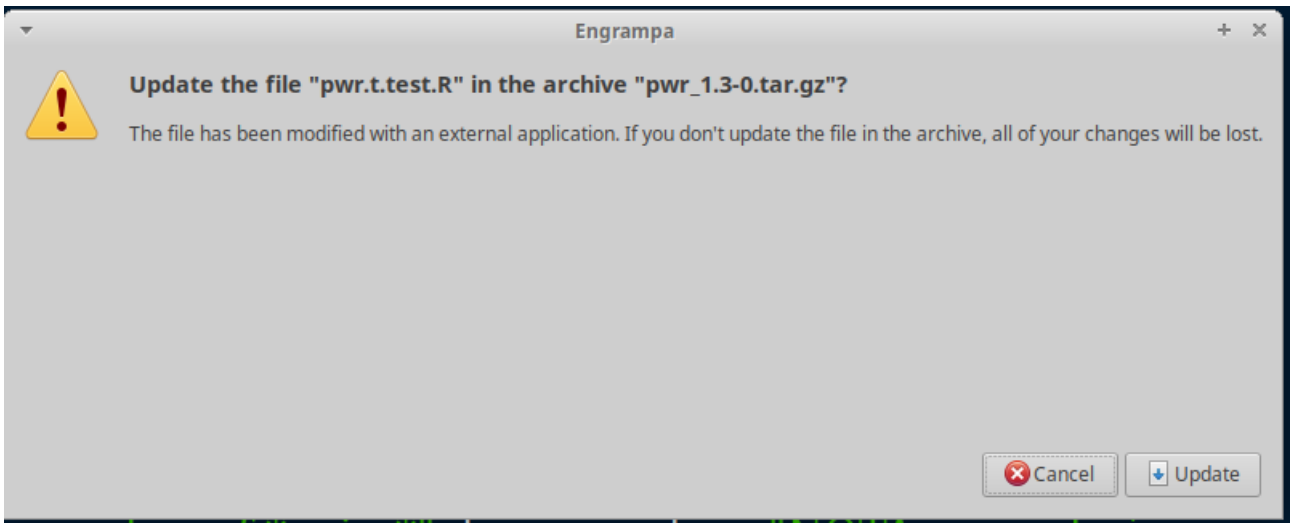
Buscamos en la función el texto a cambiar, que hallamos casi al final

```
    sig.level <- uniroot(function(sig.level) eval(p.body) - power,
                         c(1e-10, 1 - 1e-10))$root
else stop("internal error")
NOTE <- switch(type, paired = "n is number of *pairs*", two.sample = "n is number in *each* group",
  NULL)
METHOD <- paste(switch(type, one.sample = "One-sample", two.sample = "Two-sample",
  paired = "Paired"), "t test power calculation")
```

entonces, al archivo pwr.t.test.R, lo editamos con el procesador que lo abrimos

```
                         c(1e-10, 1 - 1e-10))$root
else stop("internal error")
NOTE <- switch(type, paired = "n is number of *pairs*", two.sample = "NOTA: n es el número en cada grupo",
  NULL)
METHOD <- paste(switch(type, one.sample = "One-sample", two.sample = "Two-sample",
```

colocamos guardar, puede aparecernos un mensaje, en el que oprimimos update



removemos ahora el viejo paquete pwr de nuestra computadora

```
> remove.packages("pwr")
```

Removing package from ‘/home/alfredo/R/x86_64-pc-linux-gnu-library/3.4’
(as ‘lib’ is unspecified)
Salimos del espacio de trabajo, ya que tiene la biblioteca vieja cargada

```
> q()
```

Save workspace image? [y/n/c]: y
y volvemos a ingresar al espacio de trabajo.
Instalamos la biblioteca modificada

```
> install.packages("~/Downloads/pwr_1.3-0.tar.gz",repos=NULL,type="source",dependencies=TRUE)
```

Installing package into ‘/home/alfredo/R/x86_64-pc-linux-gnu-library/3.4’
(as ‘lib’ is unspecified)
* installing *source* package ‘pwr’ ...
file ‘R/pwr.t.test.R’ has the wrong MD5 checksum
** R
** inst
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** installing vignettes
** testing if installed package can be loaded
* DONE (pwr)

cargamos la biblioteca pwr, supuestamente modificada

```
> library(pwr)
```

Si ejecutamos ahora el test de potencia, que tenemos en el historial

```
> pwr.t.test(n = 11, d = ez, sig.level = 0.05, power = NULL, type="two.sample",alternative="two.sided")
```

Two-sample t test power calculation

```
          n = 11
          d = 2.128697
  sig.level = 0.05
      power = 0.9972893
alternative = two.sided
```

NOTE: NOTA: n es el número en cada grupo

Como podemos ver la modificación que hemos introducido, ahora aparece en nuestra salida.
Siguiendo el mismo mecanismo puede modificar funciones dentro del código, argumentos, etc.
No es un trabajo fácil, pero sin duda si lo aplica, sentirá una gran satisfacción además de poder moldear las funciones a su necesidad. Como en todo el uso cotidiano, da velocidad y seguridad en el proceso y como siempre le hará ganar segundos dentro de cada minuto, minutos dentro de cada hora, horas dentro de cada día y así días de su vida.
No desespere si le resulta complejo e inaplicable, miles de usuarios obtienen grandes beneficios del uso de R sin conocer o utilizar estos recursos. Por supuesto, manejarlos le darán el título de EXPERTO.

Módulo 8. Clase 9

Creación de un paquete

En esta clase daremos los lineamientos mínimos necesarios para la creación de una biblioteca o paquete de R. Un paquete de R es un conjunto de archivos que contienen funciones, datos, ayudas y ejemplos referidos a un tema en particular. A lo largo del curso hemos recurrido a la descarga e instalación de bibliotecas, las cuales fueron diseñadas por otros usuarios de R. Estas bibliotecas han dado grandes soluciones a problemas específicos. Ha llegado la hora de hacer nuestra contribución!
En esta clase desarrollaremos nuestro paquete, planteando un ejemplo muy sencillo con el objetivo de ver los pasos a seguir. Un paquete puede ser desarrollado para compartir funciones y procedimientos con otras personas o también es una buena forma de guardar de manera ordenada un grupo de funciones que hemos creadas y que se aplican a una determinada situación.
En esta clase crearemos un paquete que tiene un set de datos (x,y) y dos funciones, cuyo cálculo y manejo no será una complicación. Así, nuestra biblioteca tendrá:
1- función "media": calcula la media de un set de datos.
2- función "desvioestandar": calcula el desvío estándar de un set de datos.
Por supuesto que estas dos funciones ya existen en el paquete stats de R, pero el objetivo es no complicarnos con funciones extrañas, para poder focalizar la atención en los pasos para lograr obtener una biblioteca.
Veamos entonces las partes que contendrá la biblioteca.

Partes de la biblioteca

1- Un data.frame llamado datosR89. Introduzca los datos en su espacio de trabajo a partir de la planilla de cálculo tablaR8-9.ods/xls

```
> datosR89<-read.table('clipboard',header=TRUE,dec=',',sep='\t')

> datosR89
```

```
   x    y
1   0  1.0
2   1  2.0
3   3  3.1
4   3  3.9
5   1  4.0
6  NA  5.0
7   2  6.2
8   7  6.9
9   1  7.9
10  4  9.0
11 10 10.1
```

2- La función media, que definimos de la siguiente manera:

```
media<-function(x,removerNA=FALSE){
if(removerNA==TRUE){
x<-x[!is.na(x)]
}
print("Sus datos son: ")
print(x)
n<-length(x)
```

```
resultado<-sum(x)/n
print(paste("la media de sus datos es: ", resultado))
}
```

Para introducirla en el espacio de trabajo copie el texto anterior resaltado en amarillo y péguelo en la línea de comando.
Si ejecutamos la función

```
> media(datosR89$x)
```

```
[1] "Sus datos son: "
 [1]  0  1  3  3  1 NA  2  7  1  4 10
[1] "la media de sus datos es:  NA"
```

El resultado fue NA, ya que uno de sus datos es NA y el argumento removerNA tiene el valor por defecto: FALSE. Por lo que podemos ejecutar la función, cambiando el argumento mencionado al valor TRUE.
Si ejecutamos

```
> media(datosR89$x,removerNA=TRUE)
```

```
[1] "Sus datos son: "
 [1]  0  1  3  3  1  2  7  1  4 10
[1] "la media de sus datos es:  3.2"
```

También podríamos ejecutar la función sin escribir el nombre del argumento, ya que la función fue definida con un set de datos y un solo argumento. Respetando el orden, la función interpreta nuestra intención

```
> media(datosR89$x,TRUE)
```

```
[1] "Sus datos son: "
 [1]  0  1  3  3  1  2  7  1  4 10
[1] "la media de sus datos es:  3.2"
```

3- Definimos ahora la funcion desvioestandar

```
desvioestandar<-function(x,removerNA=FALSE){
if(removerNA==TRUE){
x<-x[!is.na(x)]
}
print("Sus datos son:")
print(x)
n<-length(x)
media<-mean(x)
resultado<-sqrt(sum((x-media)*(x-media))/(n-1))
print(paste("es desvio estándar es: ", resultado))
}
```

Para introducirla en el espacio de trabajo, copie el texto resaltado en amarillo y péguelo en la línea de comando.
Luego de introducida la función, si ejecutamos

```
> desvioestandar(datosR89$y)
```

```
[1] "Sus datos son:"
 [1]  1.0  2.0  3.1  3.9  4.0  5.0  6.2  6.9  7.9  9.0 10.1
```

[1] "es desvio estándar es: 2.91138829739041"

pero si la ejecutamos con el argumento por defecto para los datos de la columna x, obtenemos

```
> desvioestandar(datosR89$x)
```

```
[1] "Sus datos son:"
 [1]  0  1  3  3  1 NA  2  7  1  4 10
[1] "es desvio estándar es:  NA"
```

si cambiamos el argumento

```
> desvioestandar(datosR89$x,removerNA=TRUE)
```

```
[1] "Sus datos son:"
 [1]  0  1  3  3  1  2  7  1  4 10
[1] "es desvio estándar es:  3.11982905514602"
```

Bueno, ya conocemos las dos funciones que tendrá el paquete, procederemos entonces a la construcción del mismo.
Si bien se pueden seguir diferentes mecanismos, plantearemos lo que consideramos es la forma más organizada y sencilla de tener éxito en este emprendimiento.

