

# PROGRAMA DE ATRACCIÓN DE TALENTO TECNOLÓGICO PARA EL SECTOR TURÍSTICO ANDALUZ

Machine Learning con  
lenguaje R.

20  
20

# Machine Learning en lenguaje R.

## Introducción: Ideas básicas.

- Debido al aumento de la capacidad de dispositivos y el avance de las tecnologías de la información, se estima que cada día de producen, almacenar y envían más datos que nunca antes en la historia.
- Se calcula que el 90% de los datos disponibles actualmente en el planeta se han creado en los últimos 10 años, produciéndose actualmente en torno a 2,5 quintillones (2.500.000.000.000.000) de bytes por día, siguiendo una tendencia al alza. Estos datos alimentan los modelos de Machine Learning y son el motivo principal por el que esta ciencia se ha vuelto tan popular en los últimos años.

## Introducción: Ideas básicas.

“Machine Learning es la ciencia que permite que las computadoras aprendan y actúen como lo hacen los humanos, mejorando su aprendizaje a lo largo del tiempo de una forma autónoma, alimentándolas con datos e información en forma de observaciones e interacciones con el mundo real.”  
— Dan Fagella: Speaker and CEO Emerj Artificial Intelligence Research

## Introducción: Ideas básicas.

- El objetivo de algunos algoritmos de ML es el de entrenar los modelos. Dichos algoritmos proporcionan datos de entrenamiento que permiten a los modelos aprender de ellos.
- Los algoritmos de Machine Learning pueden tener parámetros “internos”. Por ejemplo, en los árboles de decisión hay parámetros como la profundidad máxima del árbol y número de nodos. A estos parámetros se les llama “hiper-parámetros”.
- Se le llama “generalización” a la capacidad del modelo de llevar a cabo predicciones utilizando nuevos datos.

# Tipos de ML

- Aplicaciones del ML.
  - <https://www.youtube.com/watch?v=oT3arRRB2Cw>
  -
- Los tipos de ML básicos son:
  - \* Aprendizaje supervisado
  - \* Aprendizaje no supervisado
  - \* Aprendizaje profundo

## Tipos de ML

### \* Aprendizaje supervisado

Modelos que se entrena con un conjunto de datos de ejemplo, llamados de entrenamiento, en los que los resultados de salida son conocidos. Los modelos aprenden de esos resultados conocidos y realizan ajustes en sus parámetros interiores para adaptarse a los datos de entrada. Una vez el modelo es entrenado adecuadamente, y los parámetros internos son coherentes con los datos de entrada y los resultados de la batería de datos de entrenamiento, el modelo podrá realizar predicciones adecuadas ante nuevos datos no procesados previamente.

*Hay dos aplicaciones principales de aprendizaje supervisado: **regresión** y **clasificación**:*

## Tipos de ML

### \* Aprendizaje no supervisado

En el aprendizaje no supervisado, se tratan datos sin etiquetar, cuya estructura es desconocida. El objetivo será la extracción de información significativa, sin la referencia de variables de salida conocidas, y mediante la exploración de la estructura de dichos datos sin etiquetar.

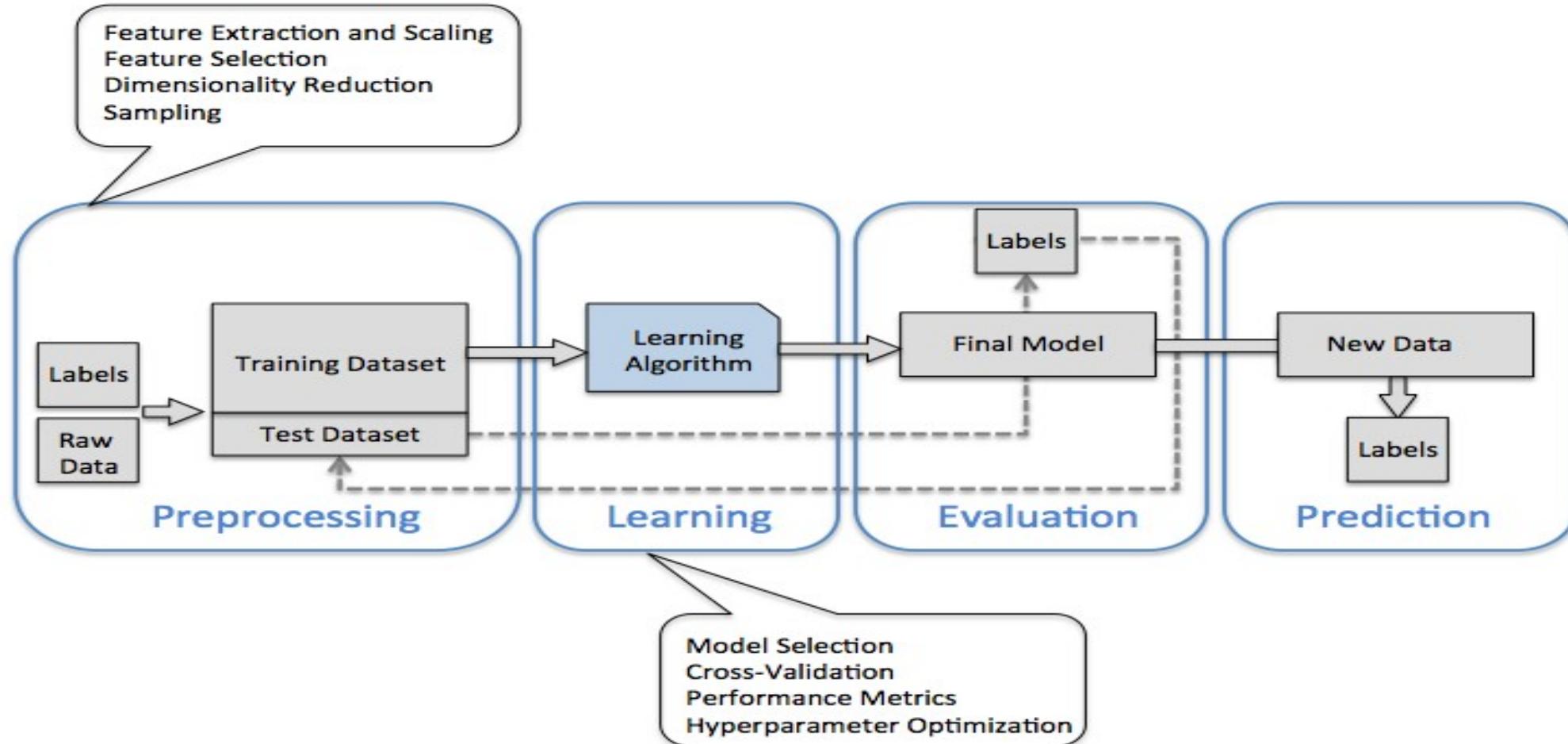
Hay dos categorías principales: **agrupamiento (clustering)** y **reducción dimensional**.

# Tipos de ML

## \* Aprendizaje profundo

Usa una estructura jerárquica de redes neuronales artificiales, que se construyen de una forma similar a la estructura neuronal del cerebro humano con los nodos de neuronas conectadas. Esta arquitectura permite abordar el análisis de datos de forma no lineal. La primera capa de la red neuronal toma datos en bruto como entrada, los procesa, extrae información y la transfiere a la siguiente capa como salida. Este proceso se repite en las siguientes capas, cada capa procesa la información proporcionada por la capa anterior, y así sucesivamente hasta que los datos llegan a la capa final, que es donde se obtiene la predicción.

# Metodología de ML



## Preprocesamiento de datos.

- Los datos se suelen encontrar en formatos no inadecuados para ser procesados por el modelo. El pre-procesamiento de datos es obligatorio.
- Algunos algoritmos requieren que las variables estén en la misma escala (por ejemplo, en el rango [0,1]) para optimizar su rendimiento, lo que se conoce como *escalado de datos*.
- Las características ser redundantes y poco significativas para el modelo. En este caso hay que usar técnicas de reducción dimensional para obtener solo las variables dependientes significativas.
- Por último se fragmenta de forma aleatoria el conjunto de datos original en subconjuntos de entrenamiento y pruebas.

# Preprocesamiento de datos.

## GESTIÓN DE DATOS FALTANTES

#Importar el dataset.

```
dataset = read.csv('/home/adastra/Escritorio/CursoR/Datasets/Data.csv')
```

#Gestión de datos faltantes.

```
dataset$Age = ifelse(is.na(dataset$Age),  
                     ave(dataset$Age,  
                          FUN = function(x) mean(x, na.rm = TRUE)), dataset$Age)
```

```
dataset$Salary = ifelse(is.na(dataset$Salary),  
                        ave(dataset$Salary,  
                            FUN = function(x) mean(x, na.rm = TRUE)), dataset$Salary)
```

```
View(dataset)
```

# Preprocesamiento de datos.

## CATEGORIZACIÓN DE DATOS.

- En ocasiones es necesario que las variables sean cuantificables con el objetivo de aplicar modelos matemáticos (como regresiones lineales o logísticas).
- Es necesario representar los datos no numéricos como las cadenas a una representación numérica que pueda ser aplicada en formulas. Se crea un factor ordinal con "levels" y "labels" para cumplir con dicho objetivo.

# Preprocesamiento de datos.

## CATEGORIZACIÓN DE DATOS.

```
# Plantilla para el PreProcesado de Datos - Datos Categóricos
# Importar el dataset
dataset = read.csv('Datasets/Data.csv', stringsAsFactors = F)

str(dataset)
# Codificar las variables categóricas
dataset$Country = factor(dataset$Country,
                         levels = c("France", "Spain", "Germany"),
                         labels = c(1, 2, 3))

dataset$Purchased = factor(dataset$Purchased,
                           levels = c("No", "Yes"),
                           labels = c(0,1))

str(dataset)
```

# Preprocesamiento de datos.

## DIVIDIR LA MUESTRA EN CONJUNTOS DE ENTRENAMIENTO Y TESTING.

- Los datos para la fase de entrenamiento (que pueden ser entre un 70 y 80 por ciento del dataset) se utilizará para enseñar al algoritmo cómo debe predecir las variables dependientes partiendo de las variables independientes. Esto es aprendizaje supervisado.
- Los datos para la fase de testing (que pueden estar entre un 20 y 30 por ciento) se utilizarán para evaluar la efectividad del modelo y comprobar que la etapa de entrenamiento se ha llevado a cabo correctamente.

# Preprocesamiento de datos.

## DIVIDIR LA MUESTRA EN CONJUNTOS DE ENTRENAMIENTO Y TESTING.

```
#La librería caTools permite utilizar funciones pseudoaleatorias
#que permitirán posteriormente, dividir en dataset con valores lo suficientemente
#aleatorios como para evitar los sesgos en el apredizaje y evitar que se obtengan siempre los mismos valores.
#install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$Purchased, SplitRatio = 0.8)
training = subset(dataset, split == TRUE)
testing = subset(dataset, split == FALSE)
```

# Preprocesamiento de datos.

## ESCALADO DE DATOS.

- En un modelo de datos puede ocurrir que la diferencia entre dos variables desde un punto de vista numérico no permita definir las diferencias propias de cada variable.
- Por ejemplo, en el DataSet anterior, las variables edad y el salario tienen rangos diferentes y por lo tanto es importante definir el peso semántico de dichas variables, independientemente de su valor numérico
- A esto se le conoce como “escalado de datos” y consiste en la estandarización de los mismos.
- El proceso de estandarizar consiste en la generación de una campana de Gauss para definir los valores normales y desviaciones y se ejecuta de forma automática con la función “**scale**” en R.
- Esto es especialmente importante en los algoritmos de regresión lineal.

# Preprocesamiento de datos.

## ESCALADO DE DATOS.

```
#ERROR: La función "scale" se encarga de generar una media de los valores almacenados en cada columna.  
# No obstante, dichos valores deben ser numéricos. En este caso no lo son, aunque al visualizar el Dataset,  
# la columna "country" aparezca como un número, se trata de un factor, el cual se encuentra representado por una cadena.
```

```
training = scale(training)
```

```
#ERROR: Lo mismo que lo anterior
```

```
testing = scale(testing)
```

```
#La solución consiste en escalar sólo aquellas columnas que tienen valores numéricos.
```

```
training[,2:3] = scale(training[,2:3])
```

```
testing[,2:3] = scale(testing[,2:3])
```

# Plantilla para Preprocesamiento de datos.

```
# Plantilla para el Pre Procesado de Datos
#Paso 1 Importar el dataset
dataset = read.csv('Datasets/Data.csv')
#Paso 2. Tratamiento de los valores NA
dataset$Age = ifelse(is.na(dataset$Age),
                     ave(dataset$Age, FUN = function(x) mean(x, na.rm = TRUE)), dataset$Age)
dataset$Salary = ifelse(is.na(dataset$Salary), ave(dataset$Salary, FUN = function(x) mean(x, na.rm = TRUE)), dataset$Salary)
#Paso 3. Codificar las variables categóricas
dataset$Country = factor(dataset$Country, levels = c("France", "Spain", "Germany"), labels = c(1, 2, 3))
dataset$Purchased = factor(dataset$Purchased, levels = c("No", "Yes"), labels = c(0,1))
# Paso 4. Dividir los datos en conjunto de entrenamiento y conjunto de test
# install.packages("caTools")
library(caTools)
set.seed(123)
split = sample.split(dataset$Purchased, SplitRatio = 0.8)
training_set = subset(dataset, split == TRUE)
testing_set = subset(dataset, split == FALSE)
#Paso 5. Escalado de valores
training_set[,2:3] = scale(training_set[,2:3])
testing_set[,2:3] = scale(testing_set[,2:3])
```

# Regresiones.

- Los algoritmos de regresión permiten definir patrones sobre los datos y de esta manera predecir comportamientos futuros.
- Se trata de técnicas estadísticas que se llevan aplicando desde hace muchos años pero que han cobrado mucha importancia en el mundo del machine learning y el aprendizaje supervisado.
- Los principales tipos de regresión son:
  - \* Lineal simple.
  - \* Lineal múltiple.
  - \* Polinomica.
  - \* Máquinas de soporte vectorial.
  - \* Arboles de decisión.
  - \* Bosques aleatorios.

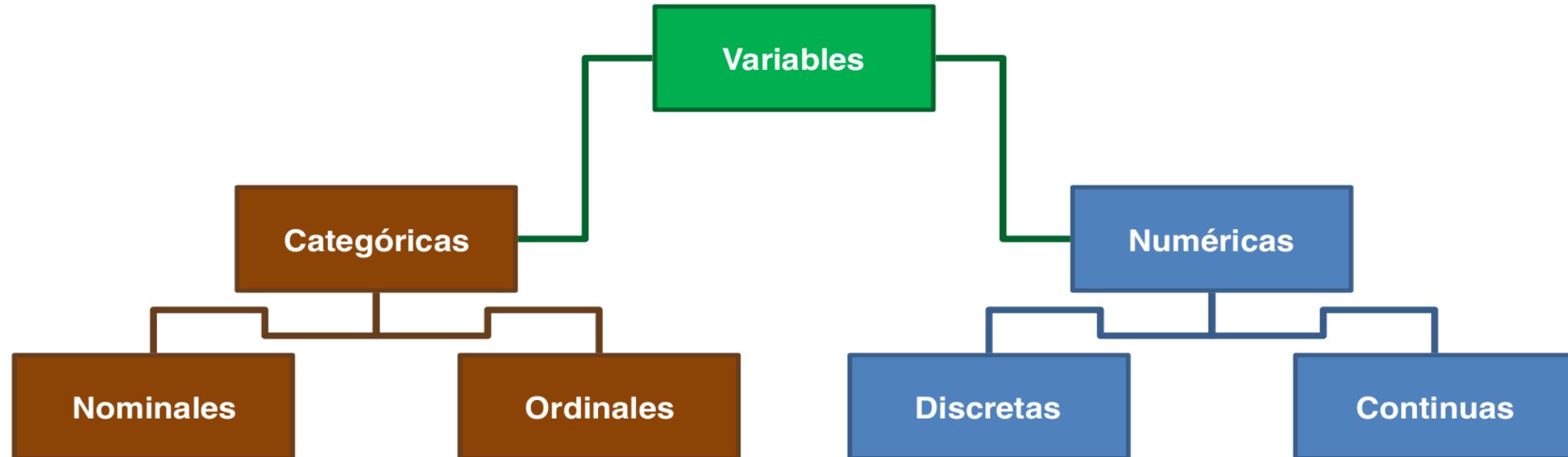
# Regresiones.

En estadística, se llama análisis de la regresión al proceso estadístico de estimar las relaciones que existen entre variables.

Se centra en estudiar las relaciones entre una variable dependiente de una o más variables independientes.

