

```
library(dplyr)  
library(magrittr)
```

```
rladies_global %>% filter(city == 'Santa Fe')
```



# ALGORITMOS BÁSICOS DE MACHINE LEARNING

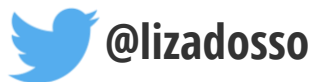
12 de Septiembre 2018 @ Club de Emprendedores, Santa Fe

# ¡ANUNCIOS!



**¡NUEVA CO-ORGANIZADORA!**

Dra. Lizza Dosso  
Ingeniería Química  
Postdoc en INCAPE



**santa fe**  
**ACTIVA**  
emprendedores e innovación

**Se viene R-Ladies**  
**Rosario**



# ¿QUÉ ES MACHINE LEARNING?



Machine Learning es la ciencia de lograr que las computadoras actúen sin que sean **explícitamente programadas** para ello.



**APRENDIZAJE  
SUPERVISADO**

**APRENDIZAJE  
NO SUPERVISADO**



# ¿QUÉ ES MACHINE LEARNING?

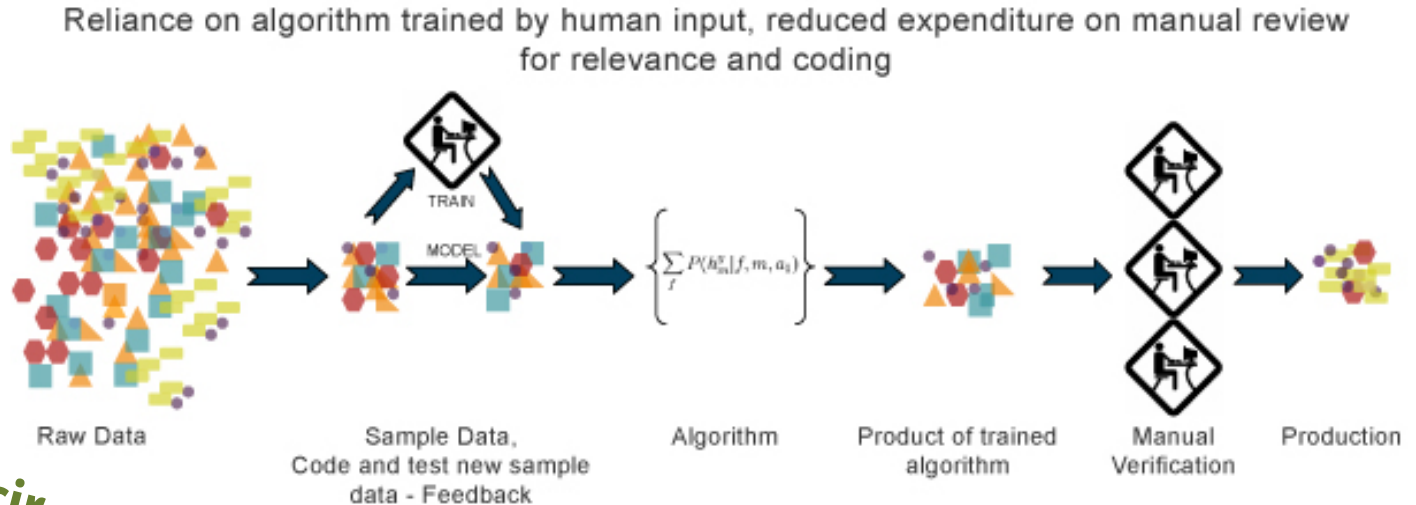


Machine Learning es la ciencia de lograr que las computadoras actúen sin que sean **explícitamente programadas** para ello.



## APRENDIZAJE SUPERVISADO

- 1) Predecir
- 2) Clasificar

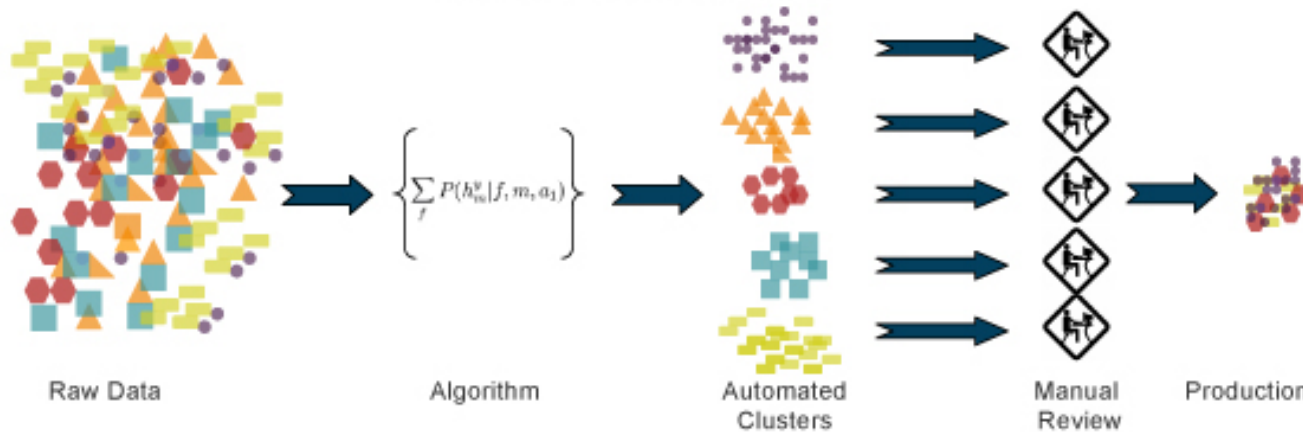


# ¿QUÉ ES MACHINE LEARNING?



Machine Learning es la ciencia de lograr que las computadoras actúen sin que sean **explícitamente programadas** para ello.

High reliance on algorithm for raw data, large expenditure on manual review for review for relevance and coding



**APRENDIZAJE  
NO SUPERVISADO**

*Inferir una función o algoritmo que  
etiqueta datos que no conocemos.*



# ÁRBOLES DE DECISIÓN





# ¿QUÉ SON LOS ÁRBOLES DE DECISIÓN?



El conjunto de datos **se divide** en base a preguntas.  
Las mejores preguntas se elijen usando un criterio definido.

## Aprendizaje Supervisado Árboles de Regresión y de Clasificación



- ✓ Datos numéricos y categóricos
- ✓ Interpretación sencilla y directa
- ✓ Algoritmos muy rápidos



- ✗ Requiere un ajuste preciso con poda
- ✗ Inestable a las variaciones de datos
- ✗ No modela fácilmente ciertas relaciones

Elegir preguntas que **acoten**  
**significativamente** el espacio  
de búsqueda.



# EJEMPLO: ÁRBOL DE CLASIFICACIÓN

¿Conviene aceptar esa oferta de trabajo?