Ordenamiento del directorio

Una vez que definió las funciones, se familiarizó con ellas y está seguro de su funcionamiento, deje en su espacio de trabajo solo las funciones y los datos que tendrá el paquete. De esta manera se evitarán confusiones en los archivos creados. Para nuestro caso nos quedaron los datos (datosR89) y las dos funciones creadas (desvioestandar y media). Todo otro objeto creado durante el desarrollo fue eliminado del espacio de trabajo con la función rm(). Así, si observamos el espacio de trabajo con la función ls() obtendremos

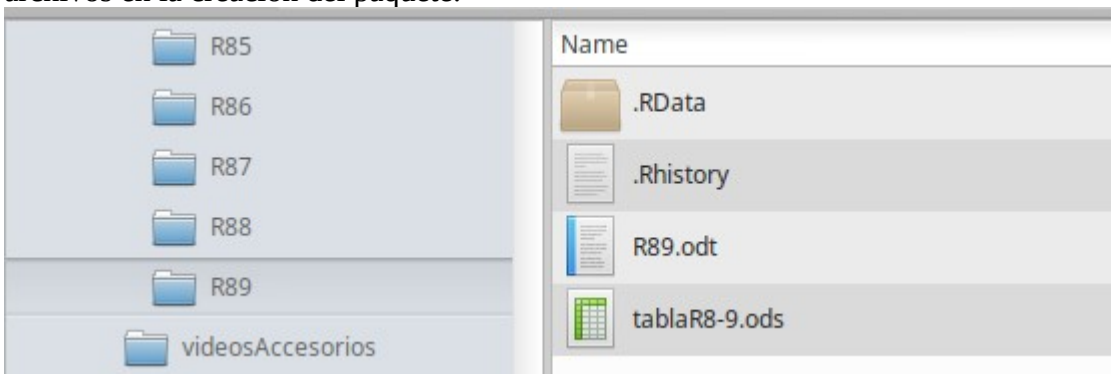
```
> ls()
[1] "datosR89"       "desvioestandar" "media"
>
```

Crearemos el paquete en el mismo directorio en que se halla nuestro espacio de trabajo. Nuestro espacio de trabajo se halla ubicado según indica getwd()

```
> getwd()
```

```
[1] "/media/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R8/R89"
```

por cuestiones de orden y para ver con claridad qué carpetas y archivos se forman, en ese directorio solo dejaremos los archivos correspondientes a la clases R89.odt y tablaR8-9.ods. Además se hallarán como archivos ocultos el espacio de trabajo .RData y .RHistory. No afectarán estos archivos en la creación del paquete.



No es necesario trabajar con el ordenamiento planteado, pero consideramos que para el primer paquete, tener la mayor simplicidad en el directorio, ayudará a comprender qué se va formando a lo largo del proceso.

Creación de paquete

Para crear el paquete algunas acciones las debemos realizar estando en el espacio de trabajo de R y otras fuera del espacio de trabajo. En adelante las instrucciones que se dan se pueden ejecutar con cualquier versión relativamente actual de R. Las acciones que se realizan desde el sistema operativo, se mostrarán como hacerlas para Linux.

La creación del paquete se realiza con la función package.skeleton(), en la que indicamos el nombre del paquete. Esta función es parte de la biblioteca utils, que se instala por defecto al instalar R y carga al iniciar una sesión de R.

```
> package.skeleton(name="mediaSD")
```

```
Creating directories ...
Creating DESCRIPTION ...
Creating NAMESPACE ...
Creating Read-and-delete-me ...
Saving functions and data ...
Making help files ...
Done.
Further steps are described in './mediaSD/Read-and-delete-me'.
```

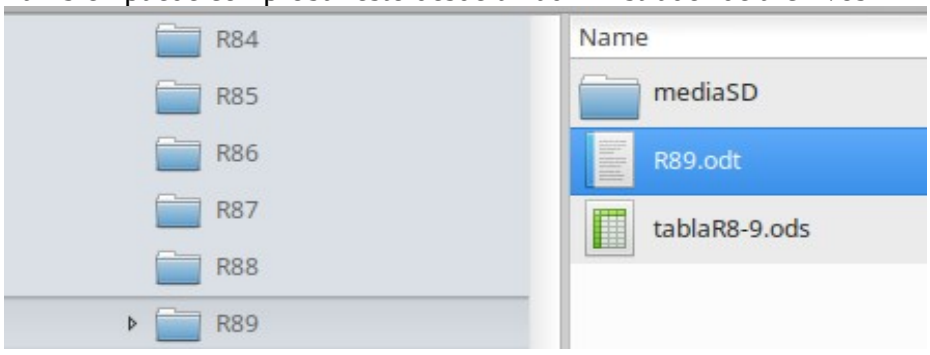
Como podemos ver nos indica que ha realizado varias tareas, finalizando con ***Done***. Esto indica que todo ha sido exitoso.

La función package.skeleton() ha creado dentro del directorio en que se halla nuestro espacio de trabajo, una carpeta llamada mediaSD, que es el nombre que le dimos al paquete, puede comprobarlo con la función dir()

```
> dir()
[1] "mediaSD"       "R89.odt"       "tablaR8-9.ods"
```

vemos que tenemos en el directorio la carpeta mencionada, además de una archivo de texto y una planilla de cálculo, que son los archivos para el desarrollo de la clase R89. Estos dos archivos no es necesario que usted los tenga en la carpeta que estamos trabajando.

También puede comprobar esto desde un administrador de archivos

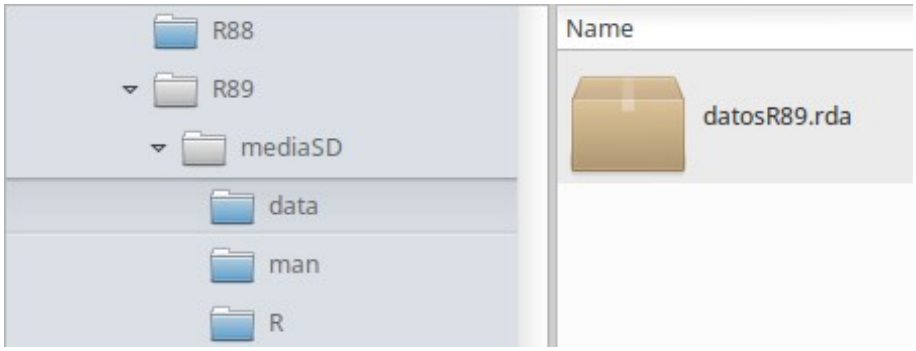


Además podemos ver que dentro del directorio: mediaSD, hay tres carpetas: data, man y R

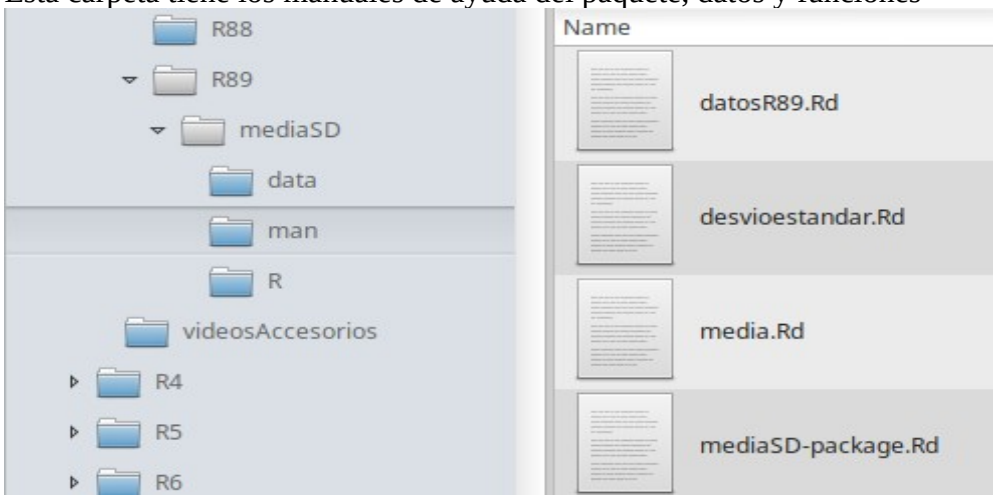
Veamos que contiene cada una de ellas

Carpeta: data

Esta carpeta contiene una archivo llamado datosR89.rda. Este archivo contiene el data.frame datosR89, con extensión .rda, que es equivalente a .RData. Este archivo viajará con el paquete y estará disponible para quien utilice el paquete, siendo especialmente útil para el archivo de ayuda.