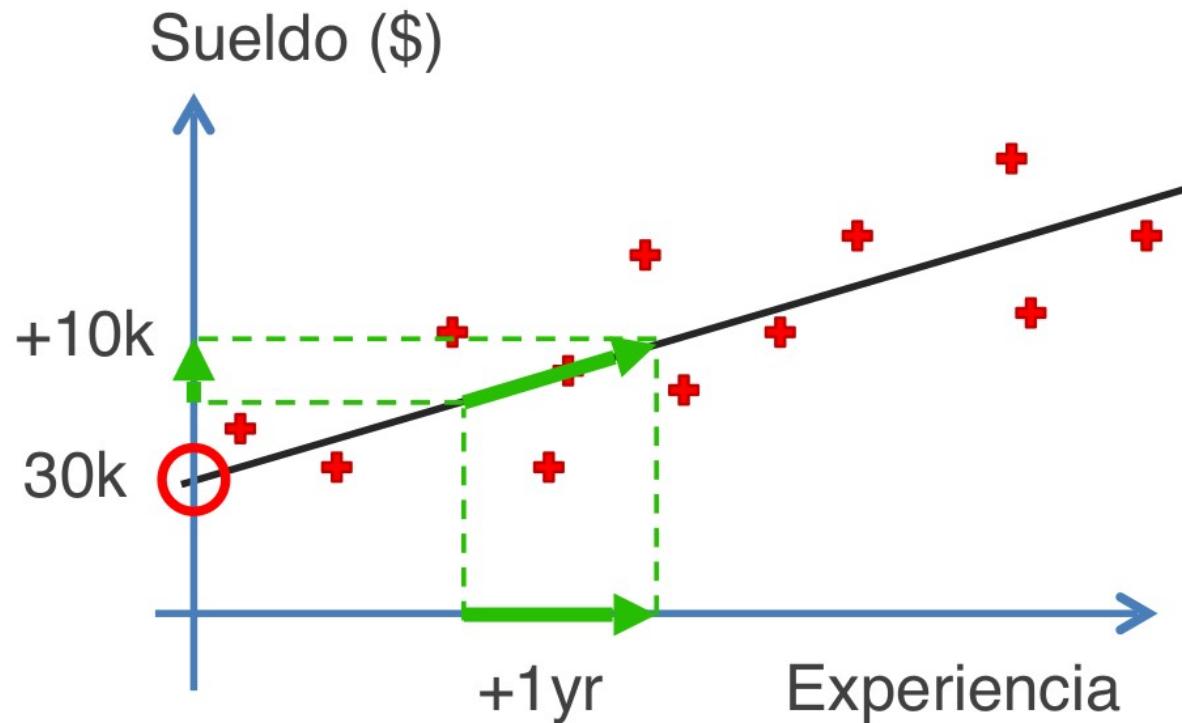
# Regresiones.

Tipos de variables.



# Regresión Lineal simple.

## Regresión Lineal Simple



$$y = b_0 + b_1 * x$$



$$\text{Sueldo} = \textcolor{red}{b_0} + \textcolor{green}{b_1} * \text{Experiencia}$$

# Regresiones Lineal simple.

```
# Regresión lineal simple.  
#Paso 1. Importar el dataset  
dataset = read.csv('Datasets/Salary_Data.csv')  
  
#Paso 2. Dividir los datos en conjunto de entrenamiento y conjunto de test  
#install.packages("caTools")  
library(caTools)  
set.seed(123)  
split = sample.split(dataset$Salary, SplitRatio = 2/3) #2 de cada 3 individuos de la variable dependiente  
entrenamiento = subset(dataset, split == TRUE) # Se genera el conjunto de entrenamiento.  
pruebas = subset(dataset, split == FALSE) # Se genera el conjunto de pruebas.
```

# **Regresiones Lineal simple.**

#Paso3. Modelo de regresión lineal simple con el conjunto de entrenamiento.

#En R la función "lm" permite crear un modelo lineal. ?lm

#El atributo "formula" permite indicar la relación entre la variable dependiente y la variable independiente.

#Su uso es el siguiente:

```
lm(formula = VariableDependiente ~ VariableIndependiente, data = DataSetR)
```

#En donde "DataSetR" será típicamente el conjunto de datos para entrenamiento.

```
regresion = lm(formula = Salary ~ YearsExperience, data = entrenamiento)
```

```
summary(regresion)
```

# **Regresiones Lineal simple.**

#Paso 4. Partiendo del modelo anterior se procede a generar una predicción.

```
prediccion = predict(regresion, newdata = pruebas)
```

```
# print(prediccion) # Se debe comparar el resultado de la predicción con el dataset de pruebas.
```

```
# En general, cuantos más años de experiencia, el salario debe tener una tendencia creciente lineal.
```

# Regresiones Lineal simple.

```
#Paso 5. install.packages("ggplot2") #Instalar la librería de gráficos en R.
```

```
library(ggplot2)
```

```
#Partiendo de la predicción anterior, se procede a generar la gráfica con los datos de entrenamiento.
```

```
ggplot() +
```

```
  geom_point(aes(x = entrenamiento$YearsExperience, y = entrenamiento$Salary),  
             colour = "red") + # Se definen los puntos "aes" representa cómo visualizar un punto.
```

```
  geom_line(aes(x = pruebas$YearsExperience,  
                y= predicción), colour="blue") +
```

```
  #La línea se debe trazar en función a la predicción realizada en el modelo lineal utilizando los datos de entrenamiento.
```

```
ggttitle("Relación del sueldo / años de experiencia (entrenamiento)") +
```

```
xlab("Años de experiencia: ") +
```

```
ylab("Sueldo: ")
```

# **Regresiones Lineal simple.**

#Paso3. Modelo de regresión lineal simple con el conjunto de entrenamiento.

#En R la función "lm" permite crear un modelo lineal. ?lm

#El atributo "formula" permite indicar la relación entre la variable dependiente y la variable independiente.

#Su uso es el siguiente:

```
lm(formula = VariableDependiente ~ VariableIndependiente, data = DataSetR)
```

#En donde "DataSetR" será típicamente el conjunto de datos para entrenamiento.

```
regresion = lm(formula = Salary ~ YearsExperience, data = entrenamiento)
```

```
summary(regresion)
```

# **Regresiones Lineal simple.**

#Paso3. Modelo de regresión lineal simple con el conjunto de entrenamiento.

#En R la función "lm" permite crear un modelo lineal. ?lm

#El atributo "formula" permite indicar la relación entre la variable dependiente y la variable independiente.

#Su uso es el siguiente:

```
lm(formula = VariableDependiente ~ VariableIndependiente, data = DataSetR)
```

#En donde "DataSetR" será típicamente el conjunto de datos para entrenamiento.

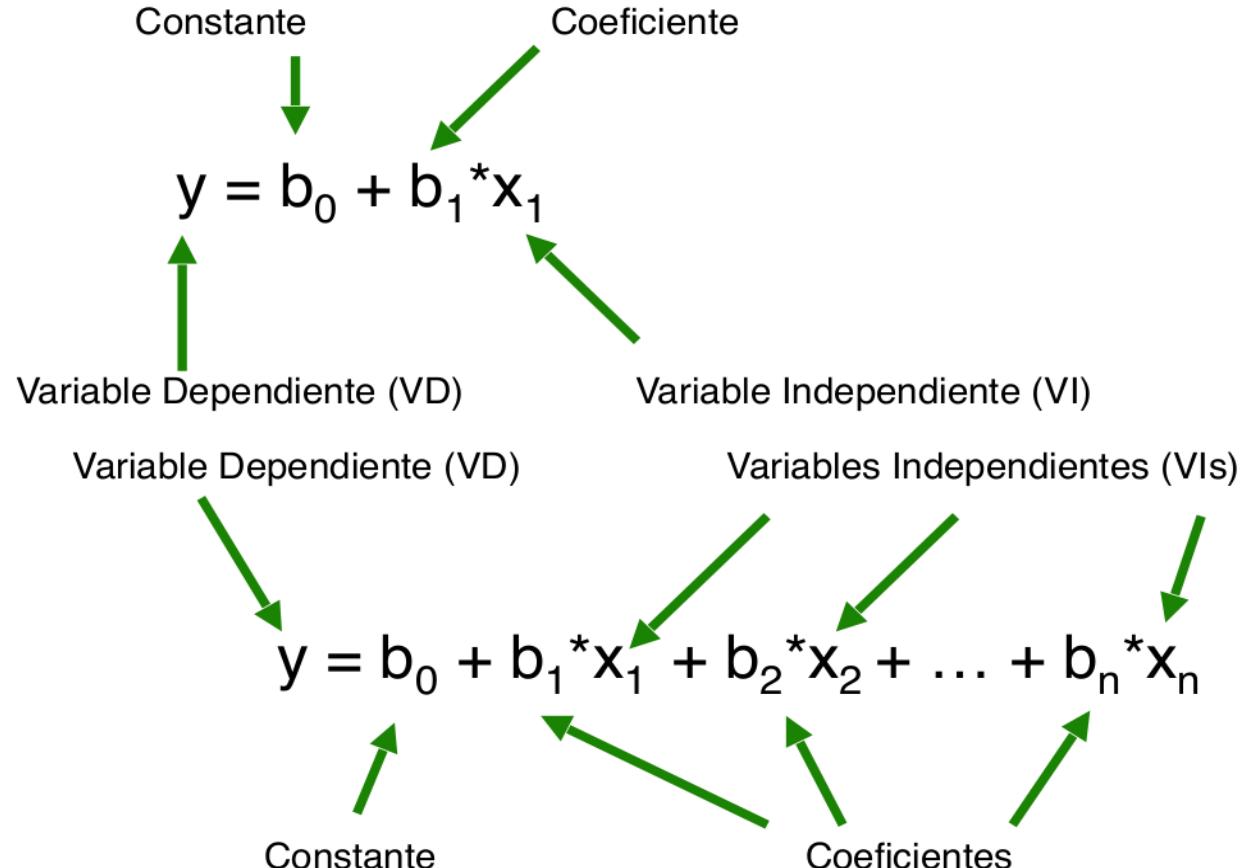
```
regresion = lm(formula = Salary ~ YearsExperience, data = entrenamiento)
```

```
summary(regresion)
```

# Regresión lineal múltiple.

Regresión  
Lineal Simple

Regresión  
Lineal Múltiple



# Regresión Lineal múltiple.

# Regresión lineal múltiple.

#Paso 1. Importar el dataset

```
dataset = read.csv('Datasets/50_Startups.csv')
```

#Categorizar las variables no numéricas del modelo de datos.

```
dataset$State = factor(dataset$State,  
                      levels = c("New York", "California", "Florida"),  
                      labels = c(1,2,3))
```

# Regresión Lineal múltiple.

#Paso 2. Dividir los datos en conjunto de entrenamiento y conjunto de test

```
#install.packages("caTools")  
library(caTools)  
set.seed(123)
```

```
split = sample.split(dataset$Profit, SplitRatio = 0.8) #80% de la muestra a predecir (variable dependiente)  
entrenamiento = subset(dataset, split == TRUE) # Se genera el conjunto de entrenamiento.  
pruebas = subset(dataset, split == FALSE) # Se genera el conjunto de pruebas.
```

# **Regresión Lineal múltiple.**

#Paso 3. Aplicar la regresión múltiple.

```
# lm(formula = Profit ~ R.D.Spend + Administration + Marketing.Spend + State, data = entrenamiento)
```

#la función sirve para Regresión lineal simple y múltiple.

#Con la siguiente sintaxis, es equivalente a decir que la variable dependiente es "Profit" y

```
# las variables independientes son todas las demás. Igual a la instrucción anterior.
```

```
regresion = lm(formula = Profit ~ ., data = entrenamiento) #la función sirve para Regresión lineal simple y múltiple.
```

```
summary(regresion)
```

# Regresión Lineal múltiple.

#Paso 4. Predecir los resultados con los datos de prueba.

```
prediccion = predict(regresion, data = pruebas)
```

#Se enseñará información sobre los valores que se han predicho para cada uno

#de los registros de pruebas. Pueden estar más o menos alejados del valor Profit,  
#pero se entiende que es un buen modelo ya que las diferencias no son tan amplias.

```
print(prediccion)
```

```
#prediccion
```

```
#2      4      5      8      11      16      20      21
```

```
#37766.77 44322.33 46195.35 55560.43 62115.99 71481.07 81782.66 89274.72
```

```
#24      26
```

```
#102385.84 109877.90
```

#Se han utilizado todas las variables, pero se puede probar la regresión utilizando  
#otras variables independientes que puedan ser mejores predictoras en función a su coeficiente.

# Regresión Lineal múltiple.

## Eliminación hacia atrás

**PASO 1:** Seleccionar el nivel de significación para permanecer en el modelo (p.e. SL = 0.05)



**PASO 2:** Se calcula el modelo con todas las posibles variables predictoras



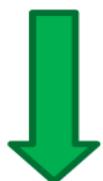
**PASO 3:** Considera la variable predictora con el p-valor más grande. Si  $P > SL$ , entonces vamos al PASO 4, si no vamos a FIN



**PASO 4:** Se elimina la variable predictora



**PASO 5:** Ajustar el modelo sin dicha variable\*



**FIN:** El modelo está listo

# Regresión Lineal múltiple: Eliminación hacia atrás.

#Paso 5. Regresión lineal múltiple: ELIMINACIÓN HACIA ATRAS.

#Algunas variables pueden tener mayor impacto que otras, en la eliminación hacia atrás

# se empieza como se ha visto anteriormente, seleccionando todas las variables independientes del modelo

# y a continuación, se eliminan las variables que son menos significativas desde el punto de vista estadístico.

#Construir un modelo óptimo con la eliminación hacia atrás.

#Paso 5.1: Seleccionar todas las variables dependientes y aplicar el modelo de regresión sobre TODO el dataset.

#no solo el conjunto de entrenamiento.

```
regresion_hacia_atras = lm(formula = Profit ~ R.D.Spend + Administration + Marketing.Spend + State, data = dataset)
```

#Si el P-Valor de una variable es superior al nivel de significancia (que tipicamente es 0,5),

#dicha variable no es relevante desde el punto de vista estadístico y se puede eliminar.

```
summary(regresion_hacia_atras)
```

# Regresión Lineal múltiple: Eliminación hacia atrás.

#En el resultado que se aprecia, se puede ver la tabla con los coeficientes de cada variable.

#Se puede apreciar que las variables dummy "State2" y "State3" tienen el coeficiente más alto, por lo tanto son las menos relevantes.

#Paso 5.2: Se elimina la variable independiente State ya que es la que menos relevancia estadística tiene.

```
regresion_hacia_atras = lm(formula = Profit ~ R.D.Spend + Administration + Marketing.Spend, data = dataset)
summary(regresion_hacia_atras)
```

#Paso 5.3: Se eliminan las variables independientes Administration y Marketing.Spend ya que son las que menos relevancia estadística tienen.

```
regresion_hacia_atras = lm(formula = Profit ~ R.D.Spend, data = dataset)
summary(regresion_hacia_atras)
```

#Fin del modelo. Ahora todas las variables tienen una significación inferior a 0.05

# Regresión Lineal múltiple: Eliminación hacia atrás.

```
#Paso 6. install.packages("ggplot2") #Instalar la librería de gráficos en R.  
library(ggplot2)  
prediccion = predict(regresion_hacia_atras, data = dataset)  
#Partiendo de la predicción anterior, se procede a generar la gráfica con los datos de entrenamiento.  
ggplot() +  
  geom_point(aes(x = entrenamiento$R.D.Spend, y = entrenamiento$Profit),  
             colour = "red") + # Se definen los puntos "aes" representa cómo visualizar un punto.  
  geom_line(aes(x = dataset$R.D.Spend,  
                y= prediccion), colour="blue") +  
  ggtitle("Relación del gato en I+D / Beneficios (entrenamiento)") +  
  xlab("Gasto I+D: ") +  
  ylab("Beneficio: ")
```