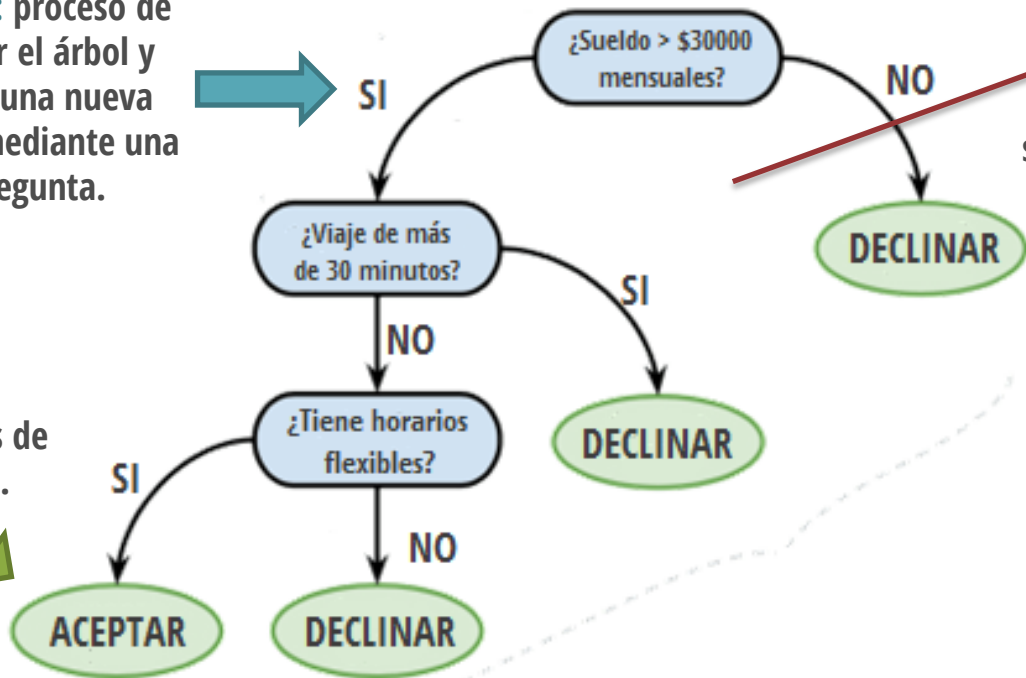


**Raíz:** población completa del caso. Todas las observaciones existentes.

**Dividir:** proceso de dividir el árbol y crear una nueva rama mediante una pregunta.

**Poda:** remover subnodos completos

**Hojas:** resultados de la clasificación.





# ÁRBOLES EN R: CLASIFICACIÓN



Librería principal: **rpart**

Script disponible en: <https://tinyurl.com/y7ctlbmd>

1. Importar librerías y datos.
2. Crear o entrenar el árbol.
3. Examinar los resultados.
4. Podar el árbol.

	Kyphosis <sup>△</sup>	Age <sup>△</sup>	Number <sup>△</sup>	Start <sup>△</sup>
1	absent	71	3	5
2	absent	158	3	14
3	present	128	4	5
4	absent	2	5	1
5	absent	1	4	15
6	absent	1	2	16
7	absent	61	2	17
8	absent	37	3	16
9	absent	113	2	16
10	present	59	6	12
11	present	82	5	14
12	absent	148	3	16

Instalamos e importamos las librerías:  
`install.packages("rpart")`  
`library(rpart)`



Importamos el dataset:  
`datosKypo <- kyphosis`  
`head(datosKypo)`

# ÁRBOLES EN R: CLASIFICACIÓN

## CREAR O ENTRENAR EL ÁRBOL



```
rpart(formula = y ~ x1 + x2 + ... + xn, data = arbol, method = "class")
```



La variable de interés o dependiente (y) en función de (~) todas las variables independientes o predictivas (xi).



Los datos sobre los que vamos a calcular el árbol.



El tipo de árbol que queremos lograr:

- **"class"** para clasificación.
- **"anova"** para regresión.

Matemáticamente, se escribe:

$$y = f(x_1, x_2, \dots, x_n)$$



# ÁRBOLES EN R: CLASIFICACIÓN

## CREAR O ENTRENAR EL ÁRBOL



Creamos el árbol de clasificación con rpart:

```
fitKypo <- rpart(Kyphosis ~ Age + Number + Start, data = kyphosis, method = "class")
```

Visualizarlo “a mano” sólo es posible en árboles chicos

fitKypo

```
> fitKypo
```

```
n= 81
```

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

```
1) root 81 17 absent (0.79012346 0.20987654)
  2) Start>=8.5 62 6 absent (0.90322581 0.09677419)
    4) Start>=14.5 29 0 absent (1.00000000 0.00000000) *
    5) Start< 14.5 33 6 absent (0.81818182 0.18181818)
      10) Age< 55 12 0 absent (1.00000000 0.00000000) *
      11) Age>=55 21 6 absent (0.71428571 0.28571429)
        22) Age>=111 14 2 absent (0.85714286 0.14285714) *
        23) Age< 111 7 3 present (0.42857143 0.57142857) *
  3) Start< 8.5 19 8 present (0.42105263 0.57894737) *
```

*¡Necesitamos mejorar la  
visualización para poder  
analizar los datos!*

# ÁRBOLES EN R: CLASIFICACIÓN

## VISUALIZAR LOS DATOS



`printcp(fit)`

Muestra la tabla para el parámetro de complejidad.

`plotcp(fit)`

Grafica los resultados de validación cruzada.

`rsq.rpart(fit)`

Grafica una aproximación al R-cuadrado y al error relativo, para diferentes particiones (o splits). Las etiquetas sólo son apropiadas para el método "anova".

`print(fit)`

Imprime los resultados.

`summary(fit)`

Detalla los resultados incluyendo las ramas subrogadas.

`plot(fit)`

Grafica el árbol de decisión.

`text(fit)`

Etiqueta el árbol de decisión.

`post(fit, file=)`

Crea un postscript (archivo) para el árbol de decisión.

El **parámetro de complejidad** se usa para controlar el tamaño del árbol de decisión y **para seleccionar el tamaño óptimo**. Si el costo de agregar otra variable al nodo, está por encima del parámetro, entonces no se continúa construyendo el árbol.

```
> printcp(fitkypo)
```

```
Classification tree:
```

```
rpart(formula = Kyphosis ~ Age + Number + Start, data  
= kyphosis,  
method = "class")
```

```
Variables actually used in tree construction:
```

```
[1] Age    Start
```

```
Root node error: 17/81 = 0.20988
```

```
n= 81
```

	CP	nsplit	rel error	xerror	xstd
1	0.176471	0	1.00000	1.0000	0.21559
2	0.019608	1	0.82353	1.1176	0.22433
3	0.010000	4	0.76471	1.1176	0.22433