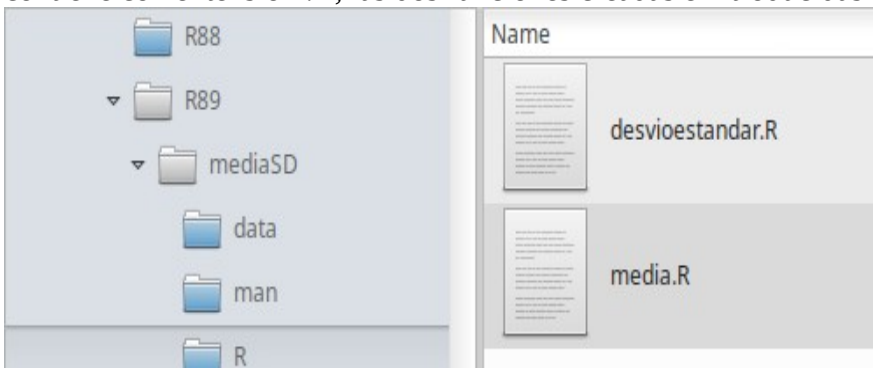


Carpeta: man
Esta carpeta tiene los manuales de ayuda del paquete, datos y funciones



El archivo mediaSD-package.Rd, tiene datos sobre el paquete
datosR89.Rd: información sobre los datos
desvioestandar.Rd: información sobre uso y propiedades de la función desvioestandar
media.Rd: información sobre la función media.
Estos 4 archivos requieren posterior modificación. Tema que veremos más adelante.

Carpeta: R
contiene con extensión .R, las dos funciones creadas e introducidas en el espacio de trabajo.

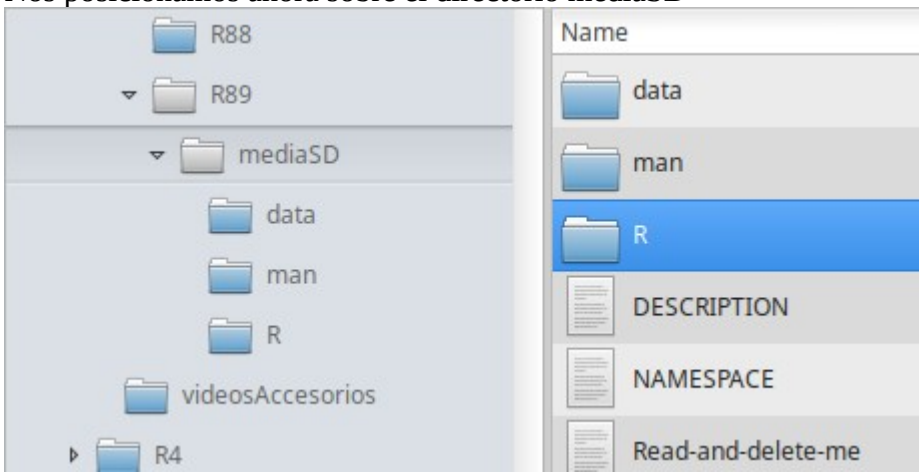


Si hace click sobre ellas, por ejemplo en media.R, se abrirá el archivo en un procesador de texto, mostrándonos el código de la función. Linux automáticamente detecta que es un archivo .R y nos muestra la función resaltando en colores diferentes sus partes.

```
File  Edit  Search  View  Document  Help
 1 media <-
 2 function(x,removerNA=FALSE){
 3 if(removerNA==TRUE){
 4 x<-x[!is.na(x)]
 5 }
 6 print("Sus datos son: ")
 7 print(x)
 8 n<-length(x)
 9 resultado<-sum(x)/n
10 print(paste("la media de sus datos es: ", resultado))
11 }
12
```

Comenzando con edición de archivos .Rd

A continuación editaremos algunos archivos creados dentro del directorio mediaSD
Nos posicionamos ahora sobre el directorio mediaSD



vemos además de las carpetas mencionadas, un archivo Read-and-delete-me. Para continuar vemos su contenido, haciendo doble click sobre él, lo que determinará que se abra con un editor de textos. Su contenido es

* Edit the help file skeletons in 'man', possibly combining help files for multiple functions.
* Edit the exports in 'NAMESPACE', and add necessary imports.
* Put any C/C++/Fortran code in 'src'.
* If you have compiled code, add a useDynLib() directive to 'NAMESPACE'.
* Run R CMD build to build the package tarball.
* Run R CMD check to check the package tarball.

Read "Writing R Extensions" for more information.

Como indica el archivo Read-and-delete-me, tenemos varios pasos que seguir
item 1: Edit the help files in man. Dejando la información deseada.
La función package.skeleton() con la que creamos el paquete, genera los archivos de ayuda, pero estos requieren modificación. Los archivos generados tienen partes generales y otras que hay que corregir.
En este caso haremos nuestros propios archivos de ayuda, utilizando los códigos adecuados. Veremos esto más adelante en el título modificación de archivos .Rd

item2: Edit the exports in 'NAMESPACE', and add necessary imports.
chequearemos el correcto nombre de los archivos a exportar en el directorio NAMESPACE
Veremos luego en detalle este ítem.

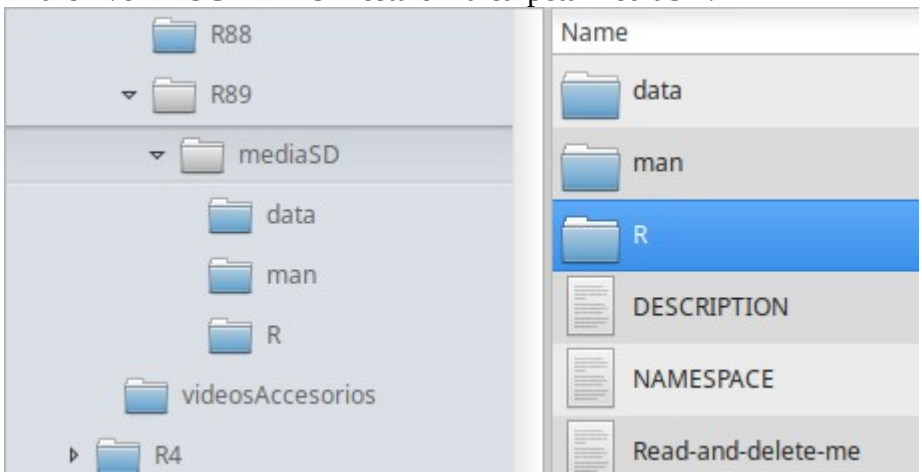
los items 3 y 4 no corresponden hacerlos en este caso ya que no tenemos dependencias o códigos en C o Fotran

Los item 5 y 6 los veremos luego de modificar los archivos de ayuda y explicaciones.

Modificación de archivos .Rd

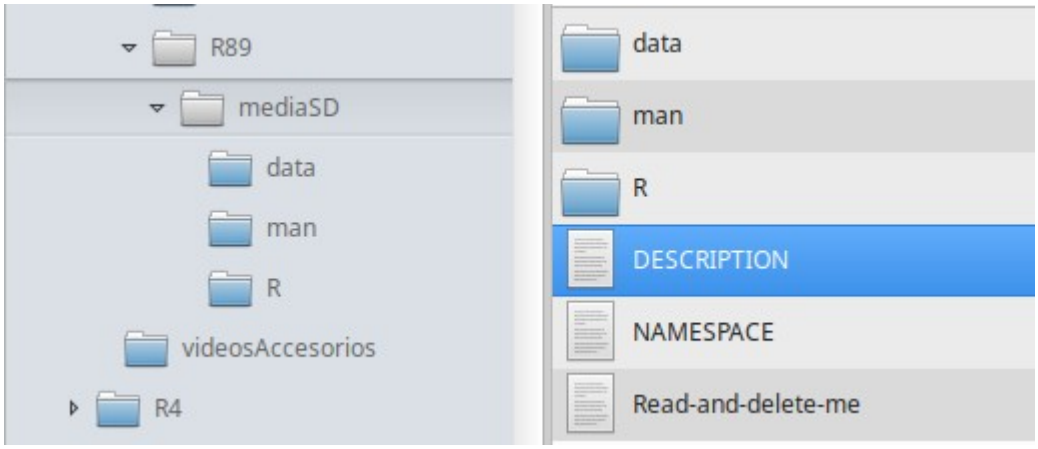
Los archivos .Rd tienen información que será procesada luego de la instalación del paquete y a la que accederemos en general con la función help(), y por lo tanto nos servirá de ayuda.
El archivo DESCRIPTION

El archivo DESCRIPTION está en la carpeta mediaSD.