# Regresión Lineal polinómica:

**Regresión  
Lineal Simple**

$$y = b_0 + b_1 x_1$$

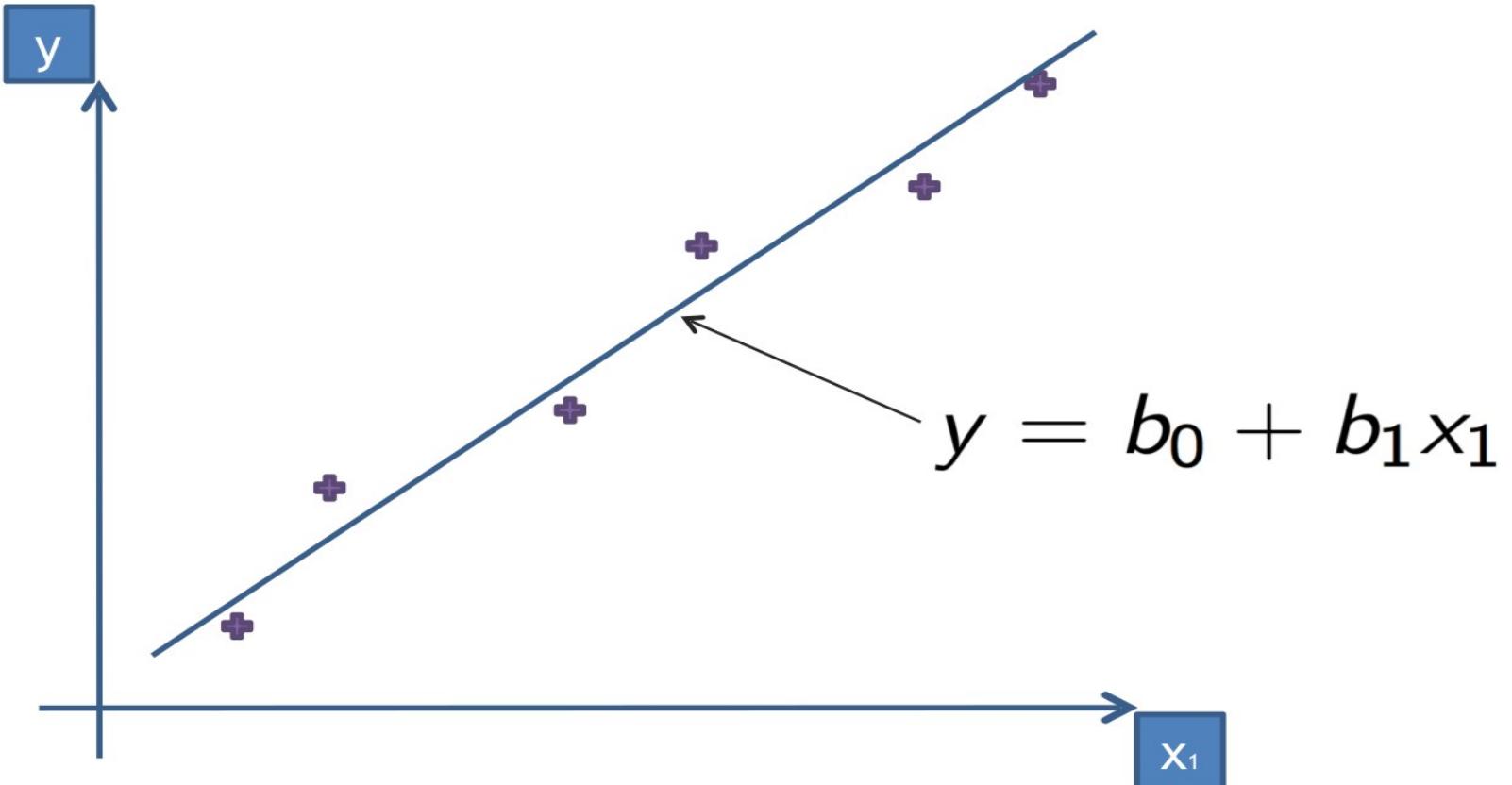
**Regresión  
Lineal Múltiple**

$$y = b_0 + b_1 x_1 + b_2 x_2 + \dots + b_n x_n$$

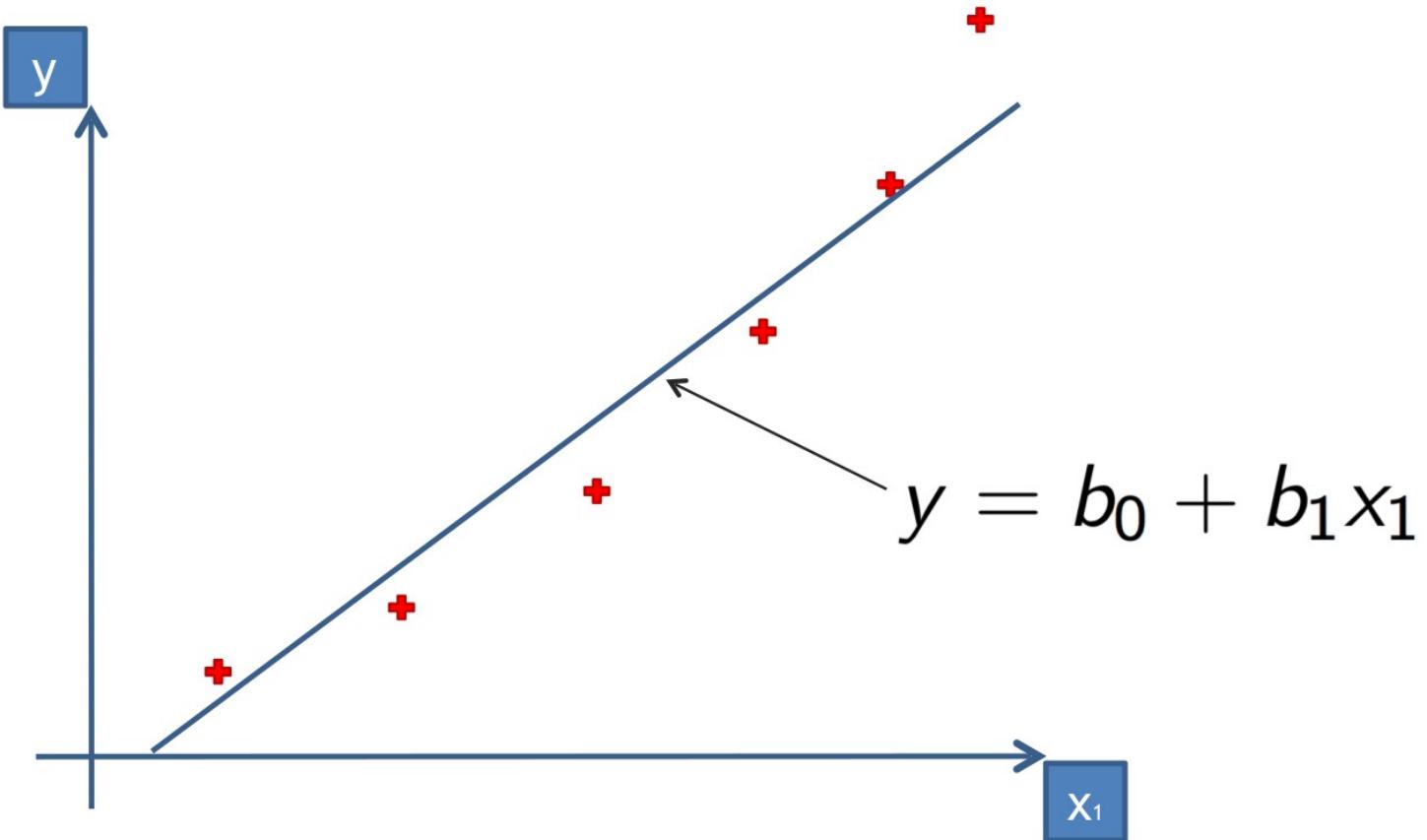
**Regresión  
Lineal  
Polinómica**

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \dots + b_n x_1^n$$

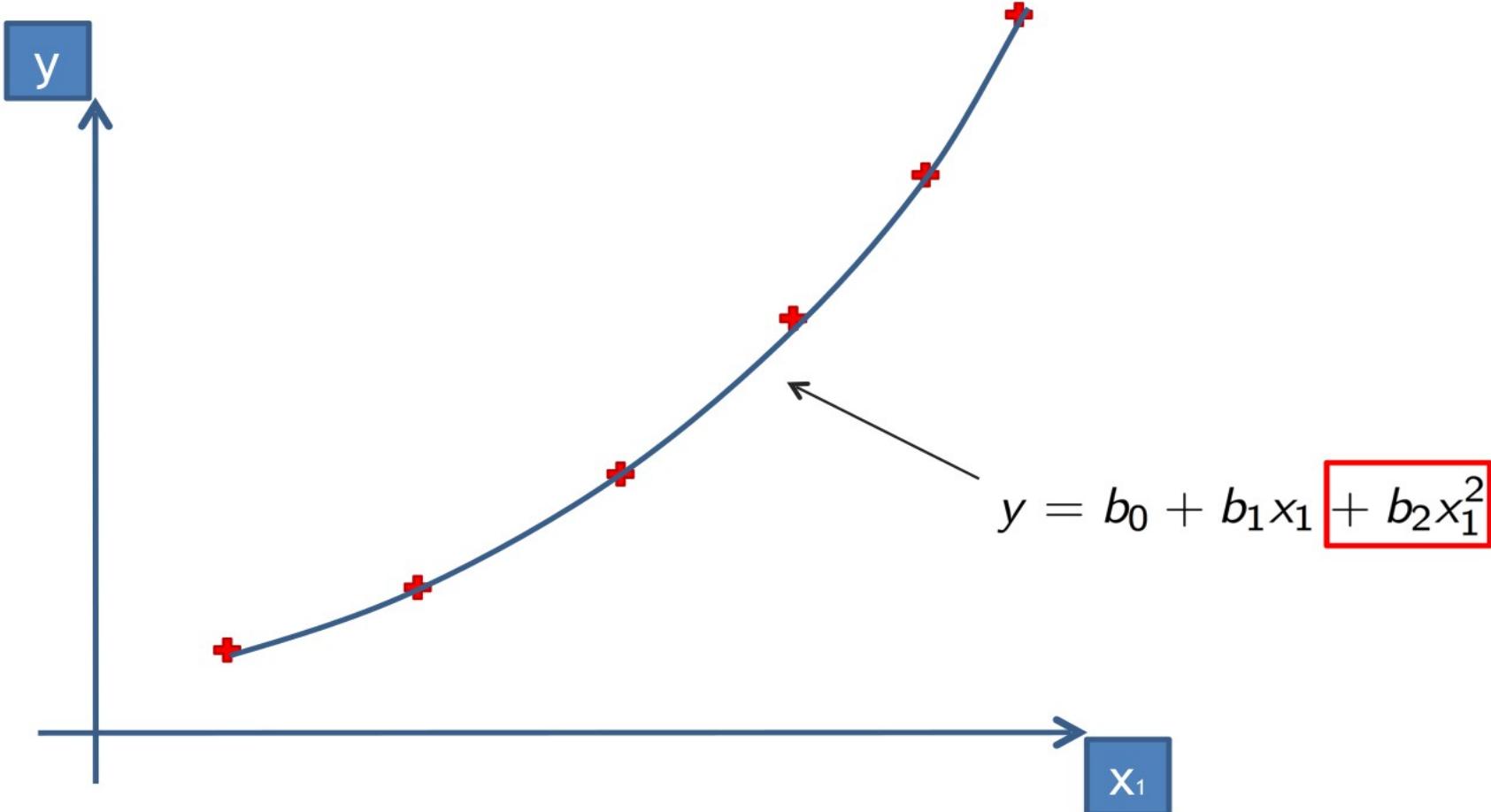
# Regresión Lineal polinómica:



# Regresión Lineal polinómica:



# Regresión Lineal polinómica:



# **Regresión Lineal polinómica:**

# Regresión Polinómica

#Paso 1. Importar el dataset

```
dataset = read.csv('Datasets/Salaries.csv')
```

# Solamente se eleccionan las columnas 2 y 3 del dataset, que son los que interesan realmente. (Level, Salary)

```
dataset = dataset[, 2:3]
```

# Regresión Lineal polinómica:

#Paso 2. Se ejecuta una regresión lineal con todas las variables y determinar que un modelo lineal no es el más óptimo.  
#El conjunto de datos en este caso es muy pequeño, por lo tanto no es práctico dividir el dataset en testing y training  
#ya que podría afectar a los resultados.

```
#  
lin_reg = lm(formula = Salary ~ .,  
             data = dataset)
```

#Para ver por qué el modelo lineal no es buena idea se procede a ver los coeficientes generados en la regresión.

```
summary(lin_reg)  
# Coefficients:  
#   Estimate Std. Error t value Pr(>|t|)  
# (Intercept) -195333     124790   -1.565  0.15615  
# Level        80879      20112    4.021  0.00383 **
```

# Se puede ver que el valor estimado es negativo.

# significa que el primer nivel empezaría no solo por debajo de lo que se indica en el dataset original sino que además, # el modelo indica que alguien que acaba de empezar a trabajar "debe pagar" por ese trabajo.  
# También se puede apreciar que a cada subida de nivel se incrementa 80879, lo cual tampoco encaja con el dataset original.  
# Aunque la variable Level tiene un p valor óptimo que está por debajo del 0.05 (es un buen predictor)  
# los datos arrojados indican que no es el modelo adecuado.

# Regresión Lineal polinómica:

#Paso 3. Se ajusta el modelo de regresión Polinómica con el conjunto de datos para ver las diferencias.

#Un modelo polinómico es similar al lineal pero es necesario tratar las variables independientes y aplicar el cuadrado, #el cubo o cualquier otra potencia a dichas variables. Esta es precisamente la diferencia entre un modelo lineal y polinómico.

```
dataset$Level2 = dataset$Level^2  
dataset$Level3 = dataset$Level^3  
dataset$Level4 = dataset$Level^4  
poly_reg = lm(formula = Salary ~ .,  
             data = dataset)
```

# Regresión Lineal polinómica:

#Paso 4. Se pueden observar los coeficientes y aunque las variables añadidas siguen teniendo algunas discrepancias  
# el modelo refleja mejor la realidad.

```
summary(poly_reg)  
# Coefficients:  
# Estimate Std. Error t value Pr(>|t|)  
# (Intercept) 184166.7 67768.0 2.718 0.04189 *  
# Level -211002.3 76382.2 -2.762 0.03972 *  
# Level2 94765.4 26454.2 3.582 0.01584 *  
# Level3 -15463.3 3535.0 -4.374 0.00719 **  
# Level4 890.2 159.8 5.570 0.00257 **
```

# Regresión Lineal polinómica:

#Paso 5. Se procede a crear el gráfico para enseñar el modelo lineal y ver que no es el más apropiado.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot() +
  geom_point(aes(x = dataset$Level , y = dataset$Salary),
             color = "red") +
  geom_line(aes(x = dataset$Level, y = predict(lin_reg, newdata = dataset)),
            color = "blue") +
  ggtitle("Predicción lineal del sueldo en función del nivel del empleado") +
  xlab("Nivel del empleado") +
  ylab("Sueldo")
```

# Regresión Lineal polinómica:

#Paso 6. Se procede a crear el gráfico para enseñar el modelo polinómico y ver que es el más apropiado.

```
ggplot() +  
  geom_point(aes(x = dataset$Level , y = dataset$Salary),  
             color = "red") +  
  geom_line(aes(x = dataset$Level, y = predict(poly_reg, newdata = dataset)),  
            color = "blue") +  
  ggtitle("Predicción polinómica del sueldo en función del nivel del empleado") +  
  xlab("Nivel del empleado") +  
  ylab("Sueldo")
```

# Regresión Lineal polinómica:

#Paso 7. Visualización del modelo polinómico de una forma más "agradable" y sin tantos picos.

```
x_grid = seq(min(dataset$Level), max(dataset$Level), 0.1)
ggplot() +
  geom_point(aes(x = dataset$Level , y = dataset$Salary),
             color = "red") +
  geom_line(aes(x = x_grid, y = predict(poly_reg,
                                         newdata = data.frame(Level = x_grid,
                                                               Level2 = x_grid^2,
                                                               Level3 = x_grid^3,
                                                               Level4 = x_grid^4))),
            color = "blue") +
  ggtitle("Predicción polinómica del sueldo en función del nivel del empleado") +
  xlab("Nivel del empleado") +
  ylab("Sueldo (en $)")
```

# Regresión Lineal polinómica:

#Paso 8. Predicción de nuevos resultados con Regresión Lineal.

# Para ello se debe usar el objeto de regresión lineal con la función predict. El atributo "newdata" recibira en este caso, #la información del data frame cuyo nivel sea "6.5".

# Con esto lo que intenta es predecir cuál debe ser el valor de X (la variable dependiente, que en este caso es Salary)  
#cuando el nivel es 6.5 (columna Level)

```
y_pred = predict(lin_reg, newdata = data.frame(Level = 6.5))
```

#Tras ejecutar lo anterior, la predicción es de 330379, un valor que está muy por encima de la información real  
#debido a que se está usando un modelo lineal.

# Regresión Lineal polinómica:

#Paso 9. Predicción de nuevos resultados con Regresión Polinómica.

# En este caso se utiliza la regresión polinómica y la columna de nivel para el valor 6.5.

# Notar que se ha tenido que ir potenciando cada columna nueva.

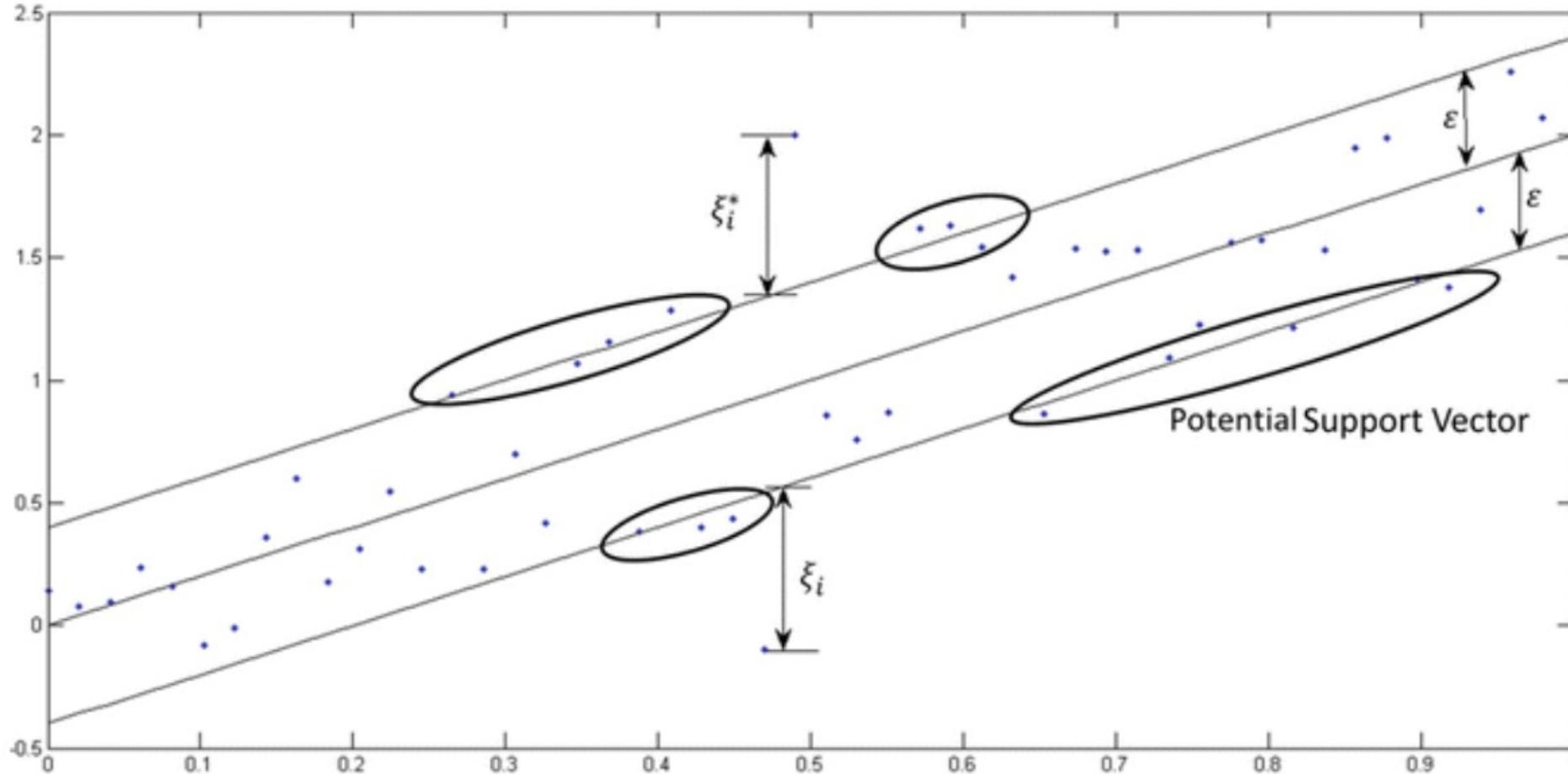
# Se puede apreciar que la predicción está mucho más adaptada y cercana a la información que se tiene.