# ÁRBOLES EN R: CLASIFICACIÓN

## VISUALIZAR LOS DATOS



`plot(fit)`

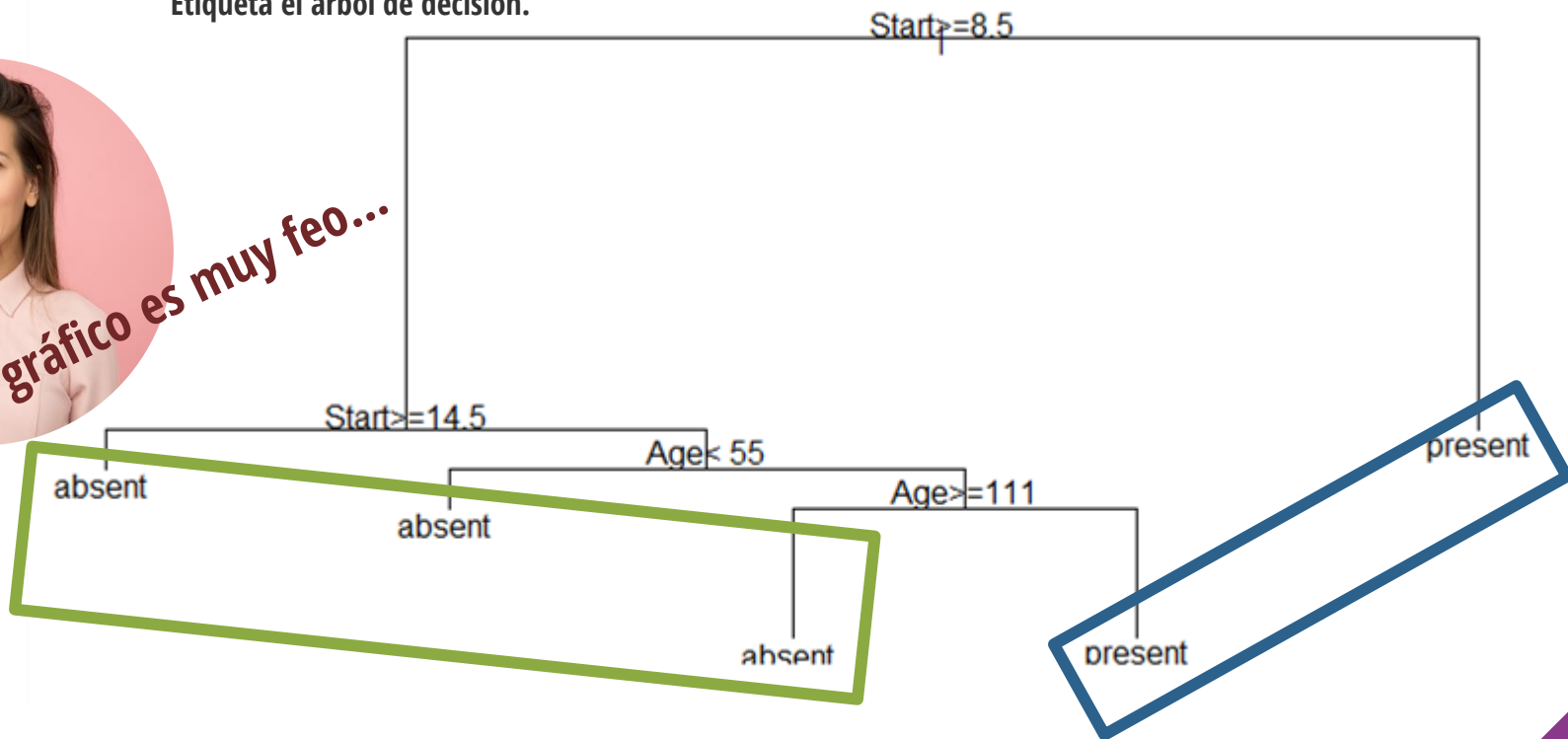
Grafica el árbol de decisión.

`text(fit)`

Etiqueta el árbol de decisión.



*El gráfico es muy feo...*



# ÁRBOLES EN R: CLASIFICACIÓN

## VISUALIZAR LOS DATOS



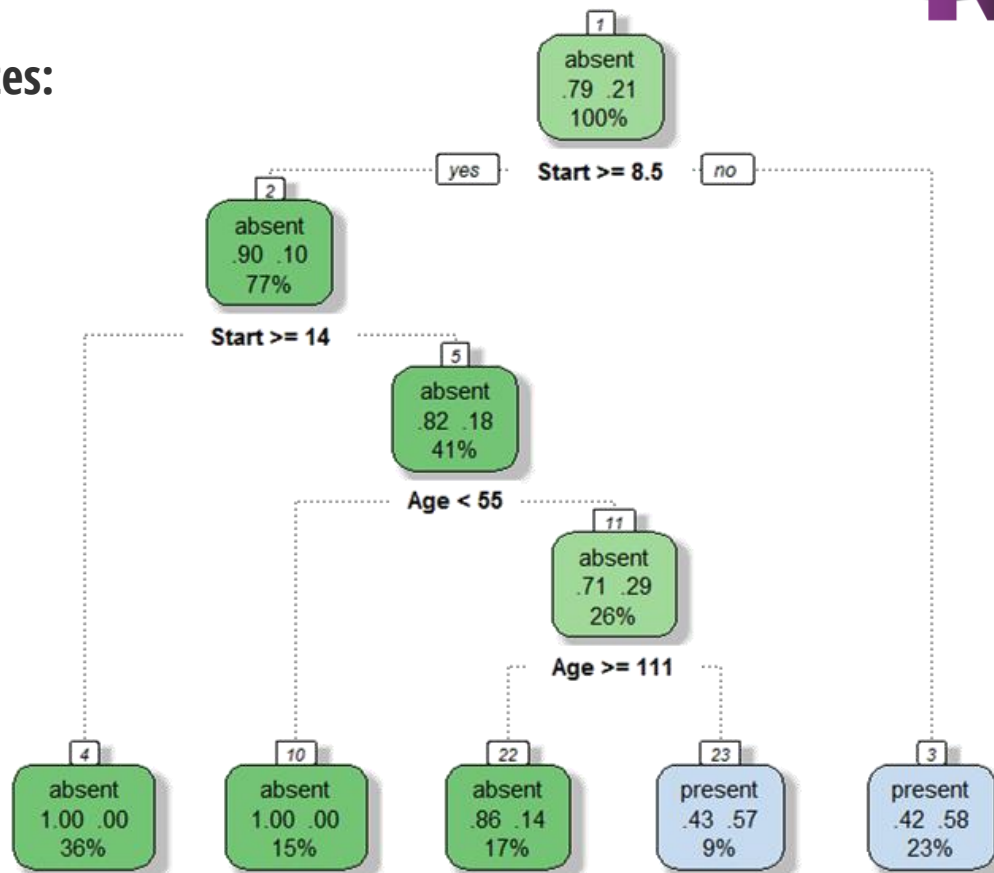
Mejoremos la visualización con paquetes:

```
library(rattle)
library(rpart.plot)
library(RColorBrewer)
```

```
fancyRpartPlot(fitKypo)
```



*¡Mejoramos el gráfico!  
Ahora falta podarlo...*



# ÁRBOLES EN R: CLASIFICACIÓN

## PODAMOS EL ÁRBOL



```
prune(tree = miArbolFit, cp = parametro)
```

Este es el árbol ya generado mediante rpart. No es el dataframe con los datos “en crudo”



Este es el valor de “recorte” del árbol, usando el parámetro de complejidad. Es decir, nos quedamos con aquellos que cumplen la condición.



```
podaKypo <- prune(tree = fitKypo,  
  cp = fitKypo$cptable[which.min(fitKypo$cptable[, "xerror"]), "CP"])
```

1. Buscamos la columna “xerror” en la tabla del parámetro de complejidad.
2. Seleccionamos los valores mínimos de dicha tabla.
3. Con esos datos, buscamos los valores en la tabla del parámetro de complejidad, pero en la columna del parámetro (“CP”).

¿Qué obtenemos?



# ÁRBOLES EN R: CLASIFICACIÓN

## PODAMOS EL ÁRBOL



```
> printcp(podakypo)
```

Classification tree:

```
rpart(formula = Kyphosis ~ Age + Number + Start, data = kyphosis  
      method = "class")
```

Variables actually used in tree construction:

```
character(0)
```

Root node error: 17/81 = 0.20988

n= 81

	CP	nsplit	rel error	xerror	xstd
1	0.17647	0	1	1	0.21559

```
> plot(podakypo)
```

```
Error in plot.rpart(podakypo) : fit is not a tree, just a root
```

*¡Podamos demasiado y nos quedamos con sólo una raíz!*





# ÁRBOLES EN R: REGRESIÓN

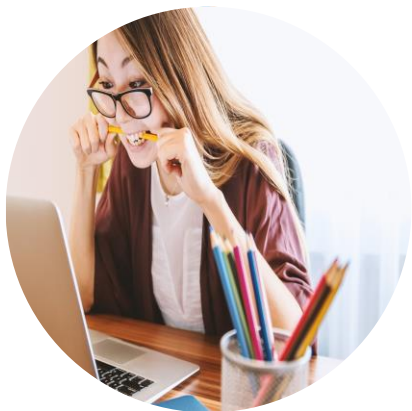


Objetivo: predecir el kilometraje según el tipo, precio, país y confiabilidad.