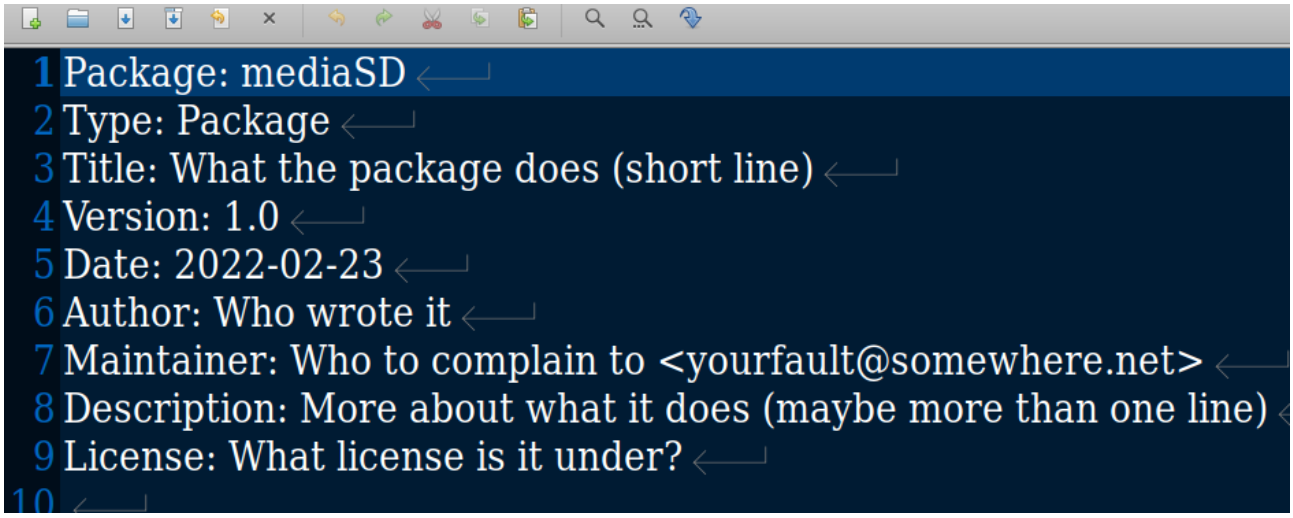


Este archivo se carga con los demás archivos del paquete y da una breve descripción del mismo. Es obligatorio que contenga ciertos datos que genera automáticamente package.skeleton(): Package, Version, Description, License, title, author and maintainer son campos obligatorios. Este archivo por su simplicidad lo modificaremos directamente en el paquete.
Cada campo tiene el formato
Keyword: Value
Siempre va primero la keyword, por ejemplo Title. seguido de dos puntos y un espacio.
por ejemplo
Type: Package

Busque el archivo DESCRIPTION en la carpeta mediaSD y haga doble click sobre él abriéndolo con un editor de textos

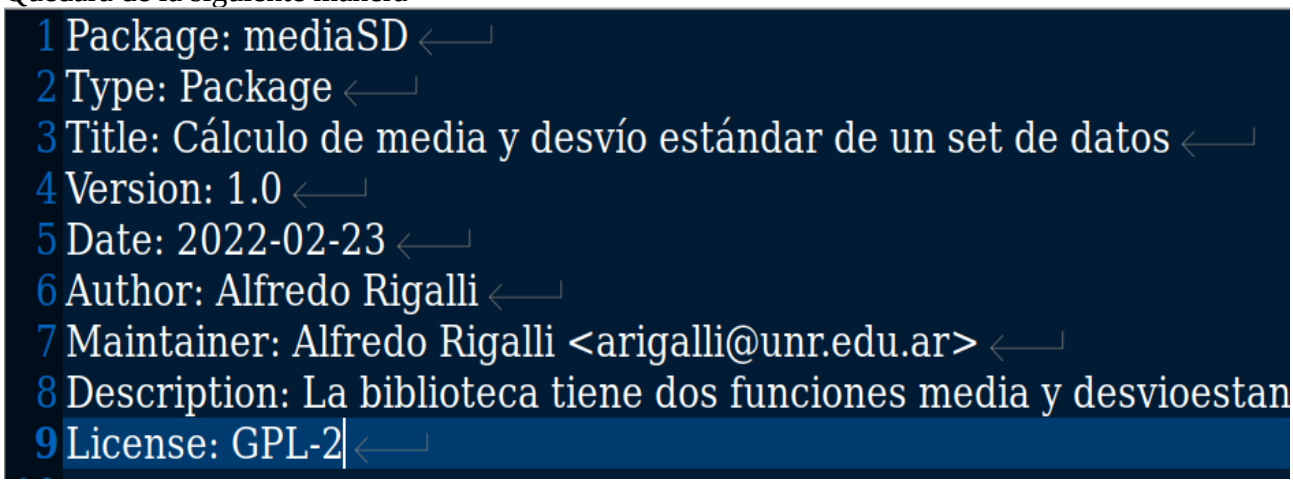


al hacer doble click hallará



Como puede ver, falta aclarar y completar algunos ítems
Title:
Author
Maintainer
Description
License:

Entonces, lo que hacemos es corregir y agregar esta información y luego guardamos los cambios introducidos.
Quedará de la siguiente manera



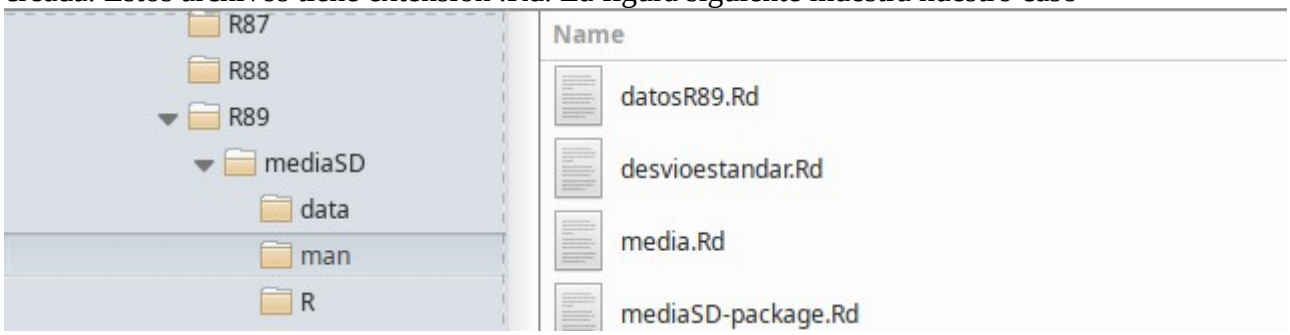
Si lo desea puede copiar el texto que se halla a continuación resaltado en amarillo y reemplazar completamente el contenido del archivo DESCRIPTION.
Atención: no es conveniente usar letras con acento: á, í, no son del codigo ASCII y dará errores a la hora de cargar el paquete y realizar los archivos tar.gz

Package: mediaSD
Type: Package
Title: Calculo de media y desvio estandar de un set de datos
Version: 1.0
Date: 2022-02-23
Author: Alfredo Rigalli
Maintainer: Alfredo Rigalli <arigalli@unr.edu.ar>
Description: La biblioteca tiene dos funciones media y desvioestandar, que calculan la media y el desvio estandar de un set de datos
License: GPL-2

La versión será siempre una secuencia de al menos dos números, separados por puntos o guiones.
El título no debe superar 65 caracteres.
Maintainer debe contener un solo nombre y un email en formato como indicado en la figura.
Autores: puede contener el número de autores que se desee.
Description: no tiene límite de caracteres
License: si se enviará el paquete a CRAN, se sugiere que sea GPL-2 (general public license version 2)
Este archivo sigue un formato llamado Debian-control-file, desarrollado por la distribución de Linux: Debian.

Modificación carpeta man

La carpeta man, tiene archivos con documentación sobre los datos y las funciones de la biblioteca creada. Estos archivos tiene extensión .Rd. La figura siguiente muestra nuestro caso



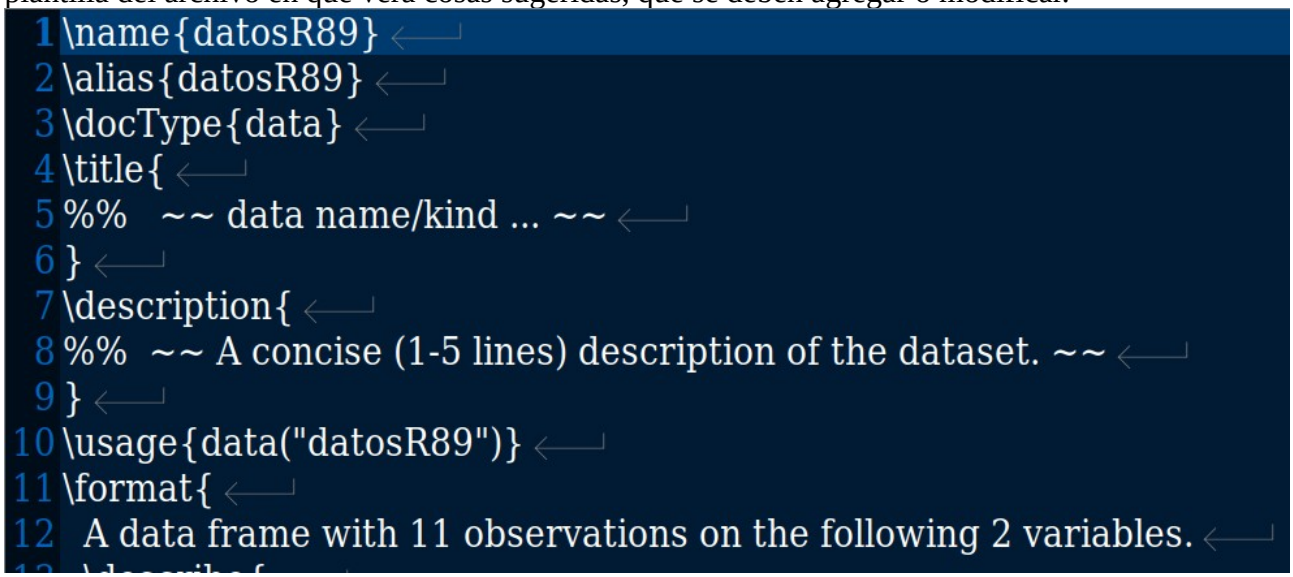
La función package.skeleton() crea estos archivos con partes que deben ser modificadas. Para modificar el archivo se deben seguir algunos lineamientos sobre el código a utilizar y ubicación de las líneas:

Las lineas comienzan con \
cada bloque de información va entre {}
Las líneas que comienzan con %% son comentarios y se pueden eliminar.
Se pueden establecer títulos con el siguiente formato
\ uso{descripción del uso}
\ autor{Alfredo Rigalli}
\ ejemplos{
data(datos)

media(datos$x,na.rm=TRUE)
desvioestandar(datos$y)
}

Modificación del archivo datosR89.Rd

Si hace doble click sobre el archivo datosR89.Rd, que se halla en el directorio man, hallará una plantilla del archivo en que verá cosas sugeridas, que se deben agregar o modificar.



Modificaremos entonces los archivos de una manera mínima para que se pueda instalar el paquete. Si hay errores en el código de escritura, como llaves faltantes, etc, se abortará la instalación.
Así. dejaremos los archivos con la siguiente estructura, resaltada en amarillo.
Sugerimos copiar el texto resaltado en amarillo a continuación y reemplazar todo el contenido del archivo datosR89.rda

\name{datosR89}
\alias{datosR89}
\docType{data}
\title{datosR89 ejemplo}
\description{
Funciones para calculo de media y desvio estandar de un set de datos}
\usage{
data("datosR89")
}
\format{
  Un data.frame con 11 filas y 2 variables.
  x: variable numerica
  y: variable numerica
}
\examples{
data(datosR89)
}
\keyword{datasets}

luego guardar el archivo sin cambiar su nombre.

Modificación del archivo desvioestandar.Rd

hacemos modificaciones dejando mínima información. Podemos dejarlo de la manera siguiente, resaltada en amarillo. Copie el texto y reemplace el contenido del archivo desvioestandar.Rd

```
\name{desvioestandar}
\alias{desvioestandar}
\title{Desvio Estandar
}
\description{
Calcula el desvio estandar de un set de datos
}
\usage{
desvioestandar(x, removerNA = FALSE)
}
\arguments{
  \item{x}{
vector numerico o columna de un data.fram
}
  \item{removerNA}{
argumento seteado en FALSE por defecto. Es decir al aplicar la función no remueve los valores NA
}
}
\value{
devuelve el set de datos utilizados y el valor del desvio estandar
}
\examples{
          # para los datos datosR89$y que no tiene NA
          desvioestandar(datosR89$y)
          #salida
          "Sus datos son:"
           1.0  2.0  3.1  3.9  4.0  5.0  6.2  6.9  7.9  9.0 10.1
          "es desvio estandar es:  2.91138829739041"

          # para los datos datosR89$x que  tiene NA
           desvioestandar(datosR89$x)
          # salida
          "Sus datos son:"
           0  1  3  3  1 NA  2  7  1  4 10
          "es desvio estandar es:  NA"
          desvioestandar(datosR89$x,removerNA=TRUE)
          salida
          "Sus datos son:"
           0  1  3  3  1  2  7  1  4 10
           "es desvio estandar es:  3.11982905514602"
}
\keyword{manip}
```

Guarde las modificaciones del archivo sin cambiar el nombre del mismo.

El último item es una palabra clave de R, que puede buscarse ejecutando desde la línea de comando de R

```
> RShowDoc("KEYWORDS")
```

mostrará una lista, de la que puede elegir una o más palabras. Si hubiera más de una keyword, se deben escribir en lineas sucesivas

```
\keyword{kw1}
\keyword{kw2}
etc.
```

Modificación del archivo media.Rd

Copie el texto resaltado en amarillo y reemplacelo por el que se halla en el archivo media.Rd

```
\name{media}
\alias{media}
\title{Media
}
\description{
calcula la media de un set de datos numericos
}
\usage{
media(x, removerNA = FALSE)
}
\arguments{
  \item{x}{
un vector numerico o una columna de un data.frame
}
  \item{removerNA}{
argumento seteado en FALSE por defecto. Es decir al aplicar la función no remueve los valores NA
}
}
\value{
devuelve el set de datos utilizados y el valor de la media
}
\examples{
        # calculo de media de un set de datos con valores NA
        media(datosR89$x,removerNA=TRUE)
        "Sus datos son: "
        0  1  3  3  1  2  7  1  4 10
        "la media de sus datos es:  3.2"
}
\keyword{manip}
```

Modificación del archivo mediaSD-package.Rd

copie el texto resaltado en amarillo y reemplace el contenido del archivo mediaSD-package.Rd

```
\name{mediaSD-package}
\alias{mediaSD-package}
\alias{mediaSD}
```

\docType{package}
\title{
\packageTitle{mediaSD}
}
\description{
\packageDescription{mediaSD}
}
\details{

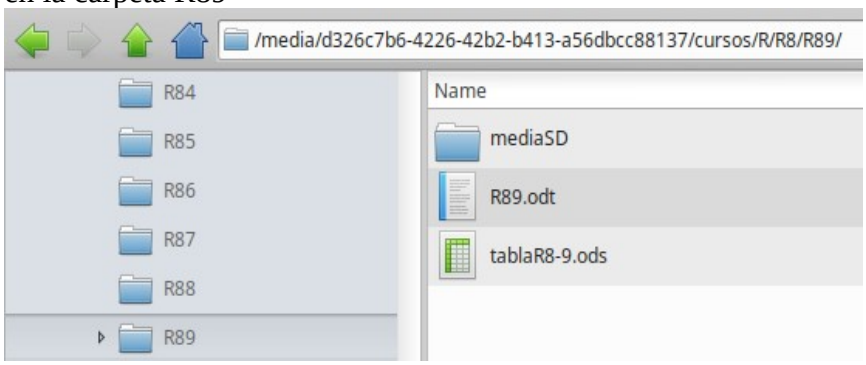
The DESCRIPTION file:
\packageDESCRIPTION{mediaSD}
\packageIndices{mediaSD}
}
\author{
\packageAuthor{mediaSD}
Maintainer: \packageMaintainer{mediaSD}
}
\keyword{package}

Ya tenemos todo listo en nuestro paquete. Veamos la instalación del mismo en nuestra computadora.

Control de archivo NAMESPACE

Editamos con un procesador de texto el archivo NAMESPACE que se halla en la carpeta mediaSD
En este archivo figuran los datos y funciones a exportar. En nuestro caso son los datos datosR89.rda y las funciones desvioestandar.R y media.R. Entonces en el texto del archivo debe figurar:
export("datosR89", "desvioestandar", "media")
De ser asi estamos en el camino correcto
Instalación del paquete en la computadora

Una vez que hemos modificado los archivo .Rd del directorio man, nos colocamos fuera de R, en el directorio que contiene la carpeta mediaSD, contendiendo los archivos del paquete, en nuestro caso en la carpeta R89



alfredo@alfredo-System-Product-Name:/media/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R8/R89$
y ejecutamos desde la consola el comando indicado que tiene R CMD INSTALL luego el camino de directorios y finalmente el nombre de la carpeta que contiene el código fuente del paquete. Cada parte se marca en un color diferente

```
alfredo@alfredo-System-Product-Name:/media/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R8/R89$ R CMD INSTALL /media/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R8/R89/mediaSD
* installing to library ‘/home/alfredo/R/x86_64-pc-linux-gnu-library/3.4’
* installing *source* package ‘mediaSD’ ...
** R
** data
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (mediaSD)
```

Posibles errores: Puede ocurrir que no todo sea tan fácil y tengamos más de un error que termine no instalando el paquete. Veamos algunos errores. Si luego de la ejecución anterior obtiene algo así:

```
* installing to library ‘/home/alfredo/R/x86_64-pc-linux-gnu-library/4.1’
* installing *source* package ‘mediaSD’ ...
** using staged installation
** R
** data
** byte-compile and prepare package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded from temporary location
Error: package or namespace load failed for ‘mediaSD’ in namespaceExport(ns, exports):
 undefined exports: datosR89
Error: loading failed
Execution halted
ERROR: loading failed
* removing ‘/home/alfredo/R/x86_64-pc-linux-gnu-library/4.1/mediaSD’
```

Vemos que todo venía bien hasta que aparece la nefasta palabra: Error
Nos dice que el paquete o el namespace ha fallado y en este caso la falla es "undefined exports: datosR89
Es evidente que el archivo datosR89.rda del directorio data, tiene algo raro.
Una solución es abrir el archivo NAMESPACE con un editor y tendremos
export("datosR89","desvioestandar", "media")
elimine datosR89 de manera que quede
export("desvioestandar", "media")
guarde los cambios e intente nuevamente el código

```
alfredo@alfredo-System-Product-Name:/media/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R8/R89$ R CMD INSTALL /media/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R8/R89/mediaSD
```

Si al final aparece DONE (mediaSD)
se ha instalado el paquete aunque quizás tenga algun problema con las ayudas.