```
y_pred_poly = predict(poly_reg, newdata = data.frame(Level = 6.5,  
                                         Level2 = 6.5^2,  
                                         Level3 = 6.5^3,  
                                         Level4 = 6.5^4))
```

# Regresión: SVM (Support Vector Machines)

- Las Máquinas de Soporte Vectorial sirven tanto para regresiones lineales como no lineales, por eso para el aprendizaje supervisado con regresión lineal se les llama SVR.
- En lugar de ajustar el mayor corredor (o la mayor calle) posible entre dos clases, ajustando el margen de los mismos como hacen en general las SVM, las SVR intentan mantener cuántas más observaciones posibles del conjunto de datos dentro del corredor en torno a la recta limitando unos márgenes máximos.
- La anchura del pasillo (o la calle) en torno a la recta se controla mediante un hiper parámetro llamado épsilon.

# Regresión: SVM (Support Vector Machines)



# Regresión: SVM (Support Vector Machines)

- La SVR tiene un objetivo de regresión diferente a la regresión lineal.
- En la regresión lineal se intenta minimizar el error entre la predicción y los datos.
- En SVR el objetivo es que los errores no superen el umbral establecido.

# Regresión: SVM (Support Vector Machines)

#Regresión con máquinas de soporte vectorial.

#Paso 1. Importar el dataset.

```
dataset = read.csv('Datasets/Position_Salaries.csv')  
dataset = dataset[, 2:3]
```

#Paso 2. Utilizar SVR con el conjunto de datos. Se debe importar la librería e1071.

#En este caso es conveniente ver la documentación de la función svm: ?svm

#El tipo será eps-regression tal como se ha visto en la documentación de la función.

#El kernel radial es el valor por defecto y funciona bien

#para el entrenamiento y predicción de datos que no siguen un comportamiento lineal.

```
install.packages("e1071")  
library(e1071)  
regression = svm(formula = Salary ~ .,  
                 data = dataset,  
                 type = "eps-regression",  
                 kernel = "radial")
```

# Regresión: SVM (Support Vector Machines)

#Paso 3. Generar la predicción de nuevos resultados con el SVM.

# El resultado indica que el modelo SVM es valido ya que para el Level indicado

# ha producido valor en el rango esperado entre los valores 6 y 7 de la columna "Level" del dataset.

```
y_pred = predict(regression, newdata = data.frame(Level = 6.5))
```

# Regresión: SVM (Support Vector Machines)

#Paso 4. Visualización del modelo de SVR.

#El modelo se adapta muy bien, la mayoría de los puntos de observación se encuentran bastante cerca de la curva.

#Se puede observar que el modelo funciona para casi todos pero hay puntos de observación

#que no se gestionan adecuadamente ya que son irregulares o atípicos, los cuales son menospreciados por el modelo

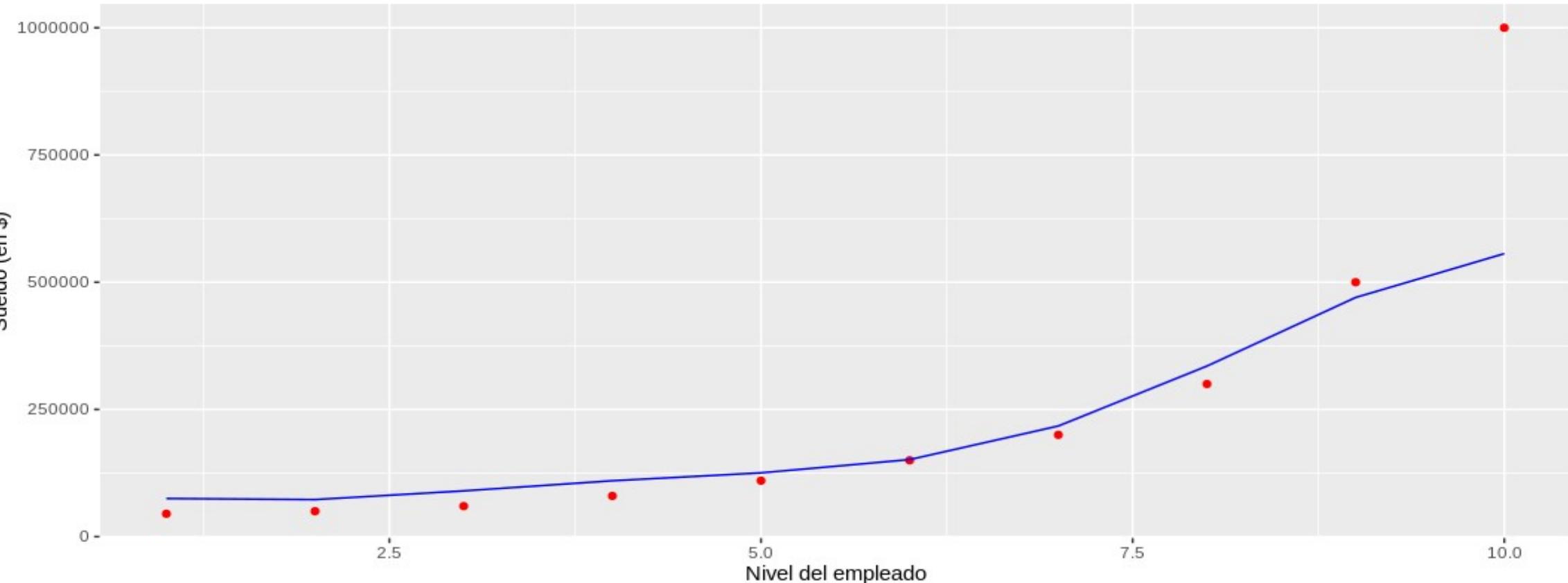
#ya que precisamente, lo que intenta es definir vectores de soporte que cubran la mayor cantidad de puntos de observación,

#pero en ocasiones no pueden cubrirlos todos.