Método: anova (para regresión = predicción)

Obtención: `datosAutos <- cu.summary`

Recuerden seguir los tres pasos: construcción, visualización, poda (si conviene)

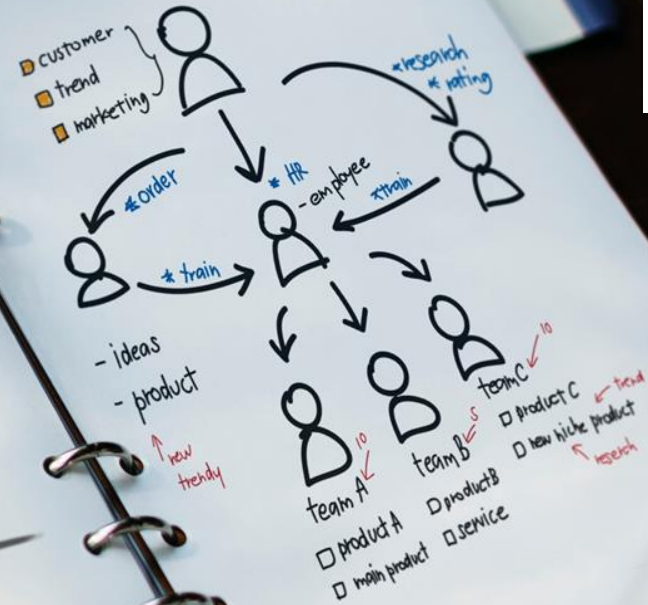


**¿Si tienen dudas y consultas?**

Slack: <https://rladies-santafe.slack.com>

Invitación: [santafe@rladies.org](mailto:santafe@rladies.org)





# REGLAS DE ASOCIACIÓN

# ¿QUÉ SON LAS REGLAS DE ASOCIACIÓN?



- » Permite descubrir qué hechos ocurren en común, dentro de un determinado conjunto de datos.
- » Uso común: descubrir qué productos se compran juntos, y hacer recomendaciones.
- » Puede hacerse con aprendizaje supervisado y no supervisado.



GrupoA (LHS) → GrupoB (RHS)

Una **regla** es una notación que representa qué cosas se agrupan: lo que está a la derecha, usualmente se agrupa con lo de la izquierda.

La regla se separa en mano izquierda (LHS) y mano derecha (RHS).

Los resultados se miden con tres valores: soporte, confianza, confianza esperada y lift



# MEDIDAS DE REGLAS DE ASOCIACIÓN



El soporte indica cuán frecuentemente el grupo aparece en el conjunto de datos.

$$\text{soporte} = \frac{\text{número de transacciones con A y B}}{\text{número total de transacciones}}$$

La confianza indica cuán a menudo la regla resulta verdadera para el dataset.

$$\text{confianza} = \frac{\text{número de transacciones con A y B}}{\text{número total de transacciones con A}}$$



La confianza esperada representa cuán frecuentemente el grupo B aparece en el conjunto de datos.

$$\text{confianzaEsperada} = \frac{\text{número de transacciones B}}{\text{número total de transacciones}}$$

El lift es la confianza real en contraste a la esperada.

$$\text{lift} = \frac{\text{confianza}}{\text{confianzaEsperada}}$$





# REGLAS DE ASOCIACIÓN EN R



Librería principal: **arules**

Script disponible en:

<https://tinyurl.com/y9ghley5>

Instalamos e importamos:

```
install.packages("arules")
```

```
library(arules)
```



Importamos el dataset:

```
ventas <- read.transactions("https://tinyurl.com/TransaccionesR",  
                             sep = ",")
```

```
> summary(ventas)
```

```
transactions as itemMatrix in sparse format with  
9835 rows (elements/itemsets/transactions) and  
169 columns (items) and a density of 0.02609146
```

```
most frequent items:
```

```
whole milk other vegetables  
2513 1903  
yogurt (other)  
1372 34055
```

```
rolls/buns  
1809
```

```
soda  
1715
```

```
element (itemset/transaction) length distribution:  
sizes
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2159	1643	1299	1005	855	645	545	438	350	246	182	117	78	77	55	46
17	18	19	20	21	22	23	24	26	27	28	29	32			
29	14	14	9	11	4	6	1	1	1	1	3	1			

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.000	2.000	3.000	4.409	6.000	32.000

```
includes extended item information - examples:
```

```
labels  
1 abrasive cleaner  
2 artif. sweetener  
3 baby cosmetics
```

# REGLAS DE ASOCIACIÓN EN R

## ANÁLISIS DE FRECUENCIA INICIAL



```
eclat(dataset, parameter = list(supp = X, maxlen = Y))
```



El conjunto de datos pasado a tipo de transacciones.



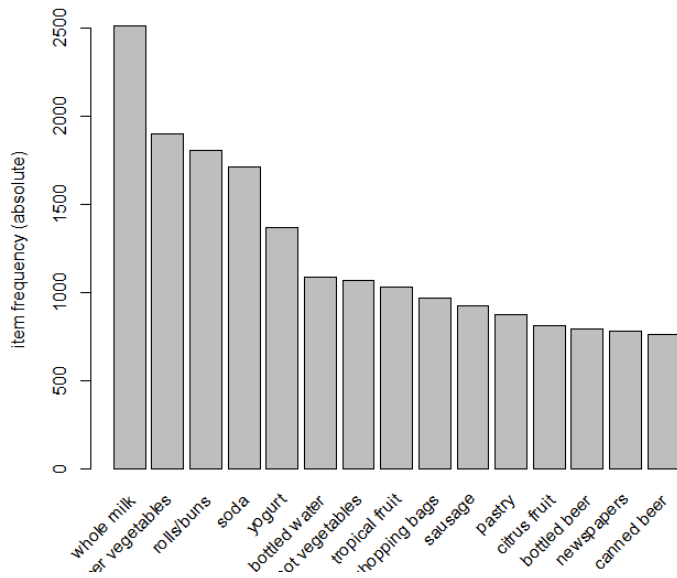
El soporte mínimo que queremos que tengan los ítems en las transacciones.



La cantidad máxima de ítems de transacciones que queremos obtener.

```
# Generamos los datos más frecuentes  
ventasFrecuentes <- eclat(ventas,  
  parameter = list(supp = 0.07, maxlen = 15))
```

```
# Visualizamos como barras  
itemFrequencyPlot(ventas, topN = 15,  
  type="absolute", main="Frecuencia de Venta")
```



# REGLAS DE ASOCIACIÓN EN R

## GENERACIÓN DE LAS REGLAS



```
apriori(data, parameter = NULL, appearance = NULL, control = NULL)
```

El conjunto de  
datos como  
transacciones.



Lista con el soporte y la  
confianza que queremos  
emplear para las reglas.



Restringimos la  
aparición de la  
presentación de la  
regla.



Controla la  
performance del  
algoritmo de reglas.