Prueba de instalación del paquete

Hacemos una prueba de instalación. Para ello también nos ubicamos en el prompt de Linux en el directorio donde esta contenida la carpeta mediaSD y ejecutamos

```
alfredo@alfredo-System-Product-Name:/media/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R8/R89$ R CMD check mediaSD
```

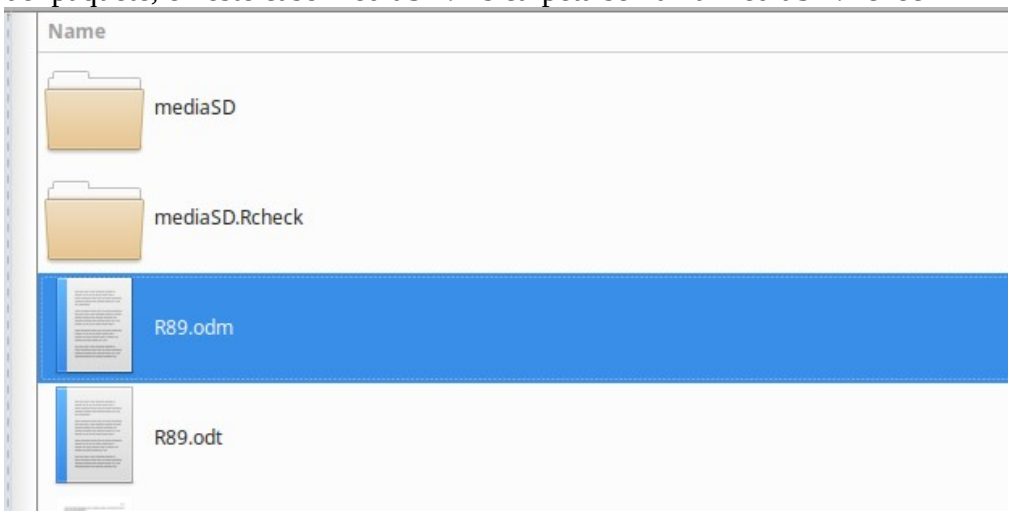
obtenemos una larga salida en que se van chequeando ciertos aspectos. En su mayoría figura OK. Y hay algunos warnings, debido a caracteres no ASCI, formatos y uso del procesador Latex

```
* using log directory ‘/media/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R8/R89/mediaSD.Rcheck’
* using R version 3.4.4 (2018-03-15)
* using platform: x86_64-pc-linux-gnu (64-bit)
* using session charset: UTF-8
* checking for file ‘mediaSD/DESCRIPTION’ ... OK
* checking extension type ... Package
* this is package ‘mediaSD’ version ‘1.0’
* checking package namespace information ... OK
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking if there is a namespace ... OK
* checking for executable files ... OK
* checking for hidden files and directories ... OK
* checking for portable file names ... OK
* checking for sufficient/correct file permissions ... OK
* checking whether package ‘mediaSD’ can be installed ... OK
* checking installed package size ... OK
* checking package directory ... OK
* checking DESCRIPTION meta-information ... NOTE
Malformed Description field: should contain one or more complete sentences.
Checking should be performed on sources prepared by ‘R CMD build’.
* checking top-level files ... OK
* checking for left-over files ... OK
* checking index information ... OK
* checking package subdirectories ... OK
* checking R files for non-ASCII characters ... WARNING
Found the following file with non-ASCII characters:
  desvioestandar.R
Portable packages must use only ASCII characters in their R code,
except perhaps in comments.
Use \uxxxx escapes for other characters.
* checking R files for syntax errors ... OK
* checking whether the package can be loaded ... OK
* checking whether the package can be loaded with stated dependencies ... OK
* checking whether the package can be unloaded cleanly ... OK
* checking whether the namespace can be loaded with stated dependencies ... OK
* checking whether the namespace can be unloaded cleanly ... OK
* checking loading without being on the library search path ... OK
* checking dependencies in R code ... OK
* checking S3 generic/method consistency ... OK
* checking replacement functions ... OK
* checking foreign function calls ... OK
```

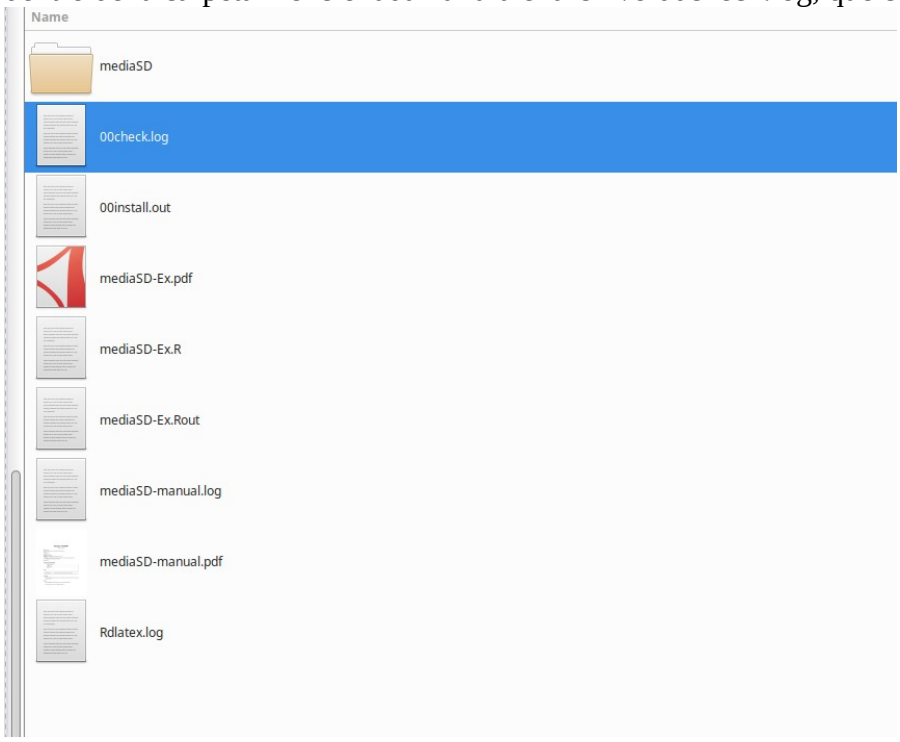
```
* checking R code for possible problems ... OK
* checking Rd files ... WARNING
desvioestandar.Rd: non-ASCII input and no declared encoding
media.Rd: non-ASCII input and no declared encoding
problems found in ‘desvioestandar.Rd’, ‘media.Rd’
* checking Rd metadata ... OK
* checking Rd cross-references ... OK
* checking for missing documentation entries ... OK
* checking for code/documentation mismatches ... OK
* checking Rd \usage sections ... OK
* checking Rd contents ... OK
* checking for unstated dependencies in examples ... WARNING
Warning: parse error in file 'mediaSD-Ex.R':
40:9: unexpected '['
39:         #salida
40:         [
            ^
* checking contents of ‘data’ directory ... OK
* checking data for non-ASCII characters ... OK
* checking data for ASCII and uncompressed saves ... OK
* checking examples ... ERROR
Running examples in ‘mediaSD-Ex.R’ failed
The error most likely occurred in:

> ### Name: desvioestandar
> ### Title: Desvio Estandar
> ### Aliases: desvioestandar
> ### Keywords: manip
>
> ### ** Examples
>
>         # para los datos datosR89$y que no tiene NA
>         desvioestandar(datosR89$y)
[1] "Sus datos son:"
Error in print(x) : object 'datosR89' not found
Calls: desvioestandar -> print
Execution halted
* checking PDF version of manual ... WARNING
LaTeX errors when creating PDF version.
This typically indicates Rd problems.
* checking PDF version of manual without hyperrefs or index ... ERROR
Re-running with no redirection of stdout/stderr.
Hmm ... looks like a package
Error in texi2dvi(file = file, pdf = TRUE, clean = clean, quiet = quiet,  :
  pdflatex is not available
Error in texi2dvi(file = file, pdf = TRUE, clean = clean, quiet = quiet,  :
  pdflatex is not available
Error in running tools::texi2pdf()
You may want to clean up by 'rm -rf /tmp/Rtmp1R6g4f/Rd2pdf1c21756a6f80'
* DONE
```

Status: 2 ERRORs, 4 WARNINGs, 1 NOTE
See
‘/media/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R8/R89/mediaSD.Rcheck/00check.log’
for details.