```
# install.packages("ggplot2")
library(ggplot2)
x_grid = seq(min(dataset$Level), max(dataset$Level), 0.1)
ggplot() +
  geom_point(aes(x = dataset$Level , y = dataset$Salary),
             color = "red") +
  geom_line(aes(x = dataset$Level, y = predict(regression,
                                                 newdata = data.frame(Level = dataset$Level))),
            color = "blue") +
  ggtitle("Predicción (SVR)") +
  xlab("Nivel del empleado") +
  ylab("Sueldo (en $)")
```

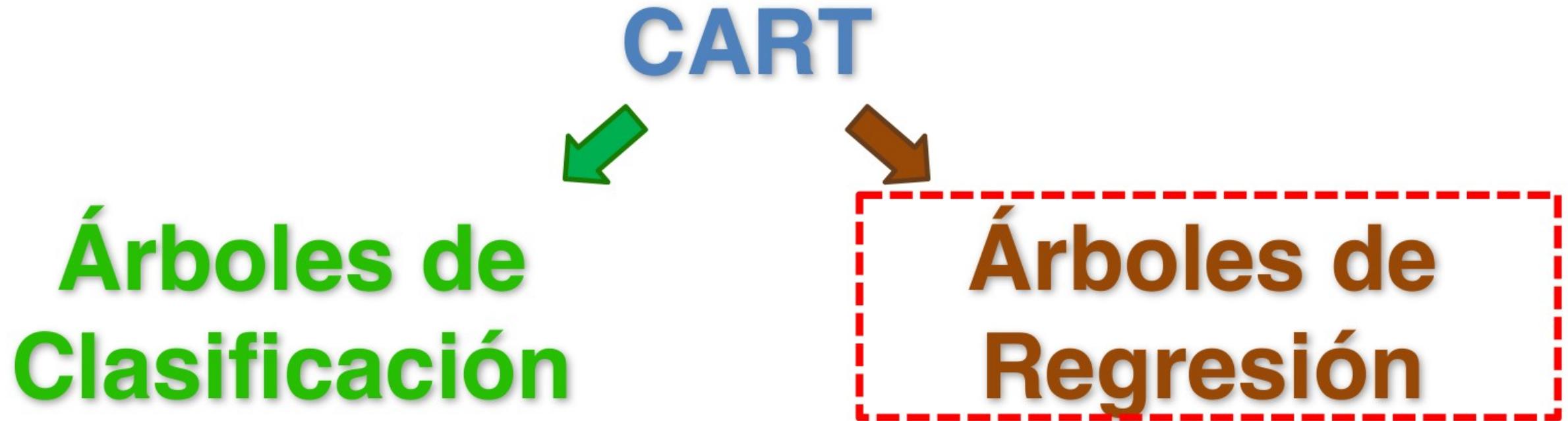
# Regresión: SVM (Support Vector Machines)

Predicción (SVR)

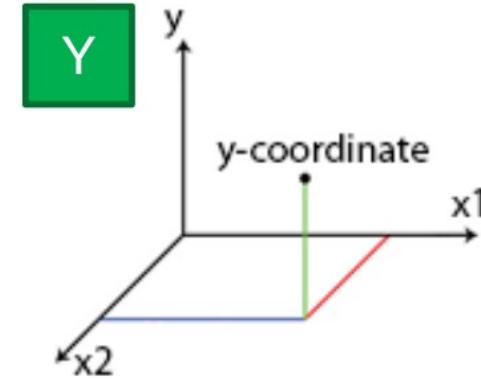
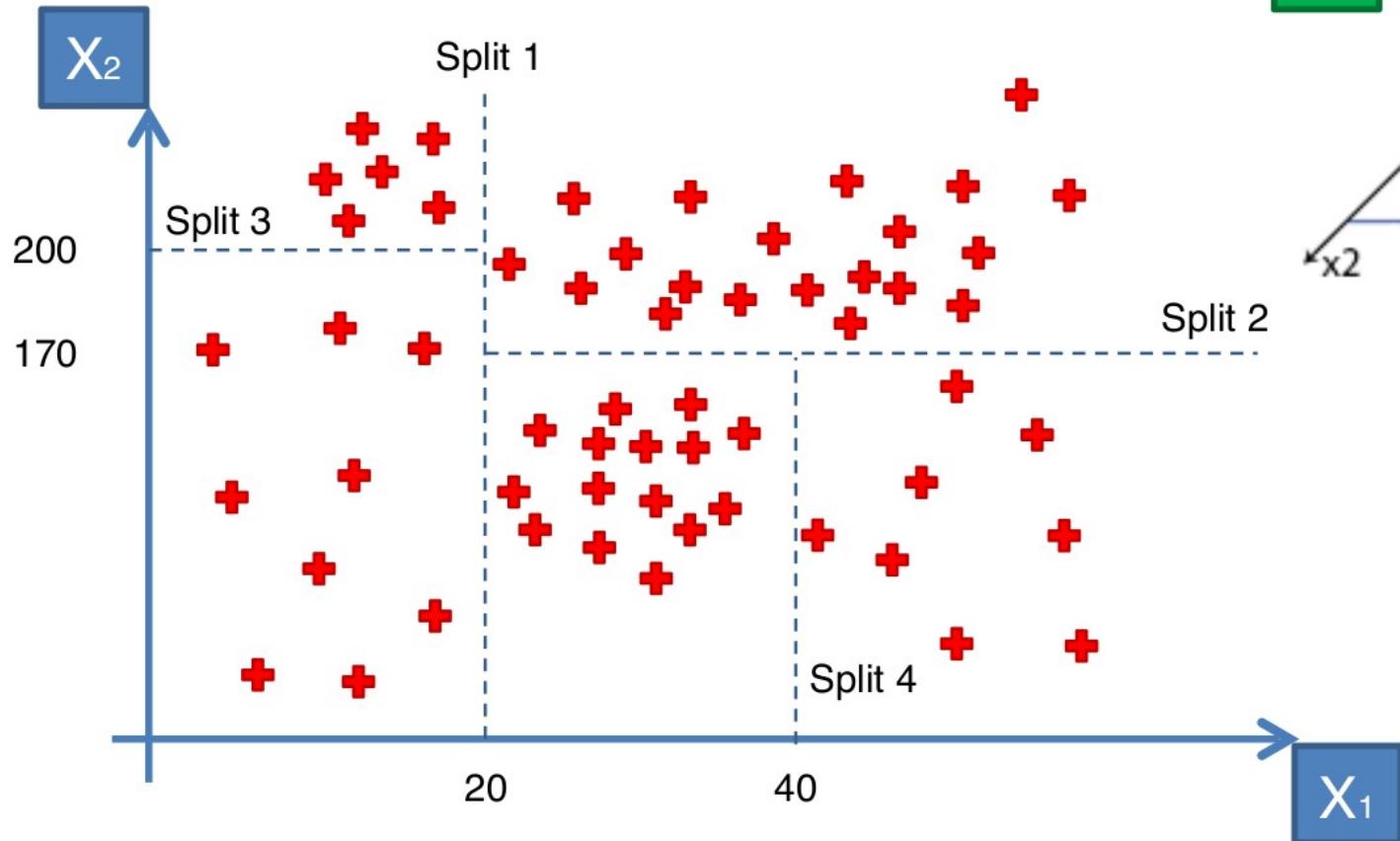


# Regresión: Árboles de decisión

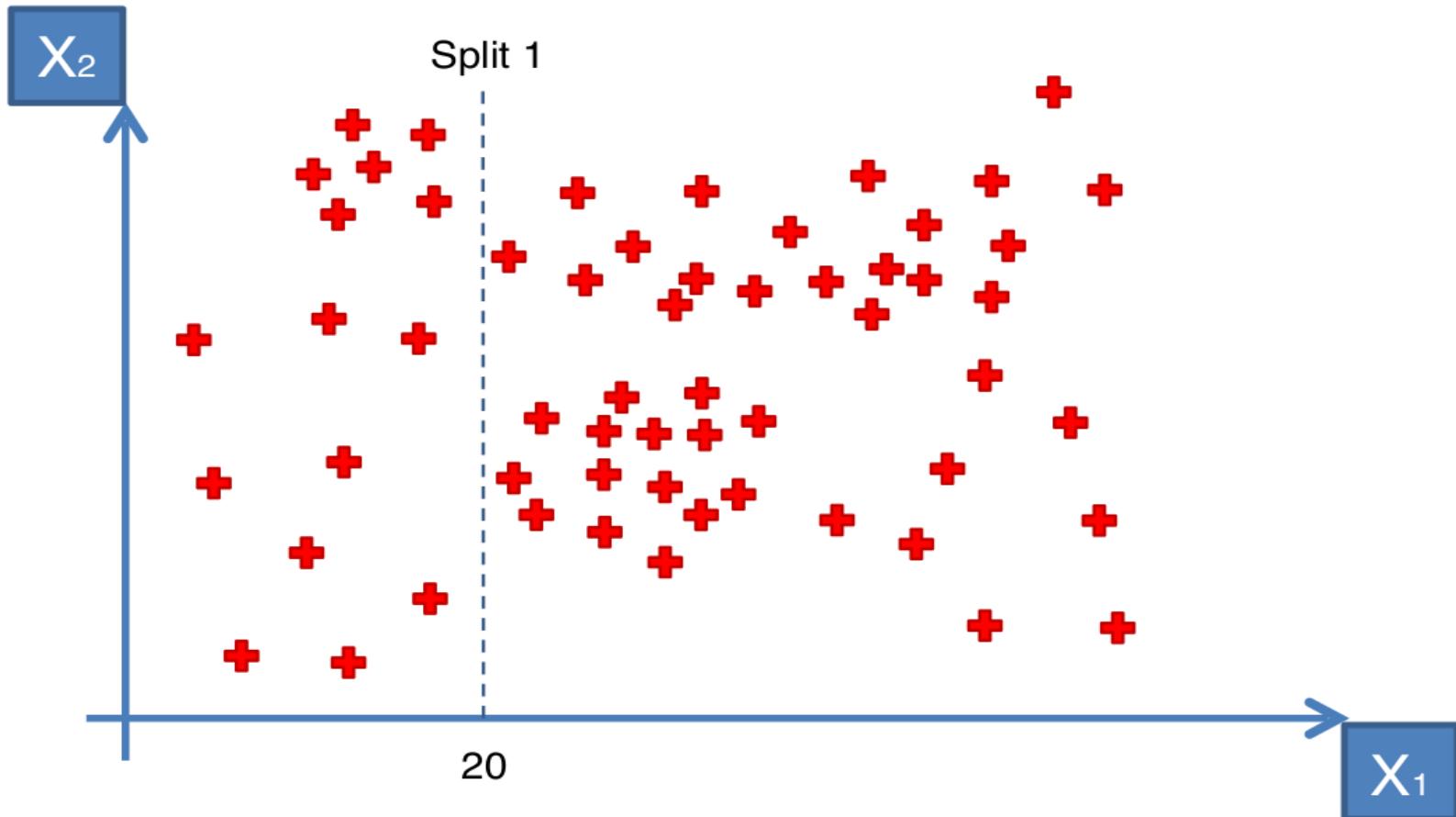
CART → Classification And Regression Trees



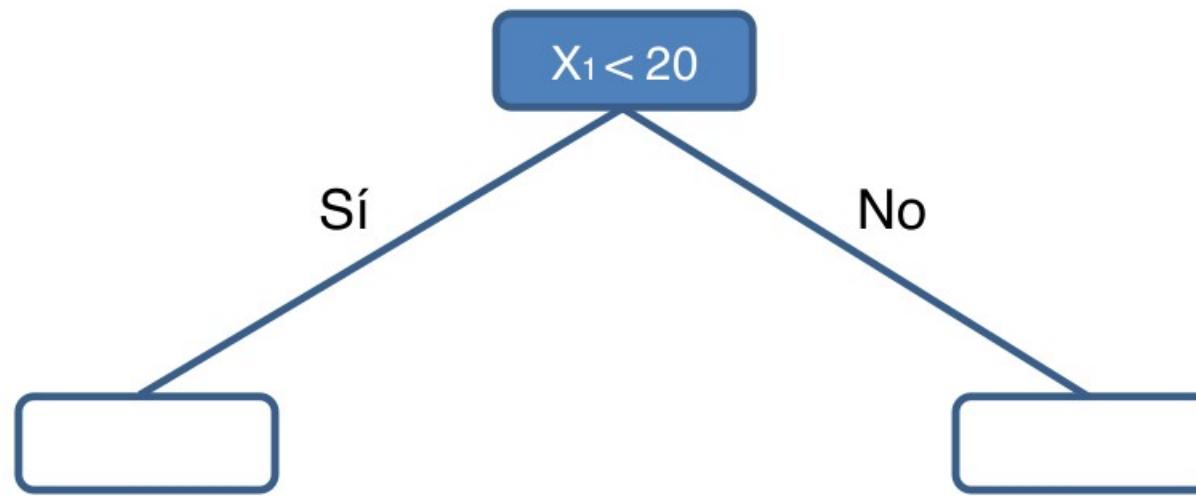
# Regresión: Árboles de decisión



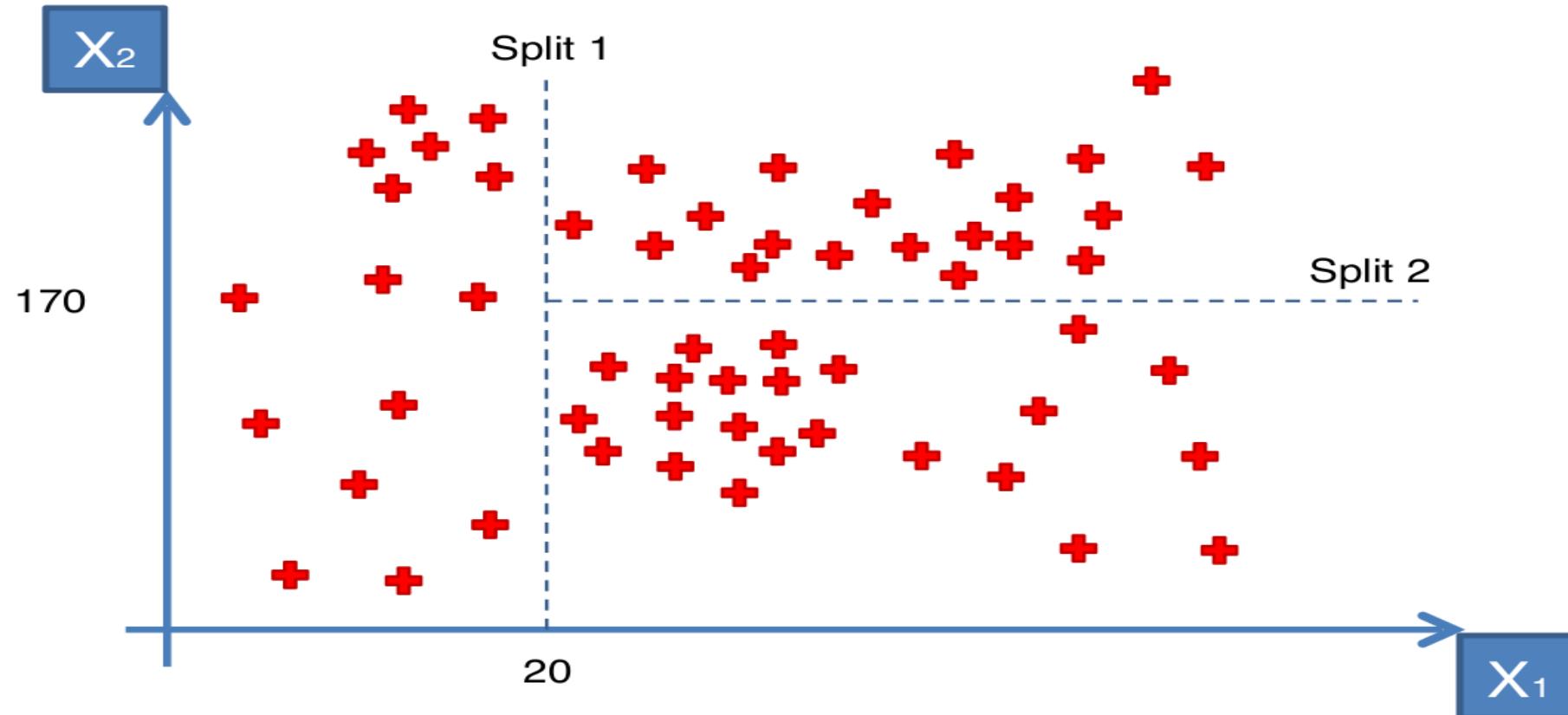