```
reglasVentas <- apriori(ventas, parameter = list(supp = 0.001, conf = 0.5))
```

Parameter specification:

confidence	minval	smax	arem	aval	originalsupport	maxtime	support	minlen	maxlen
0.5	0.1	1	none	FALSE	TRUE	5	0.001	1	10
target	ext								
rules	FALSE								

Algorithmic control:

filter	tree	heap	memopt	load	sort	verbose
0.1	TRUE	TRUE	FALSE	TRUE	2	TRUE

Absolute minimum support count: 9



# REGLAS DE ASOCIACIÓN EN R

## GENERACIÓN DE LAS REGLAS



# Quitamos reglas redundantes

```
reglasSubset <- which(colSums(is.subset(reglasVentas, reglasVentas)) > 1)
reglasVentas <- reglasVentas[-reglasSubset]
```

# Ordenamos las reglas y nos quedamos con las mejores

```
reglasLift <- sort(reglasVentas, by="lift", decreasing=TRUE)
inspect(head(reglasLift))
```



lhs	rhs	support	confidence	lift	count
[1] {Instant food products,soda}	=> {hamburger meat}	0.001220132	0.6315789	18.995654	12
[2] {baking powder,flour}	=> {sugar}	0.001016777	0.5555556	16.408075	10
[3] {ham,processed cheese}	=> {white bread}	0.001931876	0.6333333	15.045491	19
[4] {domestic eggs,processed cheese}	=> {white bread}	0.001118454	0.5238095	12.443639	11
[5] {liquor,red/blush wine}	=> {bottled beer}	0.001931876	0.9047619	11.235269	19
[6] {frozen vegetables,specialty chocolate}	=> {fruit/vegetable juice}	0.001016777	0.6250000	8.645394	10

# REGLAS DE ASOCIACIÓN EN R

## ANÁLISIS Y BÚSQUEDA DE REGLAS



¿Cómo controlar el largo de las reglas generadas?

Agregar el `maxlen` en los parámetros de configuración de `apriori`

```
reglasVCortas <- apriori(ventas, parameter = list (supp = 0.001, conf = 0.5, maxlen = 3))
```



¿Cómo obtener las reglas para un producto en particular?

```
reglasVentaLeche <- apriori(data = ventas,  
                             parameter = list (supp = 0.001,conf = 0.08),  
                             appearance = list(default = "lhs", rhs = "whole milk"),  
                             control = list (verbose = F))  
reglasVentaLeche.Lift <- sort(reglasVentaLeche, by = "confidence", decreasing = TRUE)  
inspect(head(reglasVentaLeche.Lift))
```

lhs	rhs	support	confidence	lift	count
[1] {rice,sugar}	=> {whole milk}	0.001220132	1	3.913649	12
[2] {canned fish,hygiene articles}	=> {whole milk}	0.001118454	1	3.913649	11
[3] {butter,rice,root vegetables}	=> {whole milk}	0.001016777	1	3.913649	10
[4] {flour,root vegetables,whipped/sour cream}	=> {whole milk}	0.001728521	1	3.913649	17
[5] {butter,domestic eggs,soft cheese}	=> {whole milk}	0.001016777	1	3.913649	10
[6] {butter,hygiene articles,pip fruit}	=> {whole milk}	0.001016777	1	3.913649	10



# REGLAS DE ASOCIACIÓN EN R

## ANÁLISIS Y BÚSQUEDA DE REGLAS



Quienes vieron este producto también compraron



\$ 849

Libro - It (eso) - Stephen King



\$ 4.200

Pack Saga Harry Potter  
Completa: Libros 1 A 8 - J K



\$ 649

El Bazar De Los Malos Sueños -  
Stephen King



\$ 649

Hasta 12 cuotas sin interés  
Bellas Durmientes - Stephen  
King - Tapa Dura - Plaza &



\$ 999

Hasta 12 cuotas sin interés  
Lote X 10 Libros Armá Tu  
Combo! Quiroga Lovecraft

# En este caso, fijamos el izquierdo (queremos que hayan comprado leche)

# Pero dejamos libre el derecho, para ver qué fue lo que compraron

```
reglasTambienCompraron <- apriori(data = ventas,  
                                   parameter = list(supp = 0.001, conf = 0.15, minlen = 2),  
                                   appearance = list(default="rhs",lhs="whole milk"),  
                                   control = list(verbose = FALSE))  
reglasTambienCompraron.Lift <- sort(reglasTambienCompraron, by = "confidence",  
                                   decreasing = TRUE)  
inspect(head(reglasTambienCompraron.Lift))
```

# REGLAS DE ASOCIACIÓN EN R

## ANÁLISIS Y BÚSQUEDA DE REGLAS



	lhs	rhs	support	confidence	lift	count
[1]	{whole milk}	=> {other vegetables}	0.07483477	0.2928770	1.5136341	736
[2]	{whole milk}	=> {rolls/buns}	0.05663447	0.2216474	1.2050318	557
[3]	{whole milk}	=> {yogurt}	0.05602440	0.2192598	1.5717351	551
[4]	{whole milk}	=> {root vegetables}	0.04890696	0.1914047	1.7560310	481
[5]	{whole milk}	=> {tropical fruit}	0.04229792	0.1655392	1.5775950	416
[6]	{whole milk}	=> {soda}	0.04006101	0.1567847	0.8991124	394

# En este caso, fijamos el izquierdo (queremos que hayan comprado leche)

# Pero dejamos libre el derecho, para ver qué fue lo que compraron