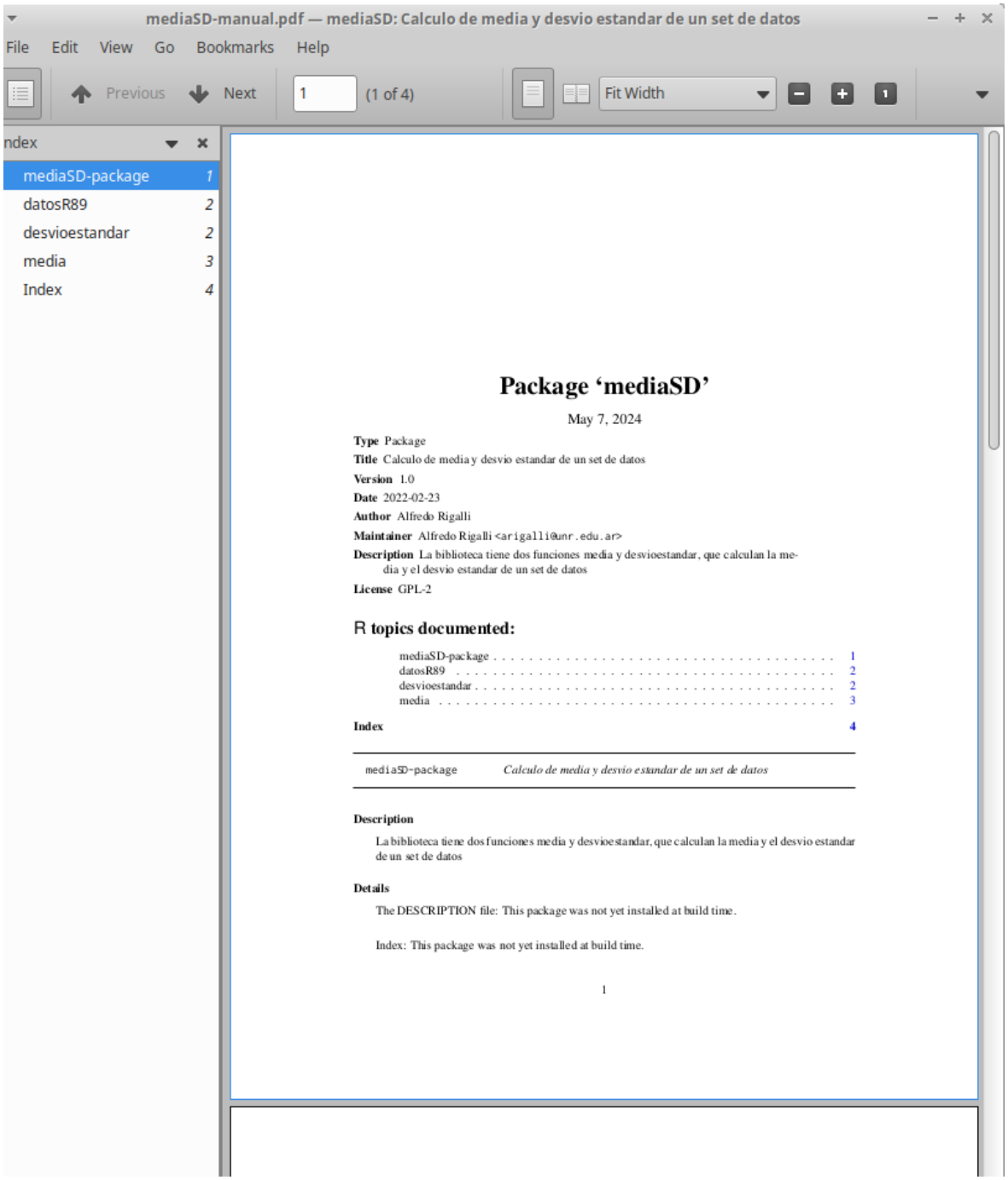
Mientras en la línea Status, que se halla al final de la salida mostrada, figure un número finito de ERRORs, no estará listo aun el paquete para su funcionamiento correcto. Por lo tanto debemos evaluar con detenimiento cada línea de la salida mostrada. Si lo desea lo puede abrir como un archivo que se halla en una nueva carpeta que hallará en el mismo directorio que se halla la carpeta del paquete, en este caso mediaSD. Es carpeta se llama mediaSD.Rcheck



dentro de la carpeta mencionada hallara el archivo 00check.log, que contiene la misma salida



en esta misma carpeta hallará el archivo mediaSD-manual.pdf, que es el manual incluyendo todos los detalles de los archivos de la carpeta man.



Carga de la biblioteca

Para ello ingresamos a un nuevo espacio de trabajo en cualquier sitio que no tenga las funciones del paquete. Básicamente cualquier directorio fuera de R89
Por ejemplo ingresamos en el directorio raiz

```
alfredo@alfredo-System-Product-Name:~$ R
```

si pedimos help de las funciones media y desvioestandar creadas en el paquete, vemos que no tenemos ayuda

```
> help(media)
```

No documentation for ‘media’ in specified packages and libraries:
you could try ‘??media’

```
> help(desvioestandar)
```

No documentation for ‘desvioestandar’ in specified packages and libraries:
you could try ‘??desvioestandar’
Esto se debe a que que el paquete está instalado pero no cargado en este espacio de trabajo
Carguemosló

```
> library(mediaSD)
```

pidamos help de media

```
> help(media)
```

obtendremos

```
Description:

     calcula la media de un set de datos numericos

Usage:

     media(x, removerNA = FALSE)

Arguments:

       x: un vector numerico o una columna de un data.frame

removerNA: argumento seteado en FALSE por defecto. Es decir al aplicar
          la función no remueve los valores NA

Value:

     devuelve el set de datos utilizados y el valor de la media

Examples:

             # calculo de media de un set de datos con valores NA
             media(datosR89$x,removerNA=TRUE)
             "Sus datos son: "
             0  1  3  3  1  2  7  1  4 10
             "la media de sus datos es:  3.2"


~
(END)
```

Podemos cargar los datos del data.frame datosR89

```
> data(datosR89)
```

que podemos verificar que son los datos de nuestro paquete

```
> datosR89
```

	x	y
1	0	1.0
2	1	2.0
3	3	3.1
4	3	3.9
5	1	4.0

6 NA 5.0
7   2 6.2
8   7 6.9
9   1 7.9
10 4 9.0
11 10 10.1

provemos nuestra función media

```
> media(datosR89$x)
```

[1] "Sus datos son: "
 [1] 0 1 3 3 1 NA 2 7 1 4 10
[1] "la media de sus datos es: NA"
que nos da NA ya que el argumento removerNA=FALSE
cambie entonces el argumento removerNA a TRUE

```
> media(datosR89$x,removerNA=TRUE)
```

[1] "Sus datos son: "
 [1] 0 1 3 3 1 2 7 1 4 10
[1] "la media de sus datos es: 3.2"

Preparación del archivo tar.gz

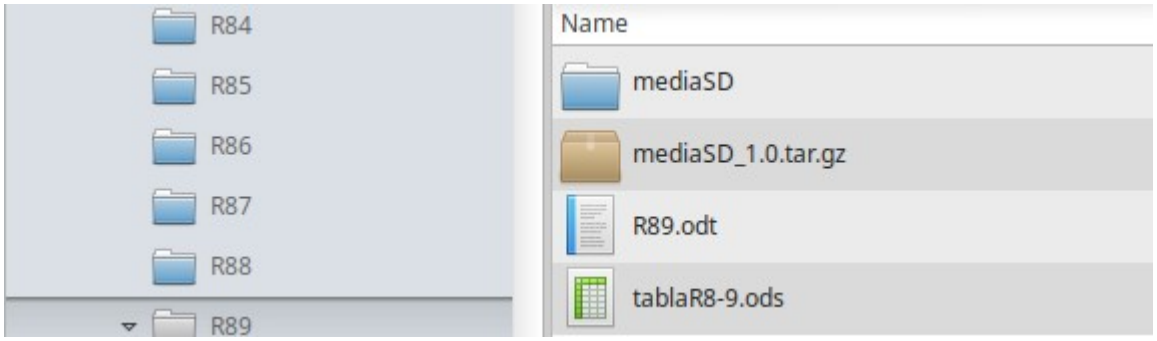
El archivo tar.gz, es el que habitualmente descargamos de los repositorios y utilizamos con la función install.packages(). Son archivos fuente que hemos utilizado a lo largo de todos los módulos
Para su creación nos ubicamos en el directorio donde se halla la carpeta mediaSD, fuera de R en el prompt de Linux y ejecutamos

```
alfredo@alfredo-System-Product-Name:/media/d326c7b6-4226-42b2-b413-a56dbcc88137/
cursos/R/R8/R89$ R CMD build mediaSD
```

* checking for file ‘mediaSD/DESCRIPTION’ ... OK
* preparing ‘mediaSD’:
* checking DESCRIPTION meta-information ... OK
* installing the package to process help pages
* saving partial Rd database
* checking for LF line-endings in source and make files and shell scripts
* checking for empty or unneeded directories
* looking to see if a ‘data/datalist’ file should be added
* building ‘mediaSD_1.0.tar.gz’

podemos comprobar que en el directorio en que estamos trabajando se halla el paquete mediaSD_1.0.tar.gz

este paquete podrá enviarlo a quien usted desee para que lo instale en su computadora.