# Regresión: Árboles de decisión: Ejemplo visual



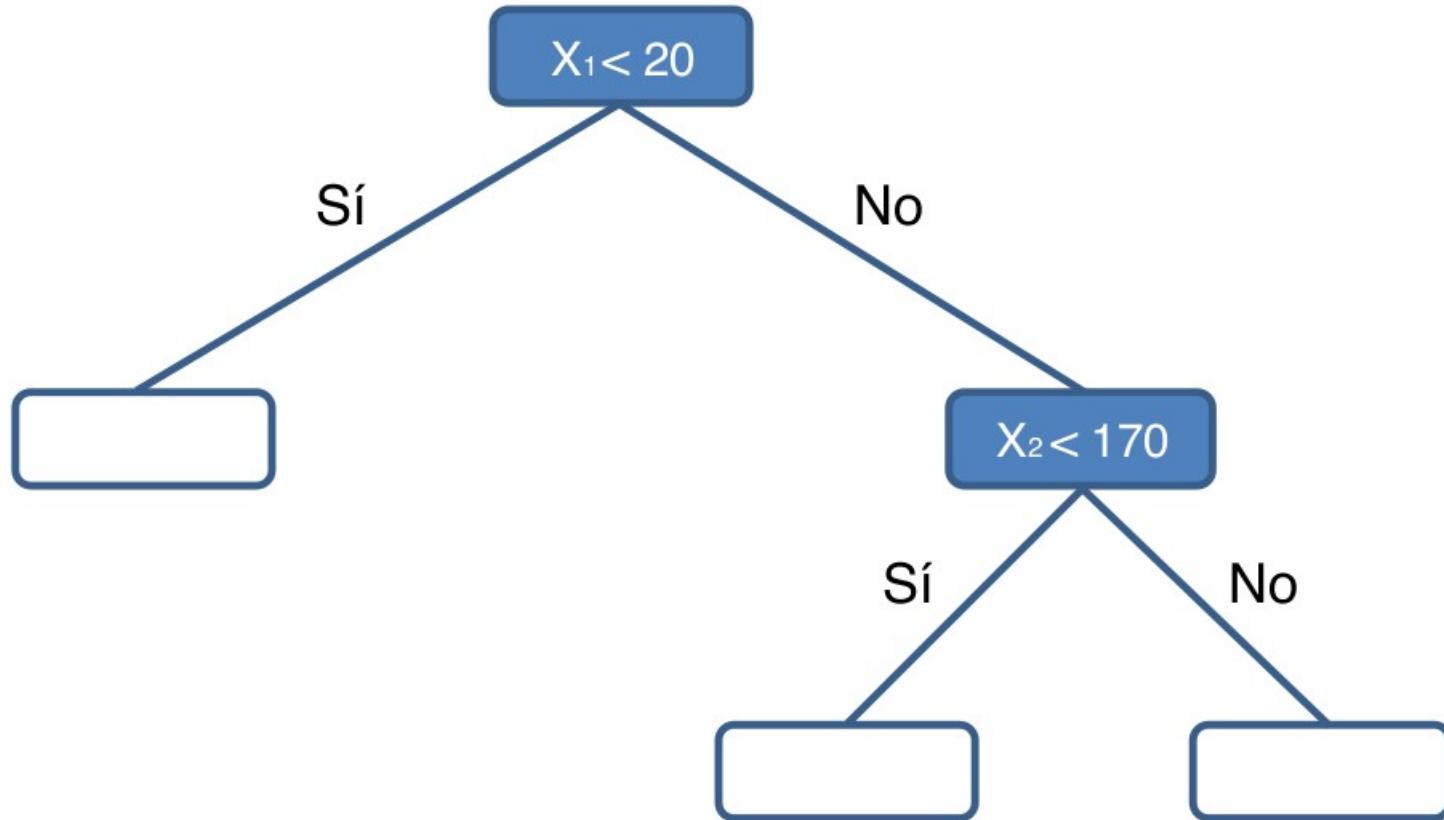
# Regresión: Árboles de decisión: Ejemplo visual



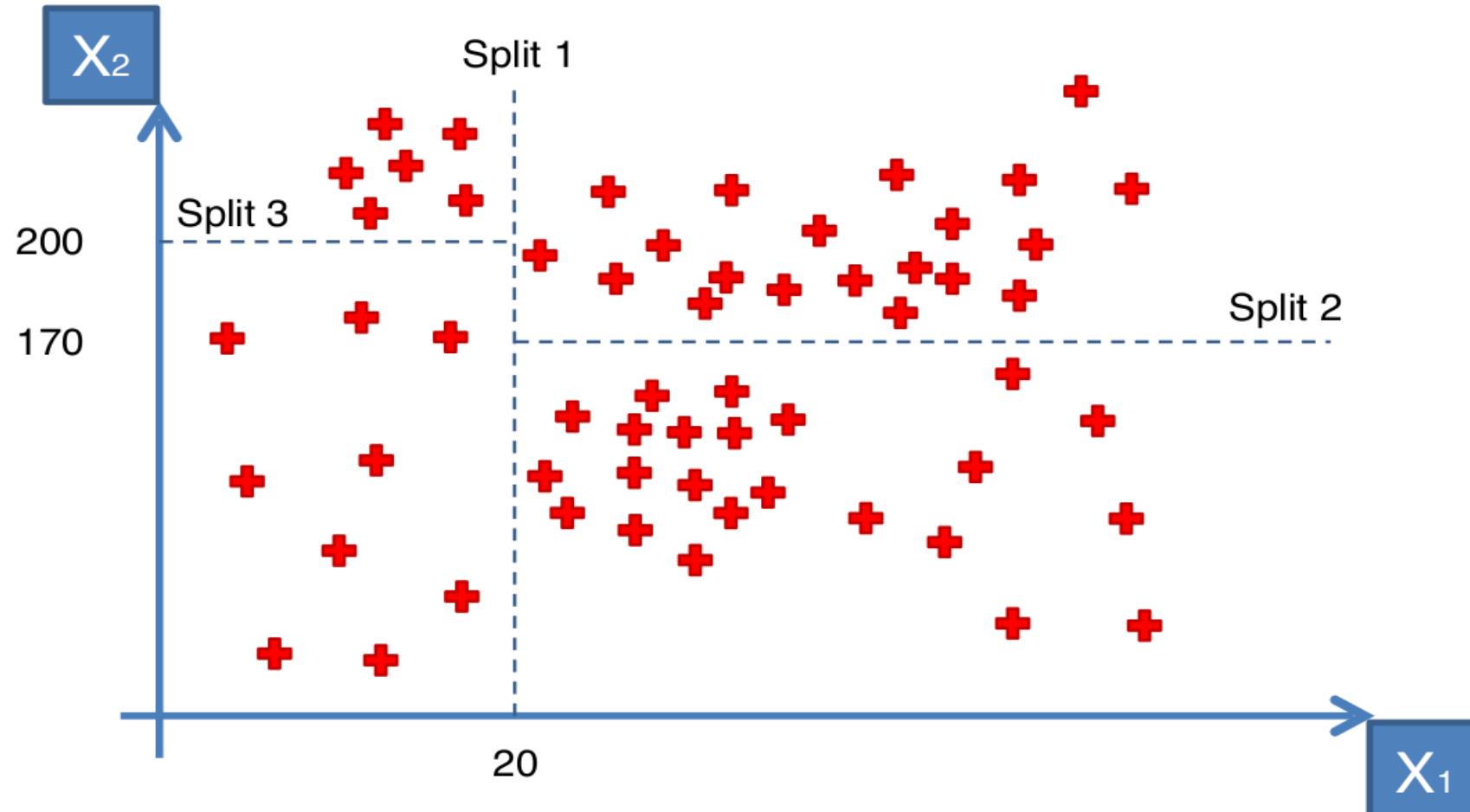
# Regresión: Árboles de decisión: Ejemplo visual



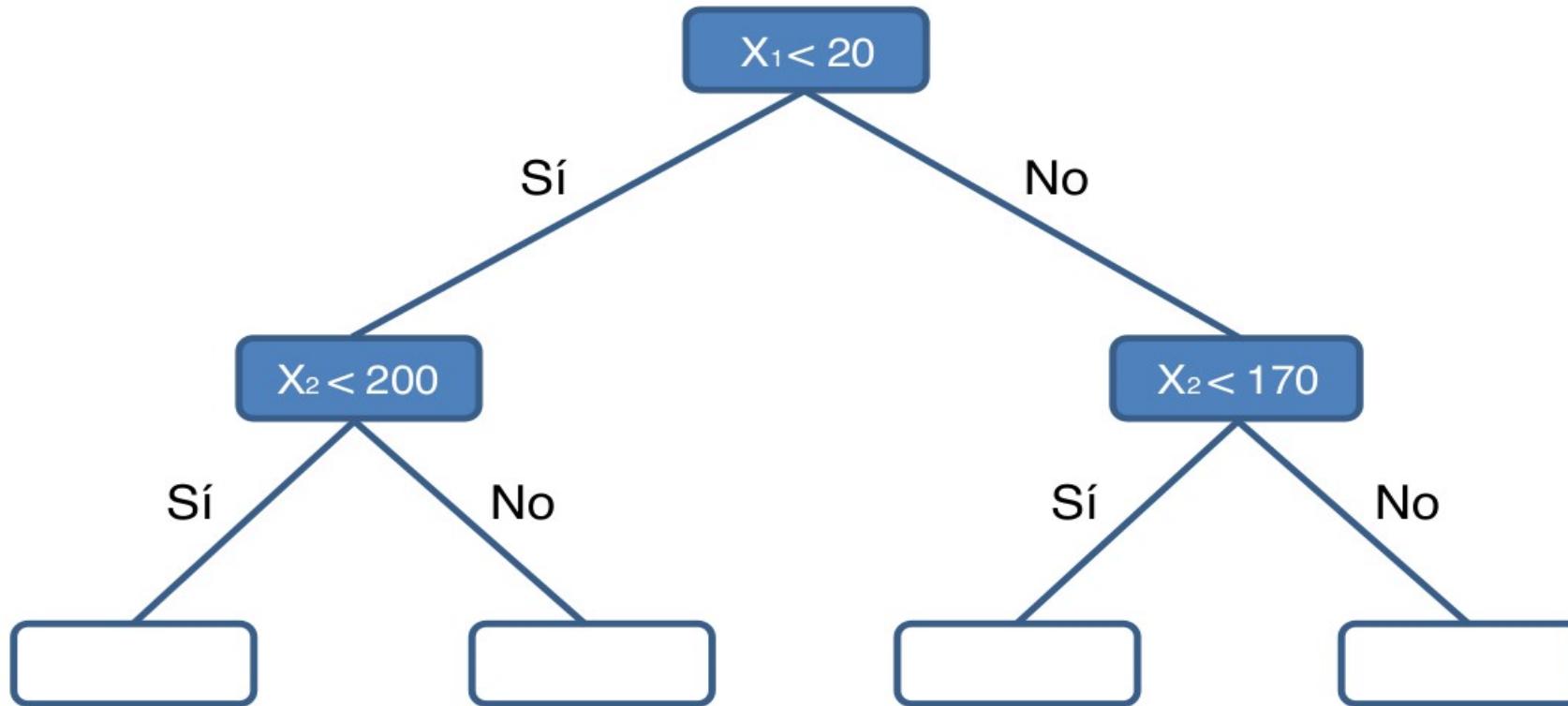
# Regresión: Árboles de decisión: Ejemplo visual



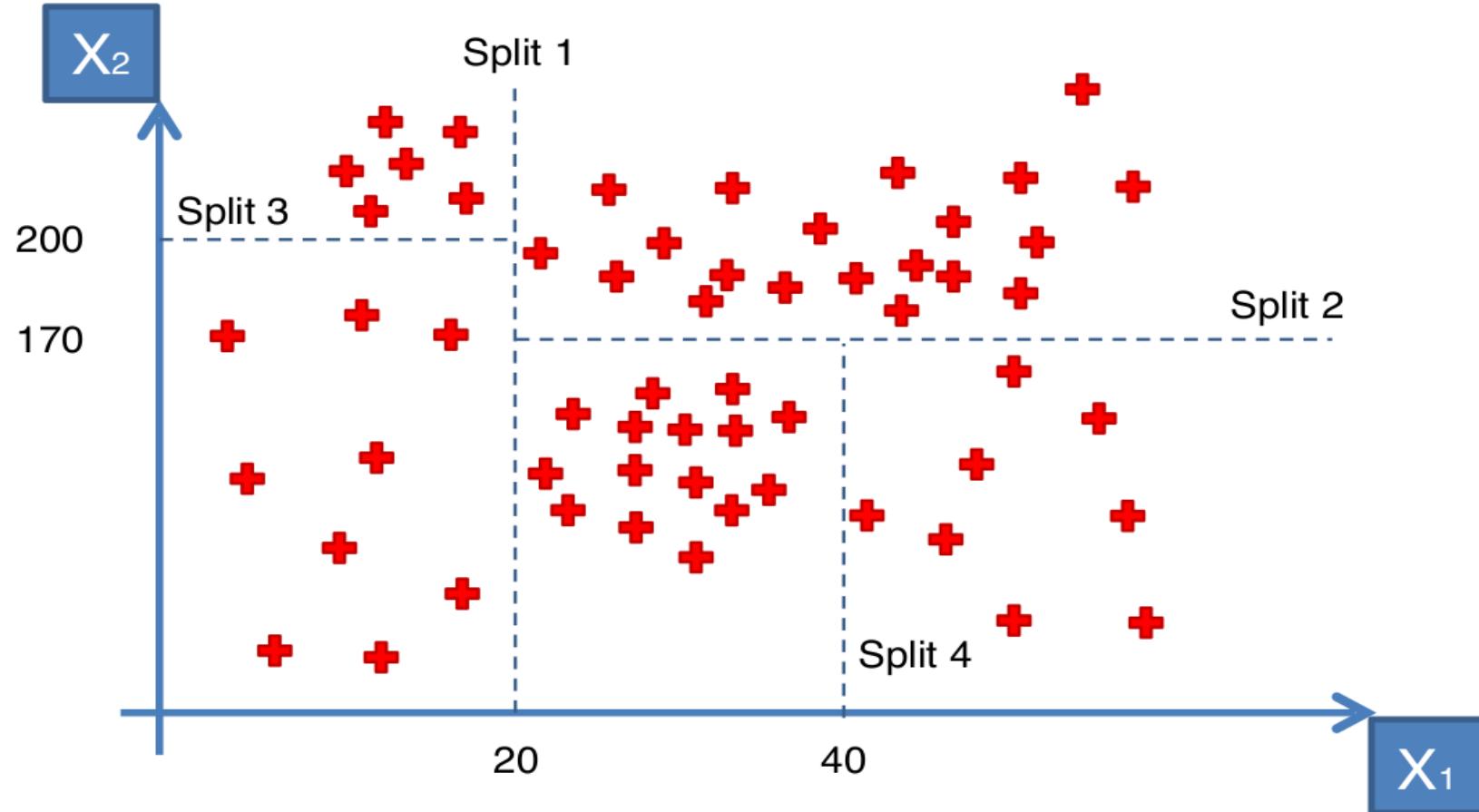
# Regresión: Árboles de decisión: Ejemplo visual



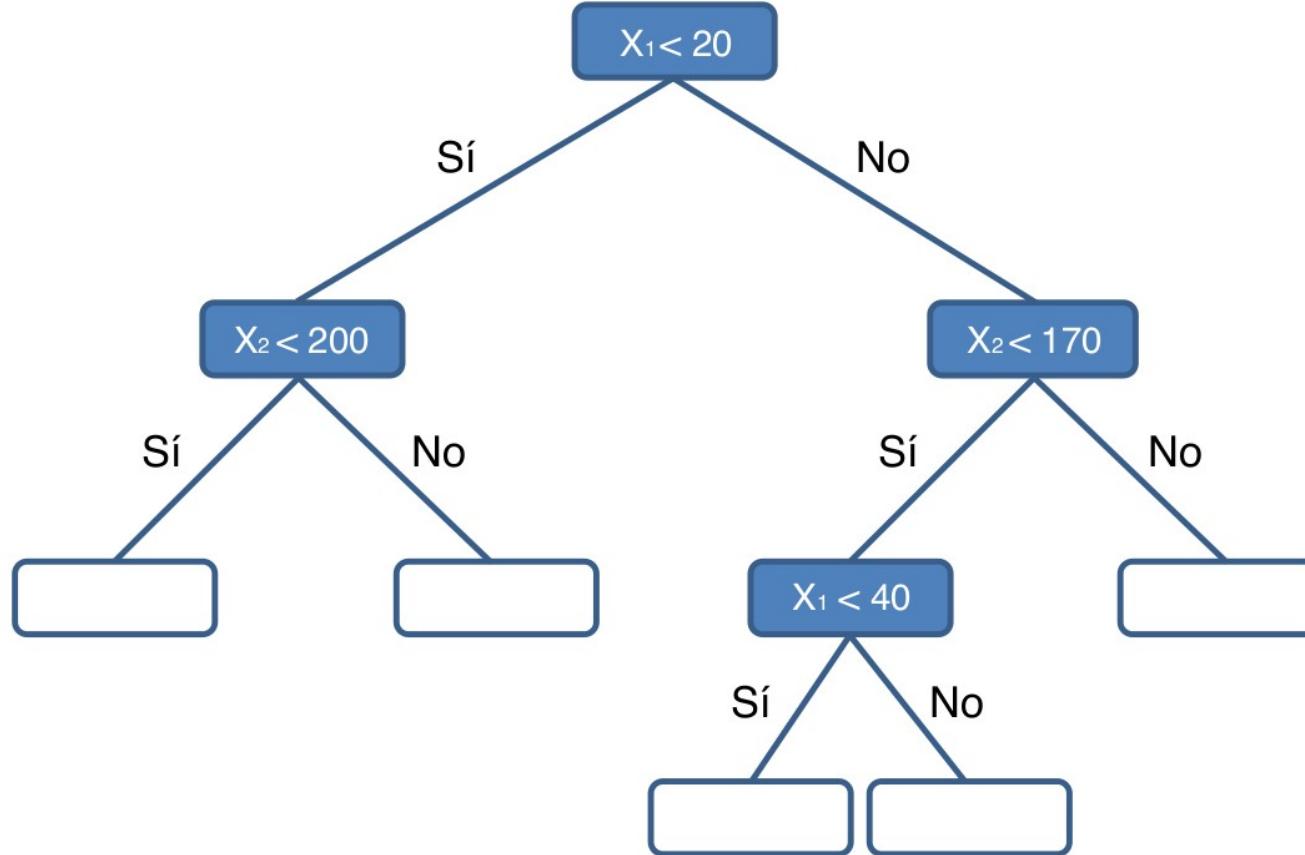
# Regresión: Árboles de decisión: Ejemplo visual



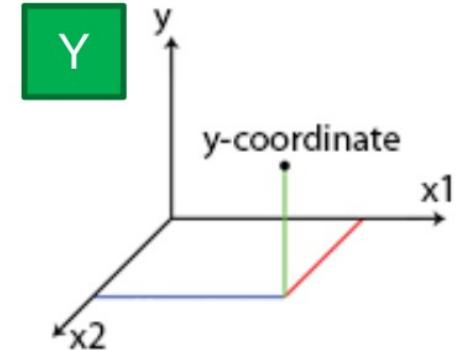
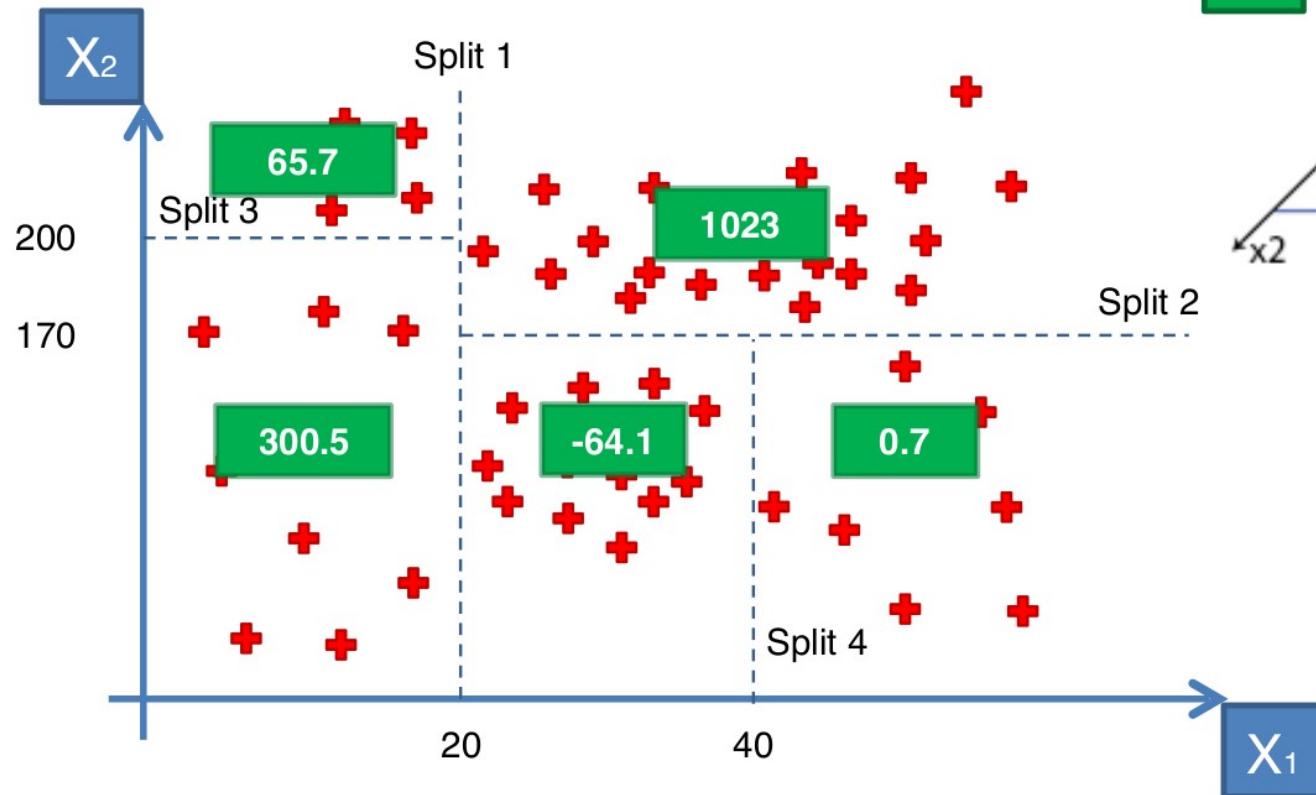
# Regresión: Árboles de decisión: Ejemplo visual



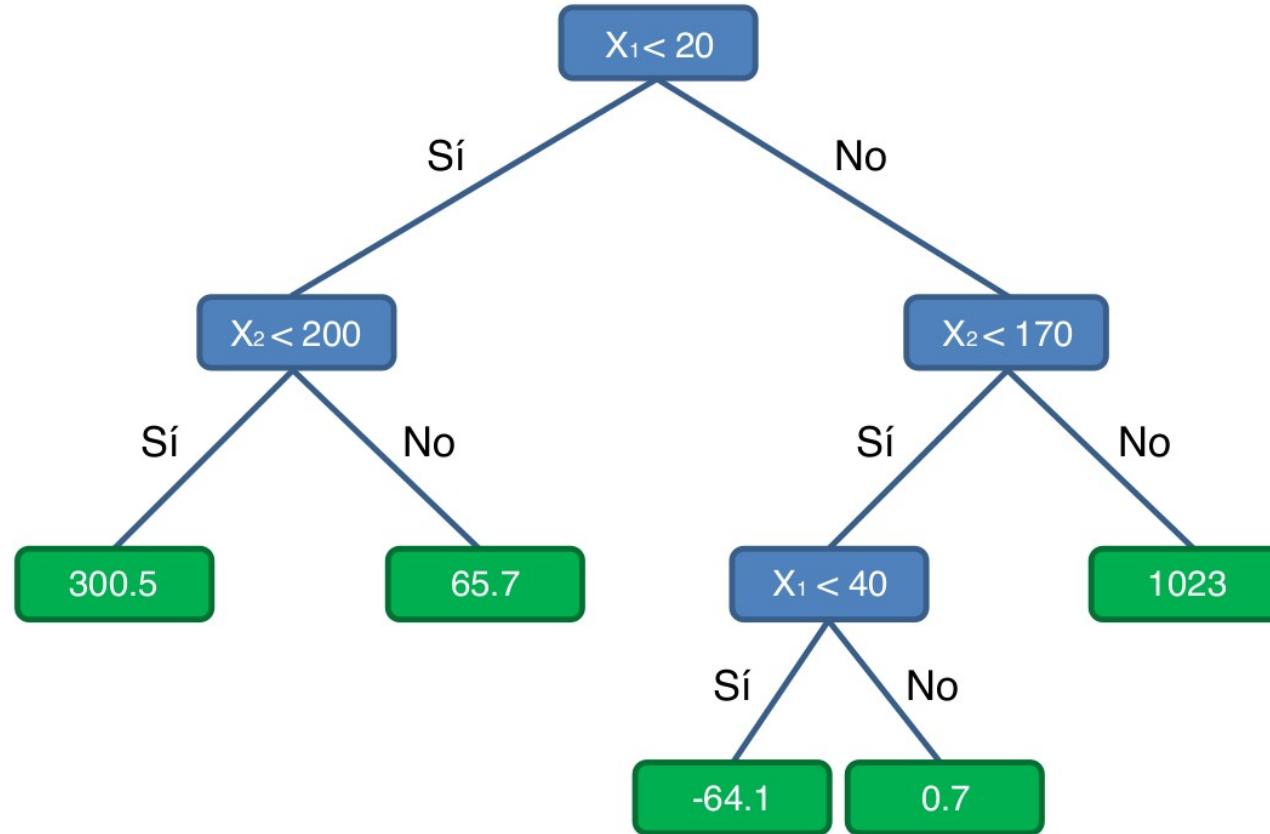
# Regresión: Árboles de decisión: Ejemplo visual



# Regresión: Árboles de decisión: Ejemplo visual



# Regresión: Árboles de decisión: Ejemplo visual



# Regresión: Árboles de decisión

#Regresión con árbol de decisión

# Paso 1. Importar el dataset