```
reglasTambienCompraron <- apriori(data = ventas,  
                                   parameter = list(supp = 0.001, conf = 0.15, minlen = 2),  
                                   appearance = list(default="rhs",lhs="whole milk"),  
                                   control = list(verbose = FALSE))  
reglasTambienCompraron.Lift <- sort(reglasTambienCompraron, by = "confidence",  
                                    decreasing = TRUE)  
inspect(head(reglasTambienCompraron.Lift))
```



**iCOFFEE BREAK!**





# ALGORITMOS GENÉTICOS

# ¿QUÉ SON LOS ALGORITMOS GENÉTICOS?



Es una meta-heurística inspirada en el **proceso de selección natural**.

Pertenece a los algoritmos evolucionarios.

Se usan para generar soluciones a problemas de optimización y de búsqueda.

Es aprendizaje no supervisado.

Son variables que me  
dan el resultado de mi  
problema.

$[x_1, x_2, x_3, \dots, x_n]$

Cada solución es un  
individuo, representado  
mediante un cromosoma.

## FUNCIÓN DE FITNESS

Es una función objetivo, que permite resumir  
en un sólo valor (o variable) cuán buena o  
adecuada es la solución que se evalúa.

Si estoy optimizando, puedo buscar valores  
máximos o valores mínimos.



# ¿CÓMO BUSCAMOS LA MEJOR SOLUCIÓN?



- » Todos los individuos están ubicados en la **población inicial**.
- » El problema **se resuelve por ciclos** (como si fueran generaciones): cada ciclo tiene que darnos una mejor población (con mejor *fitness*).
- » Las poblaciones se obtienen a través de una **selección**.

La selección se realiza mediante diferentes **operadores genéticos**.

## CROSSOVER

Se usa una combinación de los cromosomas de los dos padres, para crear un nuevo cromosoma (individuo).



## MUTACIÓN

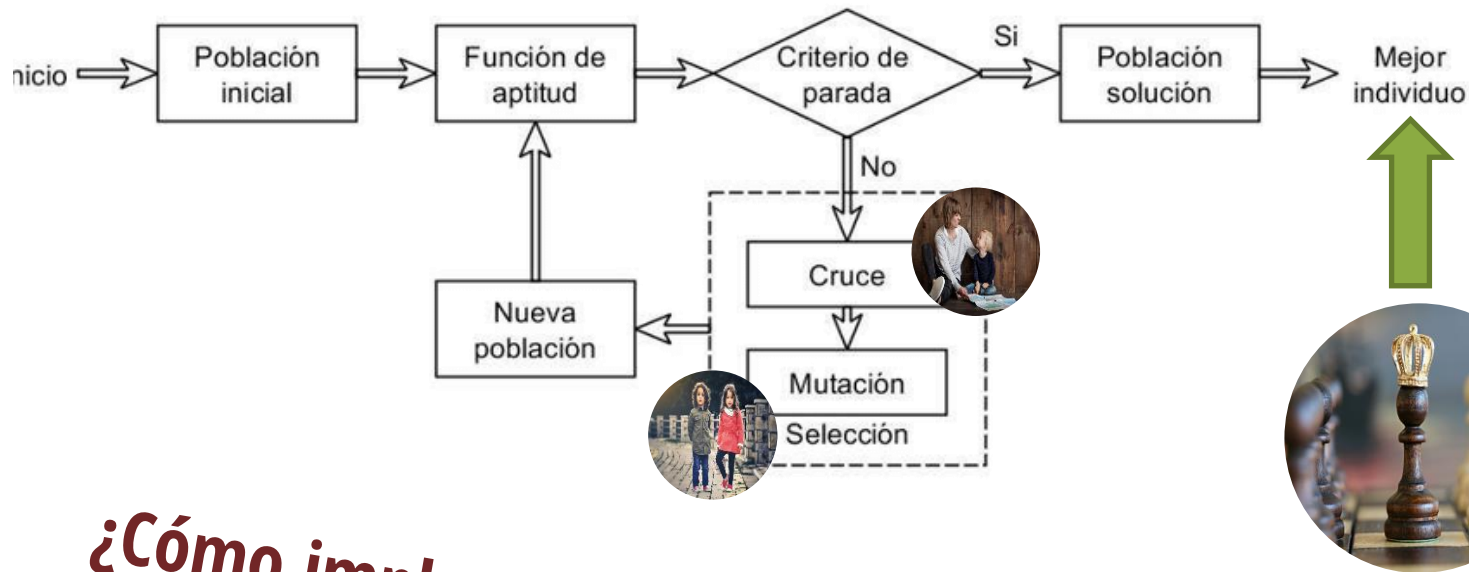
Mantiene diversidad genética. Dado un cromosoma, cambia un solo valor interno, para crear uno nuevo.

1 0 1 0 0 1 0  
↓  
1 0 1 0 1 1 0





# FUNCIONAMIENTO DE UN ALGORITMO



*¿Cómo implementamos  
esto en R?*

# ALGORITMOS GENÉTICOS EN R

## LIBRERÍAS Y DATOS NECESARIOS



El **problema del viajero** representa a un viajante : el tiene que salir del depósito y recorrer todas las ciudades, para luego volver al depósito de donde partió. ¿Cuál es la ruta más corta?

Script: <https://tinyurl.com/ya66y2ec>



```
# Importamos las librerías  
library(GA)
```

```
# Conjunto de datos de distancias en ciudades europeas  
distEuropa <- eurodist  
mapa <- as.matrix(distEuropa)
```

	Athens	Barcelona	Brussels	Calais	Cherbourg	Cologne	Copenhagen	Geneva	Gibraltar	Hamburg
Athens	0	3313	2963	3175	3339	2762	3276	2610	4485	2977
Barcelona	3313	0	1318	1326	1294	1498	2218	803	1172	2018
Brussels	2963	1318	0	204	583	206	966	677	2256	597
Calais	3175	1326	204	0	460	409	1136	747	2224	714
Cherbourg	3339	1294	583	460	0	785	1545	853	2047	1115
Cologne	2762	1498	206	409	785	0	760	1662	2436	460
Copenhagen	3276	2218	966	1136	1545	760	0	1418	3196	460

# ALGORITMOS GENÉTICOS EN R

## FUNCIÓN DE FITNESS



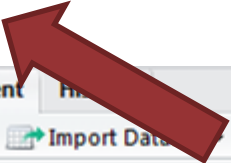
**TOUR** → secuencia de ciudades a visitar.  
**OBJETIVO** → la secuencia más corta (min)

# Función que calcula la distancia total de un tour, usando la matriz

```
largoTour <- function(tour, mDistancias = mapa) {  
  tour <- c(tour, tour[1])  
  ruta <- embed(tour, 2)[, 2:1]  
  sum(mDistancias[ruta])  
}
```

# El fitness es la inversa  
# de la distancia

```
fFitness <- function(tour, ...) {  
  1/largoTour(tour, ...)  
}
```



Environment	
Global Environment	
Data	
mapa	num [1:21, 1:21] 0 3313 2963 3175 3339 ...
Functions	
fFitness	function (tour, ...)
largoTour	function (tour, mDistancias)

# ALGORITMOS GENÉTICOS EN R

## RESOLUCIÓN DEL PROBLEMA



- "binary": variables de decisión binarias.

- "real-valued": números con coma.

- "permutation": listas de objetos.

```
ga(type = tipoSeleccion,  
    fitness = funcionDeFitness,  
    lower = valorMin,  
    upper = valorMax,  
    popSize = tamPoblacion,  
    maxiter = 5000,  
    run = 500,  
    pmutation = probabilidad)
```

Cantidad de  
iteraciones  
máximas.

Valores mínimos y máximos  
que se pueden tomar.

Cantidad de iteraciones sin mutar, y la  
probabilidad de mutar un individuo.

# ALGORITMOS GENÉTICOS EN R

## RESOLUCIÓN DEL PROBLEMA



# Resolvemos el problema con el algoritmo genético

```
algoGen <- ga(type = "permutation",
```

```
  fitness = fFitness,
```

```
  lower = 1,
```

```
  upper = attr(distEuropa, "Size"),
```

```
  popSize = 50,
```

```
  maxiter = 5000,
```

```
  run = 500,
```

```
  pmutation = 0.2)
```



GA		iter = 762		Mean = 3.779579e-05		Best = 6.961849e-05
GA		iter = 763		Mean = 3.945128e-05		Best = 6.961849e-05
GA		iter = 764		Mean = 3.853187e-05		Best = 6.961849e-05
GA		iter = 765		Mean = 3.767395e-05		Best = 6.961849e-05
GA		iter = 766		Mean = 3.847438e-05		Best = 6.961849e-05
GA		iter = 767		Mean = 3.863591e-05		Best = 6.961849e-05
GA		iter = 768		Mean = 3.657930e-05		Best = 6.961849e-05
GA		iter = 769		Mean = 3.687970e-05		Best = 6.961849e-05
GA		iter = 770		Mean = 3.619959e-05		Best = 6.961849e-05
GA		iter = 771		Mean = 3.598416e-05		Best = 6.961849e-05
GA		iter = 772		Mean = 3.688187e-05		Best = 6.961849e-05
GA		iter = 773		Mean = 3.728443e-05		Best = 6.961849e-05
GA		iter = 774		Mean = 3.756670e-05		Best = 6.961849e-05
GA		iter = 775		Mean = 3.896060e-05		Best = 6.961849e-05
GA		iter = 776		Mean = 3.999450e-05		Best = 6.961849e-05
GA		iter = 777		Mean = 4.004364e-05		Best = 6.961849e-05
GA		iter = 778		Mean = 3.807007e-05		Best = 6.961849e-05

# ALGORITMOS GENÉTICOS EN R

## RESOLUCIÓN DEL PROBLEMA



```
# Observemos un resumen de los resultados
summary(algoGen)
```

```
-- Genetic Algorithm -----
```

GA settings:

```
Type           = permutation
Population size  = 50
Number of generations = 5000
Elitism         = 2
Crossover probability = 0.8
Mutation probability = 0.2
```

GA results:

```
Iterations           = 1648
Fitness function value = 7.254788e-05
Solutions =
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	...	x20	x21
[1,]	13	18	4	3	6	11	7	20	10	17		2	15
[2,]	15	2	14	9	12	5	8	16	19	1		18	13



Los datos no son  
muy legibles, y  
queremos verlos  
gráficamente.