Probemos a instalarlo en nuestra computadora. Ingrese al espacio de trabajo donde se halla el archivo tar.gz y primero removeremos el paquete que instalamos recientemente con

```
> remove.packages("mediaSD")
```

Removing package from '/home/alfredo/R/x86_64-pc-linux-gnu-library/3.4'
(as 'lib' is unspecified)

Como puede comprobar, el paquete ya no está en nuestra computadora

```
> library(mediaSD)
```

Error in library(mediaSD) : there is no package called 'mediaSD'

puede probar a instalarlo a partir de tar.gz, como si lo hubiera bajado del repositorio
Coloque el archivo tar.gz en Downloads y ejecute


```
>
install.packages("~/Downloads/mediaSD_1.0.tar.gz",repos=NULL,type="source",depedencies=TR
UE)
```

Installing package into '/home/alfredo/R/x86_64-pc-linux-gnu-library/3.4'
(as 'lib' is unspecified)
* installing *source* package 'mediaSD' ...
** R
** data
** preparing package for lazy loading
** help

*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (mediaSD)

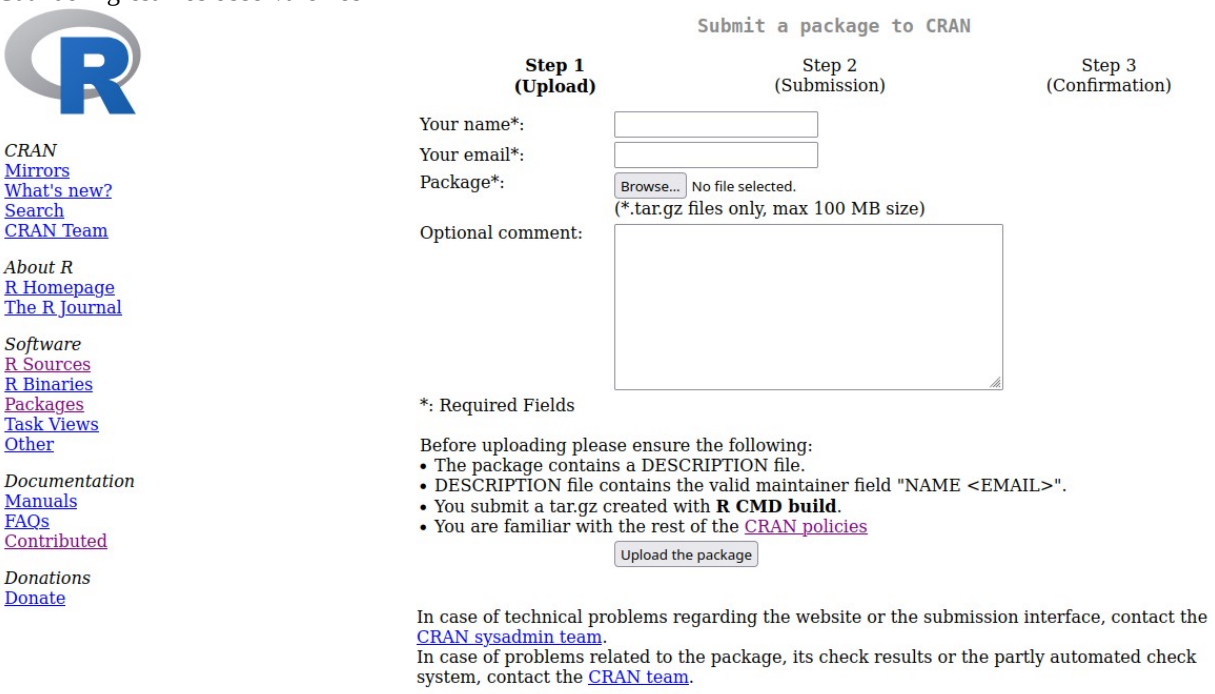
La instalación a partir del archivo fuente ha funcionado. Si lo envía a alguien lo podrá hacer.

Instalación y chequeo de funcionamiento en Windows

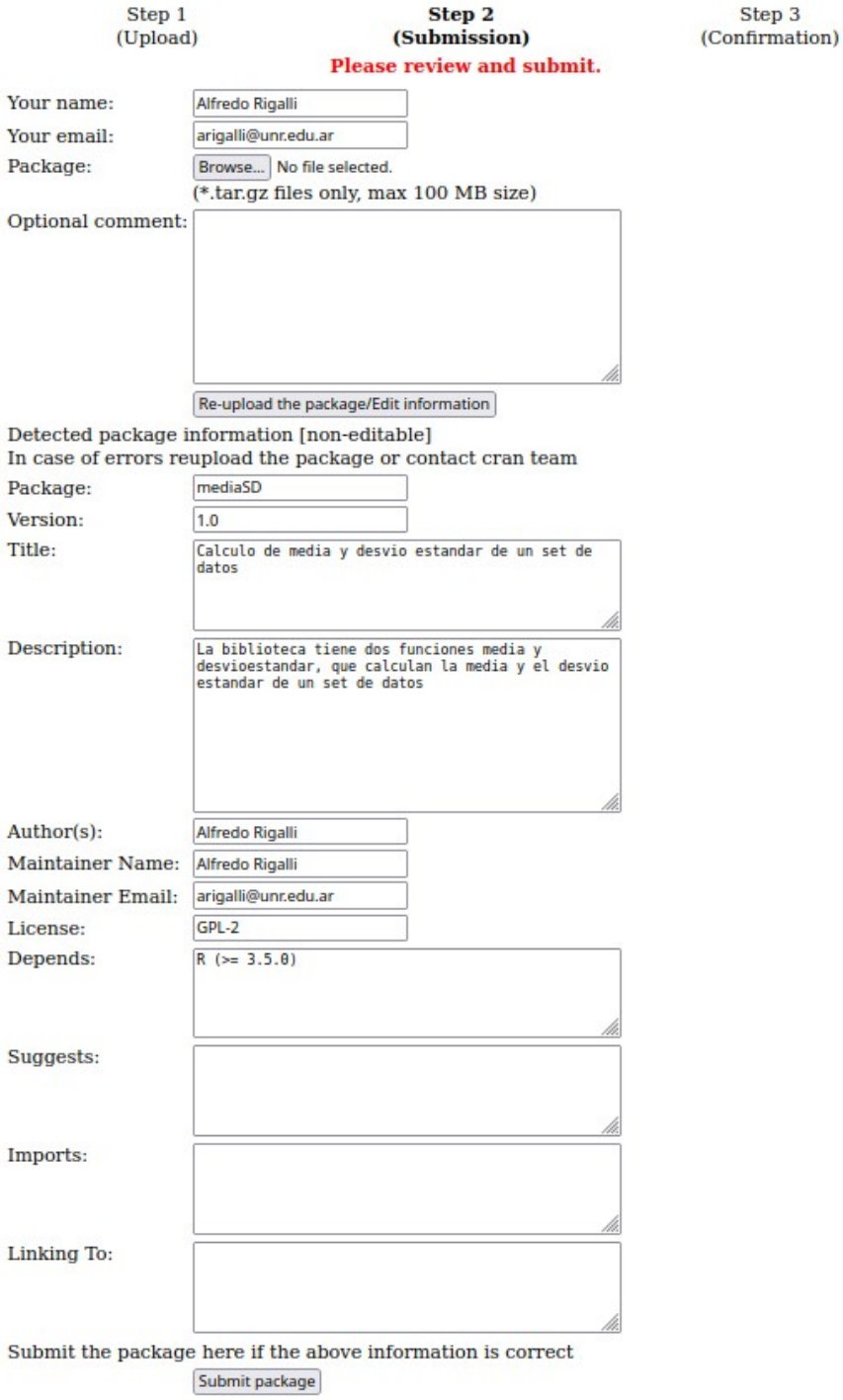
Si trabaja en Linux tendrá todas las herramientas para los pasos de creación, chequeo e instalación. Si su sistema operativo es Windows deberá incorporar algunas herramientas adicionales.
Para usuarios de Windows, se recomienda instalar RtoolsXX.exe, que se descarga de http://www.murdoch-sutherland.com/Rtools/.
Puede también hallar ayuda en el artículo: Creating R Packages: A Tutorial. Friedrich Leisch. 2009. https://cran.r-project.org/doc/contrib/Leisch-CreatingPackages.pdf

Envio del paquete a RCRAN

Por cierto esta es la etapa más esperada y quizás difícil. Lograr que el paquete sea aceptado en los repositorios de R
Una vez que se ha pasado exitosamente los pasos
1- creación del paquete con la función package.skeleton()
2- El chequeo con R CMD check
3- se ha creado el tarball o archivo tar.gz con la función build
podremos entonces subirlo al repositorio. Para ello accedemos al sitio de carga en la siguiente dirección y cargamos los datos exigidos. El sitio es: https://CRAN.R-project.org/submit.html
Cuando ingresamos observaremos



Son campos obligatorios: name, email y el archivo tar.gz. Podemos agregar una descripción adicional. Luego solo hay que oprimir Upload the package y pasaremos a otra ventana que nos mostrará los datos del paquete, al que debemos revisar y controlar, si falta algo.



Si todo está bien oprimimos Submit package y será enviado. No será aceptado hasta que el Maintainer del paquete reciba un mail y acepte su función.
Por supuesto puede haber correcciones que hacer que serán informadas por el sistema a través de mail.
El Maintainer recibirá los mails con las sugerencias de corrección, en primer lugar de manera automática y luego de pasado esta corrección, recibirá correcciones manuales realizadas por un revisor. Realizadas las correcciones, se debe subir nuevamente el paquete a través de la misma página mostrada más arriba. Por supuesto no se olvide de ejecutar primero la función check y luego build para obtener el nuevo archivo .tar.gz.