```
dataset = read.csv('Datasets/Position_Salaries.csv')
```

```
dataset = dataset[, 2:3]
```

# Paso 2. Se debe instalar la librería "rpart" la cual permite utilizar árboles de decisión para regresión.

#La librería cuenta con la función "rpart" que permite utilizar los parámetros formula y dataset, que son comunes en la función "lm"

#Además de dichos parámetros, también se cuenta con el parámetro "control".

#El control que se permite aplicar incluye el número mínimo o máximo de observaciones que deben existir en una hoja entre otras cosas.

```
install.packages("rpart")
```

```
library(rpart)
```

```
regression = rpart(formula = Salary ~ .,  
                    data = dataset,  
                    control = rpart.control(minsplit = 5))
```

# **Regresión: Árboles de decisión**

# Paso 3. Predicción de nuevos resultados con Árbol de Regresión.

# Se puede cambiar varias veces el "Level" y se podrá ver que la predicción generada no es tan buena como se espera.

# Esto se debe a que este modelo no funciona del todo bien con el conjunto de datos suministrado.

```
y_pred = predict(regression, newdata = data.frame(Level = 6.5))
```

# Regresión: Árboles de decisión

#Paso 4. Se genera la gráfica para comprobar el comportamiento del modelo con las predicciones y los diferentes puntos de observación que se encuentran en el dataset original.

```
# Visualización del modelo de árbol de regresión
```

```
# install.packages("ggplot2")
```

```
library(ggplot2)
```

```
x_grid = seq(min(dataset$Level), max(dataset$Level), 0.1)
```

```
ggplot() +
```

```
  geom_point(aes(x = dataset$Level , y = dataset$Salary),  
             color = "red") +
```

```
  geom_line(aes(x = dataset$Level, y = predict(regression,  
                                                 newdata = data.frame(Level = dataset$Level))),
```

```
             color = "blue") +
```

```
  ggtitle("Predicción con Árbol de Decisión (Modelo de Regresión)") +
```

```
  xlab("Nivel del empleado") +
```

```
  ylab("Sueldo (en $)")
```

# Regresión: Random Forest (Bosques aleatorios).

**PASO 1:** Elegir un número aleatorio K de puntos de datos del Conjunto de Entrenamiento.



**PASO 2:** Construir el Árbol de Decisión asociado a esos K Puntos de Datos.



**PASO 3:** Elegir el número **Ntree** de árboles que queremos construir y repetimos los PASOS 1 y 2.



**PASO 4:** Para un nuevo punto de datos, hacer que cada uno de los **Ntree** árboles haga una predicción del valor de Y para el punto en cuestión, y asigne al nuevo punto la predicción final basada en el promedio de todas las predicciones Y de los árboles.

# Regresión: Random Forest (Bosques aleatorios).

#Regresión de bosques aleatorios.

# Paso 1. Importar el dataset

```
dataset = read.csv('Datasets/Position_Salaries.csv')
dataset = dataset[, 2:3]
```

# Paso 2. Ajustar modelo de Random Forest con el conjunto de datos. En primer lugar se debe instalar el paquete correspondiente.

# Ver la documentación la función ?randomForest. Es necesario indicar en "x" las variables independientes o la matriz de características # y en "y" las variables dependientes o a predecir.

# Es importante tener en cuenta que en "x" se espera una matriz, array o dataset mientras que en "y" un vector.

# Por lo tanto no se puede enviar en "x" lo que se ha visto en los otros modelos con dataset\$Level porque dicha variable no es un dataframe.

```
# > class(dataset$Level)
```

```
# [1] "integer"
```

```
# > class(dataset[1])
```

```
# [1] "data.frame"
```

# Por otro lado, el parámetro "ntree" permite indicar al algoritmo el número de árboles que debe generar.

#Se recomienda ir cambiando este valor e ir aumentando para apreciar el comportamiento del modelo.

```
install.packages("randomForest")
library(randomForest)
set.seed(1234)
regression = randomForest(x = dataset[,1],
                           y = dataset$Salary,
                           ntree = 10)
```

# Regresión: Random Forest (Bosques aleatorios).

# Paso 3. Predicción de nuevos resultados con bosques aleatorios.

#Hay que tener en cuenta que el modelo y su correspondiente predicción cambian en función al número de árboles que se utilizan.

#Si se va incrementando (en realidad el algoritmo no tiene un límite fijo para el parámetro "ntree") la predicción puede mejorar hasta cierto punto,

#pero puede ocurrir que al seguir aumentando el número de árboles las decisiones de estos sean estadísticamente menos significativas

#ya que el modelo de bosques aleatorios utiliza la mediana del conjunto total.

#Como efecto lateral, al incluir un número de árboles de decisión demasiado elevado, la predicción puede no ser tan exacta.

```
y_pred = predict(regression, newdata = data.frame(Level = 7))
```

# Paso 4. Visualización del modelo de Random Forest

```
# install.packages("ggplot2")
library(ggplot2)
x_grid = seq(min(dataset$Level), max(dataset$Level), 0.01)
ggplot() +
  geom_point(aes(x = dataset$Level , y = dataset$Salary),
             color = "red") +
  geom_line(aes(x = x_grid, y = predict(regression,
                                         newdata = data.frame(Level = x_grid))),
            color = "blue") +
  ggtitle("Predicción (Random Forest)") +
  xlab("Nivel del empleado") +
  ylab("Sueldo (en $)")
```

# FAQs sobre modelos de regresión para aprendizaje supervisado.

**¿Cuáles son los pros y los contras de cada modelo?**

**¿Cómo sé qué modelo debo elegir para resolver mi problema?**

**¿Cómo puedo mejorar cada uno de estos modelos ?**

# ¿Cuáles son los pros y los contras de cada modelo?.

Regression Model	Pros	Cons
Linear Regression	Works on any size of dataset, gives informations about relevance of features	The Linear Regression Assumptions
Polynomial Regression	Works on any size of dataset, works very well on non linear problems	Need to choose the right polynomial degree for a good bias/variance tradeoff
SVR	Easily adaptable, works very well on non linear problems, not biased by outliers	Compulsory to apply feature scaling, not well known, more difficult to understand
Decision Tree Regression	Interpretability, no need for feature scaling, works on both linear / nonlinear problems	Poor results on too small datasets, overfitting can easily occur
Random Forest Regression	Powerful and accurate, good performance on many problems, including non linear	No interpretability, overfitting can easily occur, need to choose the number of trees

# ¿Cómo sé qué modelo debo elegir para resolver mi problema?

Es necesario averiguar si el problema es o no es lineal. Una vez definido esto,

Si el problema es lineal se debe intentar crear un modelo de Regresión Lineal Simple si solo hay una variable independiente o un modelo de Regresión Lineal Múltiple en el caso de tener varias.

Si el problema no es lineal, entonces hay que probar las diferentes técnicas como la Regresión Polinómica, SVR, Árboles de Decisión y Bosques Aleatorios. Para decidir cuál funcionará mejor, se utiliza el método llamado ***k-Fold Cross Validation***, y elegir el modelo que demuestre mejores resultados.

## ¿Cómo puedo mejorar cada uno de estos modelos ?

Se deben ajustar los parámetros que permiten mejorar la eficacia de los modelos. Existen dos tipos de parámetros:

- **parámetros normales:** Son aquellos en los que el modelo aprende, como los coeficientes de la regresión lineal.
- **los hiperparámetros del algoritmo:** Son parámetros en los que el algoritmo no aprende, si no que son fijos y forman parte de las ecuaciones de los modelos. Encontrar el valor óptimo es parte del ajuste de parámetros.

# Modelos de clasificación.

- Se trata de modelos en los que se pretende agrupar las observaciones del modelo de datos por características comunes.
- El objetivo de los modelos de clasificación clasificar la mayor cantidad de observaciones de una forma en la que el aprendizaje pueda llevarse a cabo utilizando patrones, que es precisamente la forma en el que mejor aprende el ser humano.
- Dependiendo del modelo utilizando pueden generarse sesgos o comportamientos aprendidos que no reflejan la realidad, por este motivo se trata de modelos en los que cuanta más información se pueda utilizar para entrenamiento y pruebas, mejor.
- Los modelos de este tipo son especialmente interesantes en el mundo del machine learning dada su aplicación práctica

## Modelos de clasificación.

- Los modelos de este tipo son especialmente interesantes en el mundo del machine learning dada su aplicación práctica.
  - Clasificar si el correo que llega es spam o no.
  - Un conjunto de datos clasificar condiciones medicas.
  - A partir de un conjunto de muestras de clientes de un banco, conceder un crédito o no a un cliente.
  - Partiendo del historial de conducta criminal de varios presos determinar si un ex-convicto volverá a cometer algún delito o no.

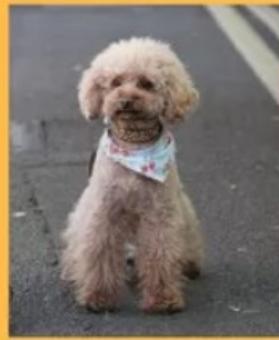
# Modelos de clasificación: Problemas de underfitting.

## Underfitting

Entreno al modelo con  
1 sola raza de perro



Muestra nueva:  
¿Es perro?



La máquina fallará en reconocer al perro por falta de suficientes muestras. No puede generalizar el conocimiento.

# Modelos de clasificación: Problemas de overfitting.

## Overfitting

Entreno al modelo con  
10 razas de perro color marrón



Muestra nueva:  
¿Es perro?



La máquina fallará en reconocer un perro nuevo porque no tiene  
estrictamente los mismos valores de las muestras de  
entrenamiento.

# Modelos de clasificación: Regresión logística.

- La Regresión Logística es un método de regresión que permite estimar la probabilidad de una variable cualitativa binaria en función de una variable cuantitativa.
- Una de las principales aplicaciones de la regresión logística es la de clasificación binaria, en el que las observaciones se clasifican en un grupo u otro dependiendo del valor que tome la variable empleada como predictor (variable independiente).
- Por ejemplo, clasificar a un individuo desconocido como hombre o mujer en función del tamaño de la mandíbula.

# Modelos de clasificación: Regresión logística.

# Regresión Logística para predicción y clasificación.

#PASO 1. Importar el dataset

```
dataset = read.csv('Datasets/Social_Network_Ads.csv')  
dataset = dataset[, 3:5] #Se seleccionan las 3 últimas columnas: Age, EstimatedSalary, Purchased
```

# Dividir los datos en conjunto de entrenamiento y conjunto de test

```
# install.packages("caTools")  
library(caTools)  
set.seed(123)  
split = sample.split(dataset$Purchased, SplitRatio = 0.75) #300 observaciones para entrenar y las 100 restantes para probar.  
training_set = subset(dataset, split == TRUE)  
testing_set = subset(dataset, split == FALSE)
```

# Modelos de clasificación: Regresión logística.

#PASO 2. Escalado de valores.

# En este caso sí es necesario escalar ya que los datos que se encuentran en la columna de edad son muy bajos con respecto a los # que hay en la columna de EstimatedSalary.

# Es decir, ambas columnas tienen escalas diferentes y se hace necesario normalizar.

```
training_set[,1:2] = scale(training_set[,1:2])
```

```
testing_set[,1:2] = scale(testing_set[,1:2])
```

# Modelos de clasificación: Regresión logística.

#Paso 3. Ajustar el modelo de regresión logística con el conjunto de entrenamiento.

# En este caso en lugar de crear una "regresión" se crea una "clasificación" la cual se puede obtener por medio de la función `glm`.

# La sintaxis de `glm` (Generalized Lineal Model) es similar "`lm`" pero se debe especificar el parámetro "family" el cual por defecto es gausiano.

# En este caso, dado que se intenta predecir si la variable dependiente "Purchased" es 1 o 0, se utiliza la familia "binomial".

```
classifier = glm(formula = Purchased ~ .,  
                 data = training_set,  
                 family = binomial)
```

# Modelos de clasificación: Regresión logística.

#Paso 4. Predicción de los resultados con el conjunto de testing

# Se evalua con el modelo generado en el paso anterior los datos testing.

# En este caso el algoritmo cambia un poco comparado con los vistos en sobre regresión ya que ahora se predice una probabilidad entre 0 y 1.

# La función "predict" recibe el clasificador, tipo y datos de testing.

# El clasificador representa el modelo generado con la función "glm" anteriormente.

# El tipo recibirá el parámetro "response" para que devuelva el conjunto de probabilidades para el dataset en un solo vector el cual posteriormente

# se podrá traducir a "venta" o "no venta".

# El newdata representa el conjunto de datos de testing (training\_set).

# En este caso se elimina la columna "Purchased" ya que es precisamente la que se pretende predecir (variable dependiente)

```
prob_pred = predict(classifier, type = "response",
```

```
    newdata = testing_set[,-3])
```

# Ahora se puede verificar el valor de la variable "prob\_pred" desde la terminal, de esta manera se puede comprobar que la predicción aporta datos estadísticos

# sobre cada uno de los registros del conjunto de pruebas.

# Con dicha información, se compara con el dataset de testing original para verificar la probabilidad y el dato real de "compra" o "no compra".

# Modelos de clasificación: Regresión logística.

#Paso 5. Se utiliza la instrucción "ifelse" que trabajará sobre el vector de predicción.

# la condición concretamente es "prob\_pred > 0.5" la cual si se cumple, devolverá 1 y sino 0.

#Hay que tener en cuenta que devolverá un vector con un 0 o 1 para cada uno de los registros.

```
y_pred = ifelse(prob_pred > 0.5, 1, 0)
```

#Paso 6. Crear la matriz de confusión.

#La matriz de confusión es un término que se utiliza para determinar el nivel de error hacia arriba o hacia abajo.

#La función table se encarga de comprobar los resultados de dos vectores.

#En este caso concreto interesa enviar el vector correspondiente a los valores de la columna 3 del conjunto de testing (Columna Purchased)

#Y el vector de predicción generado en la línea anterior.

#La función table realiza una agrupación y suma los valores de 1 (compra) y 0 (no compra).

```
cm = table(testing_set[, 3], y_pred)
```

#Después de ejecutar la instrucción anterior se puede ver por la terminal:

```
# > cm = table(testing_set[, 3], y_pred)
```

```
# > cm
```

```
# y_pred
```

```
# 0 1
```

```
# 0 57 7
```

```
# 1 10 26
```

#Los resultados son sencillos de interpretar.

# Significa que 57 realmente no han comprado y 26 sí lo han hecho.

# Es decir, que 57 registros del conjunto de testing se han predecido correctamente con un 0 y 26 se han predecido correctamente con 1.

# Esto significa que el porcentaje de predicción correcto ha sido del 87% frente a un 13% incorrecto.

# Modelos de clasificación: Regresión logística.

# Paso 7. Visualización del conjunto de entrenamiento.

# Con el siguiente código se generará una nube de puntos en donde se podrán ver las observaciones del conjunto de entrenamiento.

# Hay una linea que divide al conjunto rojo del verde, indicando cuales han "comprado" (1) y cuales no (0).

# Se pondrán ver algunas observaciones en verde que aparecen en el cuadrante rojo y viceversa, representan precisamente aquellas predicciones incorrectas

# que ha generado el modelo, es decir, aquellas que el modelo ha predicho que no ha comprado (0) pero en el dataset se indica que sí y viceversa.