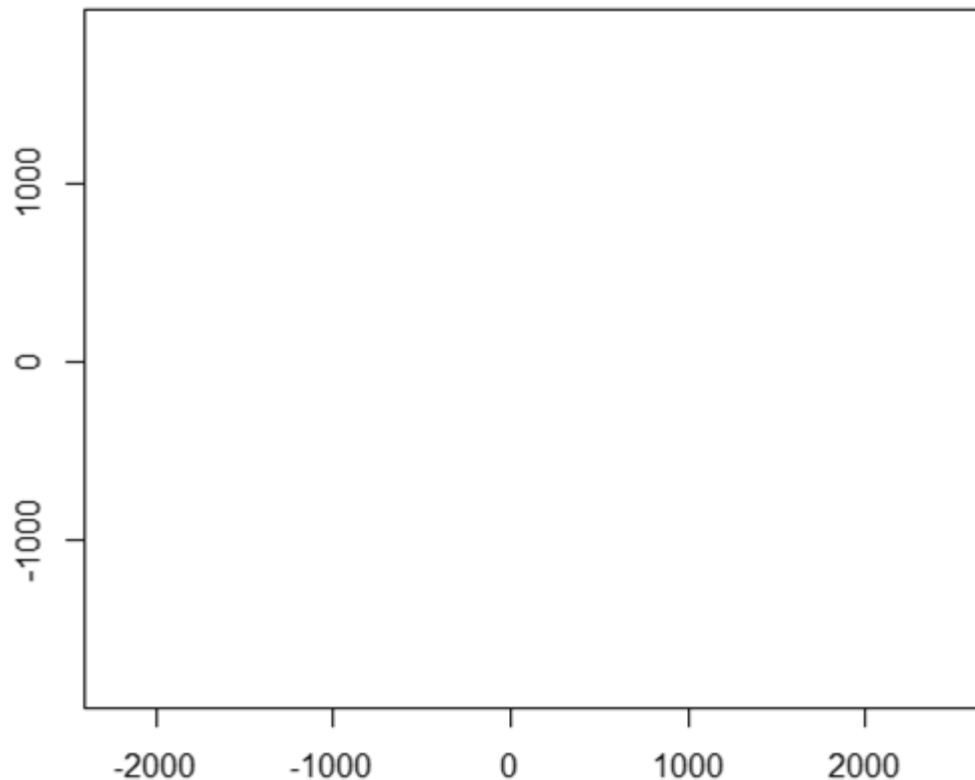
# ALGORITMOS GENÉTICOS EN R

## VISUALIZACIÓN DEL RECORRIDO



```
# Obtenemos la escala de distancias
mds <- cmdscale(distEuropa)
# Valores por columnas
x <- mds[, 1]
y <- -mds[, 2]

# Construimos un gráfico vacío
plot(x = x, y = y, type = "n", asp = 1)
```



# ALGORITMOS GENÉTICOS EN R

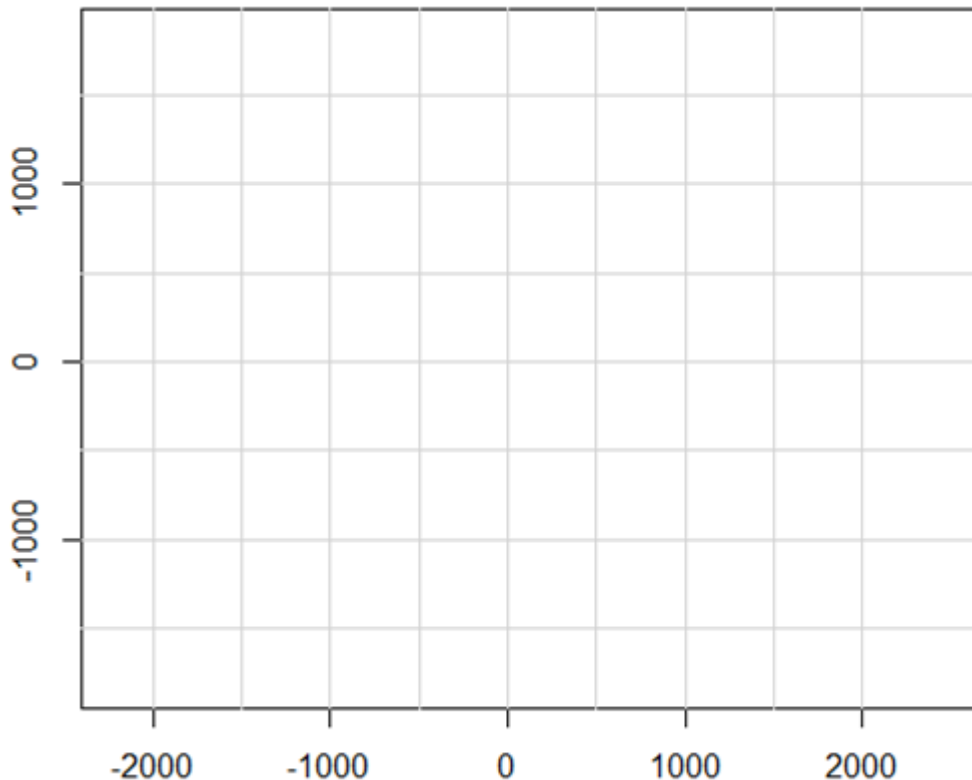
## VISUALIZACIÓN DEL RECORRIDO



```
# Obtenemos la escala de distancias
mds <- cmdscale(distEuropa)
# Valores por columnas
x <- mds[, 1]
y <- -mds[, 2]

# Construimos un gráfico vacío
plot(x = x, y = y, type = "n", asp = 1)

# Grilla color gris claro
abline(h = pretty(range(x), 10), v =
  pretty(range(y), 10),
  col = "light gray")
```



# ALGORITMOS GENÉTICOS EN R

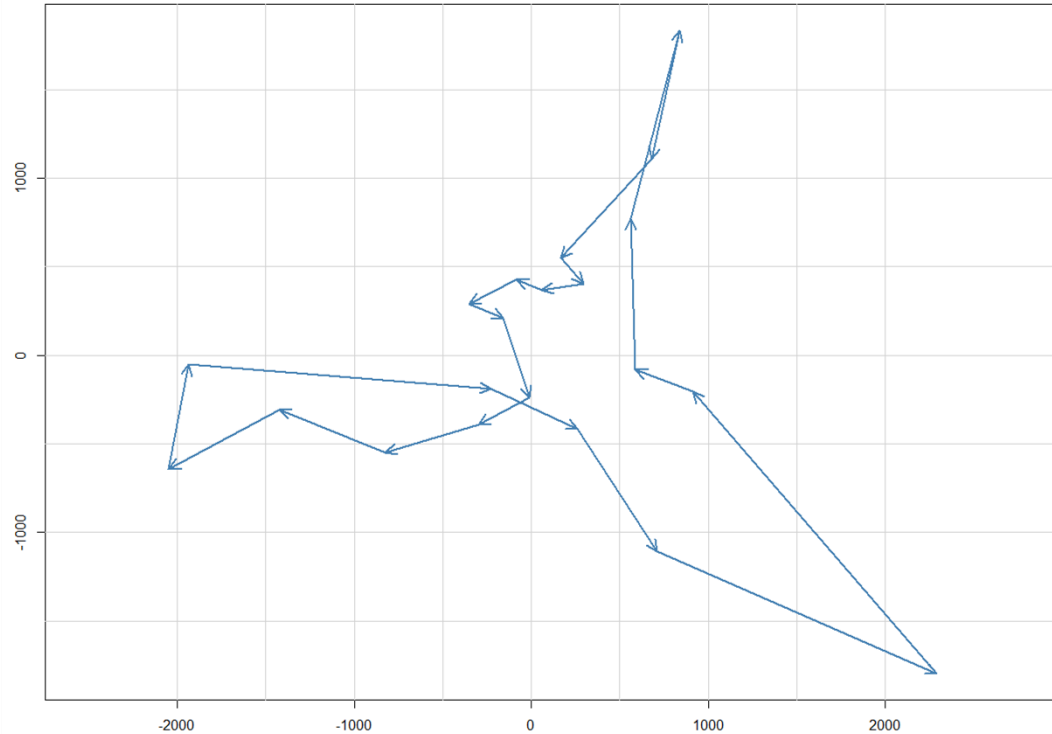
## VISUALIZACIÓN DEL RECORRIDO



```
# Extraemos la solución del algoritmo
tour <- algoGen@solution[1, ]
tour <- c(tour, tour[1])
```

```
# Obtenemos el largo del tour o ruta
n <- length(tour)
```

```
# Agregamos las flechas del recorrido
arrows(x[tour[-n]],
       y[tour[-n]],
       x[tour[-1]], y[tour[-1]],
       length = 0.15,
       angle = 25,
       col = "steelblue",
       lwd = 2)
```

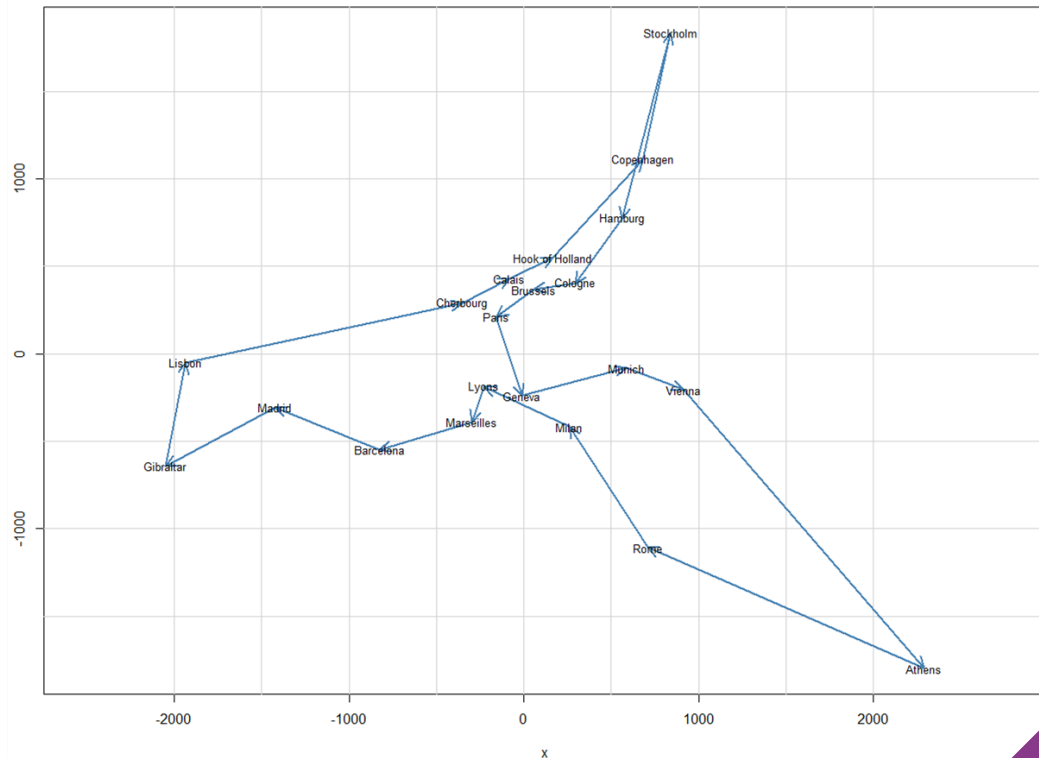


# ALGORITMOS GENÉTICOS EN R

## VISUALIZACIÓN DEL RECORRIDO



```
# Agregamos el nombre de las ciudades  
text(x, y, labels(eurodist), cex=0.8)
```



# ALGORITMOS GENÉTICOS EN R

## TRANSCRIPCIÓN DEL RECORRIDO



Esta variable tiene el orden de las ciudades a visitar.

```
tour      x1  x2  x3  x4  x5  x6  x7  x8  x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 x20 x21  x1
          5   4  11   7  20  10   6   3  18   8  17  21   1  19  16  13  15   2  14   9  12   5
```

Pero las ciudades están ordenadas alfabéticamente como nombres de filas...

```
rownames(mapa) [1] "Athens"      "Barcelona"    "Brussels"     "Calais"       "Cherbourg"
                [6] "Cologne"     "Copenhagen"   "Geneva"       "Gibraltar"    "Hamburg"
               [11] "Hook of Holland" "Lisbon"       "Lyons"        "Madrid"       "Marseilles"
               [16] "Milan"       "Munich"       "Paris"        "Rome"         "Stockholm"
               [21] "Vienna"
```

¡Usamos el primer vector para ordenar las ciudades!

```
solucion <- rownames(mapa)[tour]
```

```
[1] "Cherbourg"    "Calais"      "Hook of Holland" "Copenhagen"    "Stockholm"
[6] "Hamburg"      "Cologne"     "Brussels"       "Paris"        "Geneva"
[11] "Munich"       "Vienna"     "Athens"         "Rome"         "Milan"
[16] "Lyons"        "Marseilles" "Barcelona"      "Madrid"       "Gibraltar"
[21] "Lisbon"       "Cherbourg"
```

¡Al fin!



# “LA MOCHILA” EN R

## PRÁCTICA DE ALGORITMOS GENÉTICOS



Problema: Nos vamos de viaje y llevamos una sola mochila. Tenemos una lista de cosas que podemos llevar, *pero no entra todo*. Cada elemento mejora nuestra supervivencia. Tenemos que llevar sólo los mejores elementos, y que no superen el peso soportado por la mochila.

```
elementosMochila <- read.csv(file = "https://tinyurl.com/yby8eglc",  
                             header = TRUE, sep = ",")  
  
pesoMochila <- 20
```



### ¿Si tienen dudas y consultas?

Slack: <https://rladies-santafe.slack.com>

Invitación: [santafe@rladies.org](mailto:santafe@rladies.org)





# ¡GRACIAS POR VENIR!