```
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
prob_set = predict(classifier, type = 'response', newdata = grid_set)
y_grid = ifelse(prob_set > 0.5, 1, 0)
plot(set[, -3],
     main = 'Clasificación (Conjunto de Entrenamiento)',
     xlab = 'Edad', ylab = 'Sueldo Estimado',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

# Modelos de clasificación: Regresión logística.

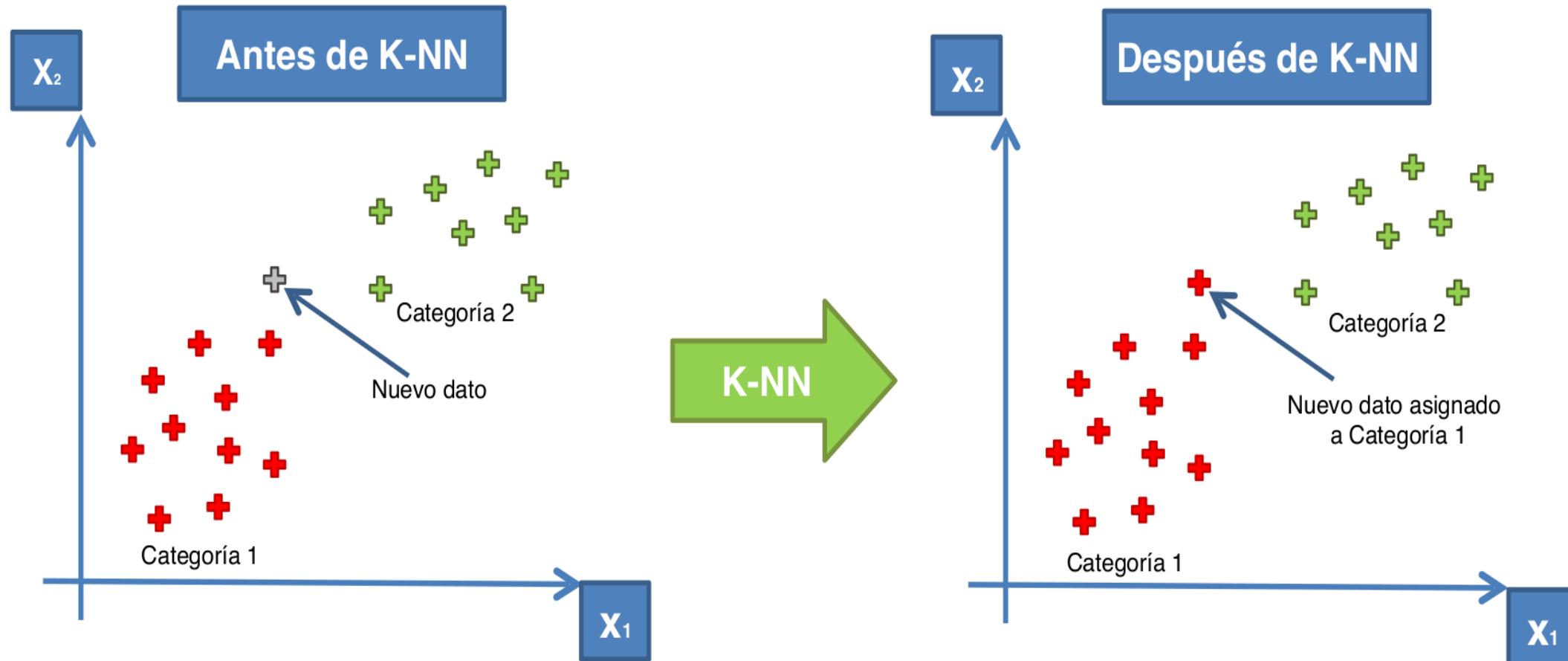
#Punto 7.1 Visualización del conjunto de testing

```
set = testing_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
prob_set = predict(classifier, type = 'response', newdata = grid_set)
y_grid = ifelse(prob_set > 0.5, 1, 0)
plot(set[, -3],
      main = 'Clasificación (Conjunto de Testing)',
      xlab = 'Edad', ylab = 'Sueldo Estimado',
      xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

# Clasificación: K-NN (K-Nearest-Neighbor).

- Sirve esencialmente para clasificar valores buscando los puntos de datos “más similares” (por cercanía utilizando la distancia euclídea) aprendidos en la etapa de entrenamiento.
- Es un método que simplemente busca en las observaciones más cercanas a la que se está tratando de predecir y clasifica el punto de interés basado en la mayoría de datos que le rodean.

# Clasificación: K-NN (K-Nearest-Neighbors).



# Clasificación: K-NN (K-Nearest-Neighbors).

**PASO 1:** Elegir el número K de vecinos



**PASO 2:** Tomar los K vecinos más cercanos del nuevo dato, según la distancia Euclídea



**PASO 3:** Entre esos K vecinos, contar el número de puntos que pertenecen a cada categoría



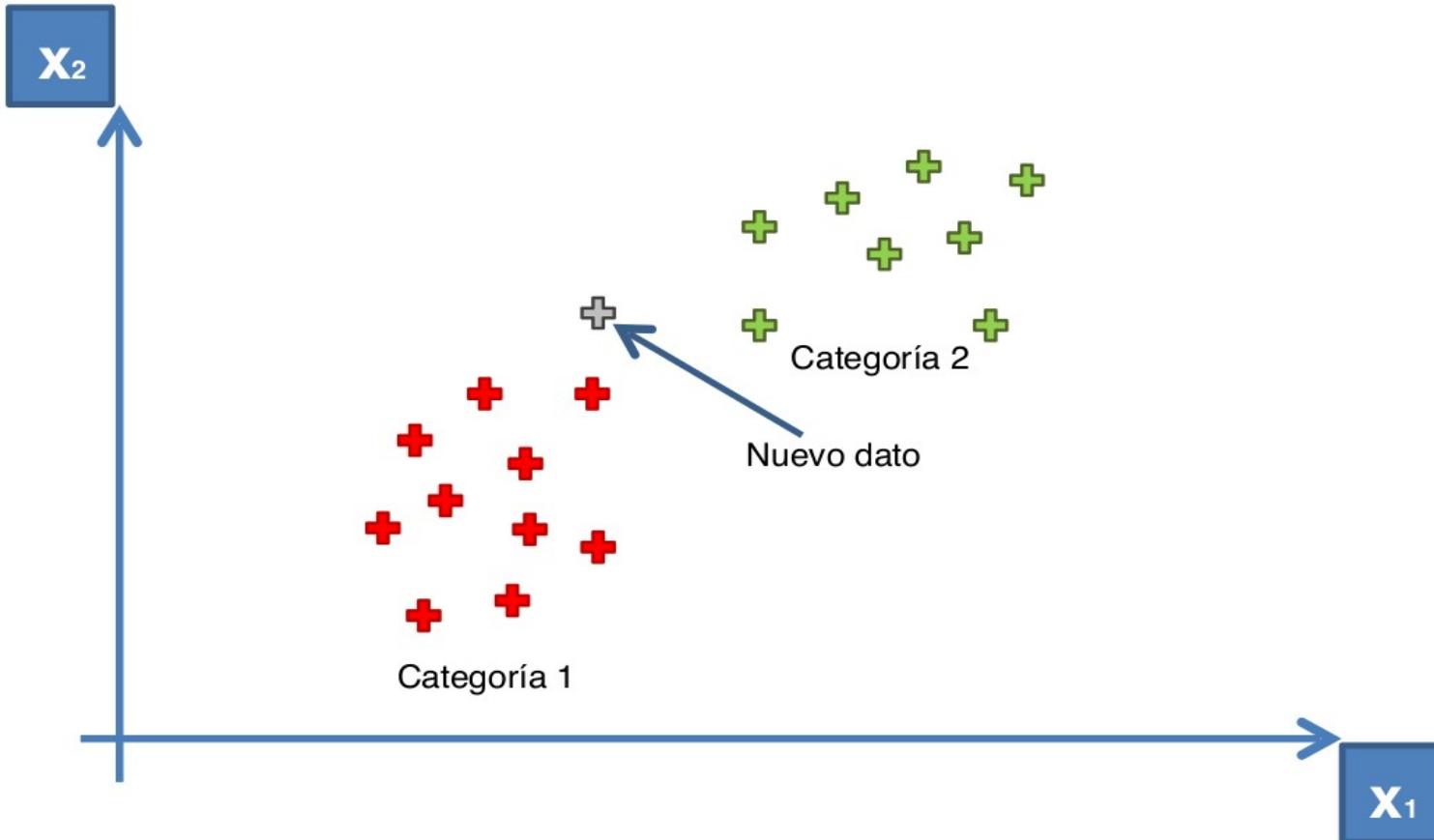
**PASO 4:** Asignar el nuevo dato a la categoría con más vecinos en ella



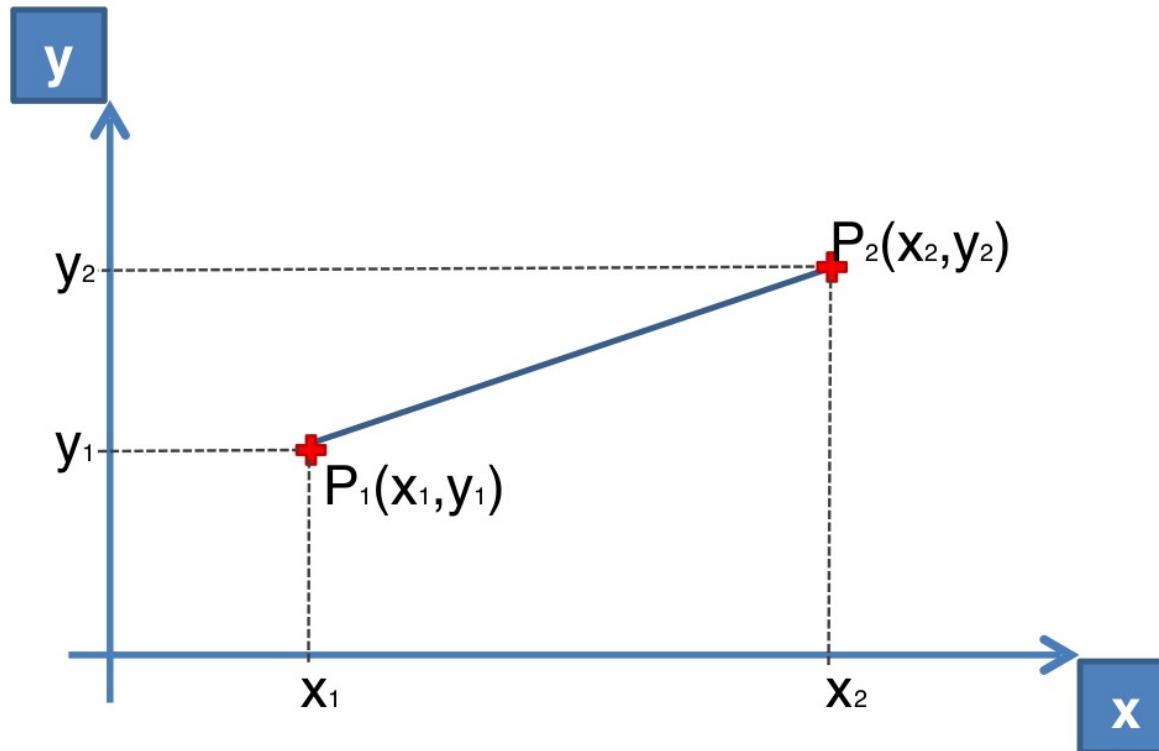
**Nuestro modelo está listo**

# Clasificación: K-NN (K-Nearest-Neighbors).

PASO 1: Elegir el número K de vecinos: K = 5



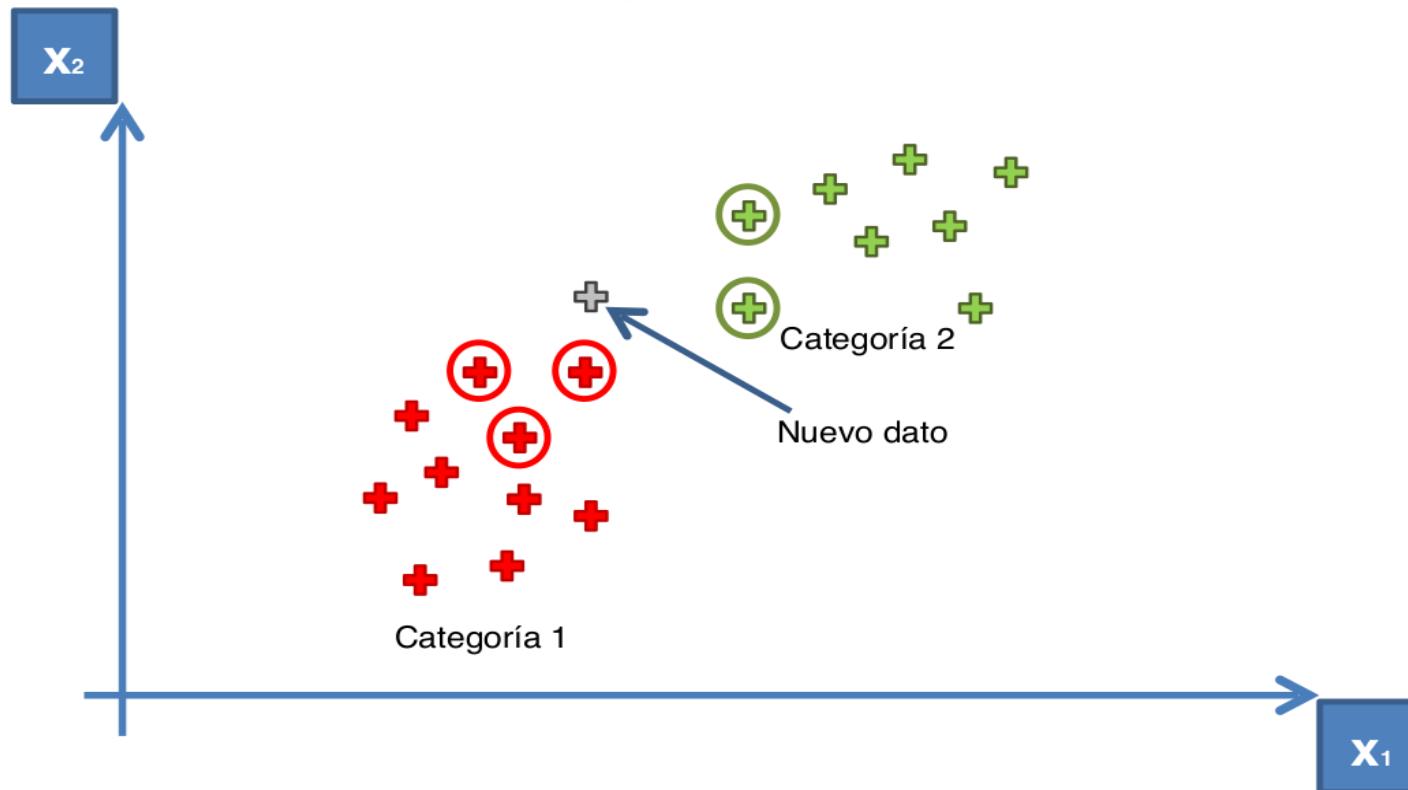
# Clasificación: K-NN (K-Nearest-Neighbors).



$$\text{Distancia Euclídea entre } P_1 \text{ y } P_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

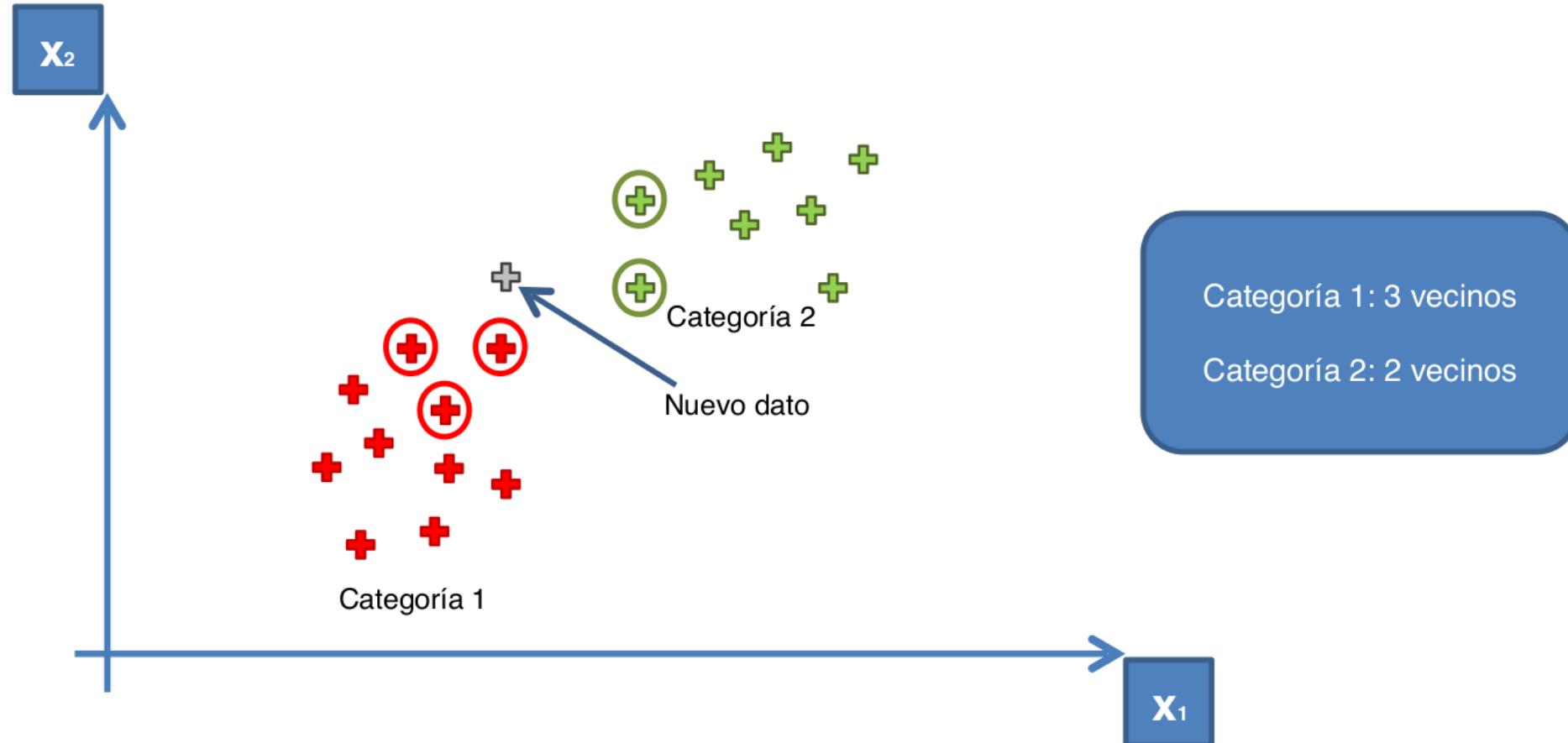
# Clasificación: K-NN (K-Nearest-Neighbors).

**PASO 2:** Tomar los  $K = 5$  vecinos más cercanos del nuevo dato,  
según la distancia Euclidiana



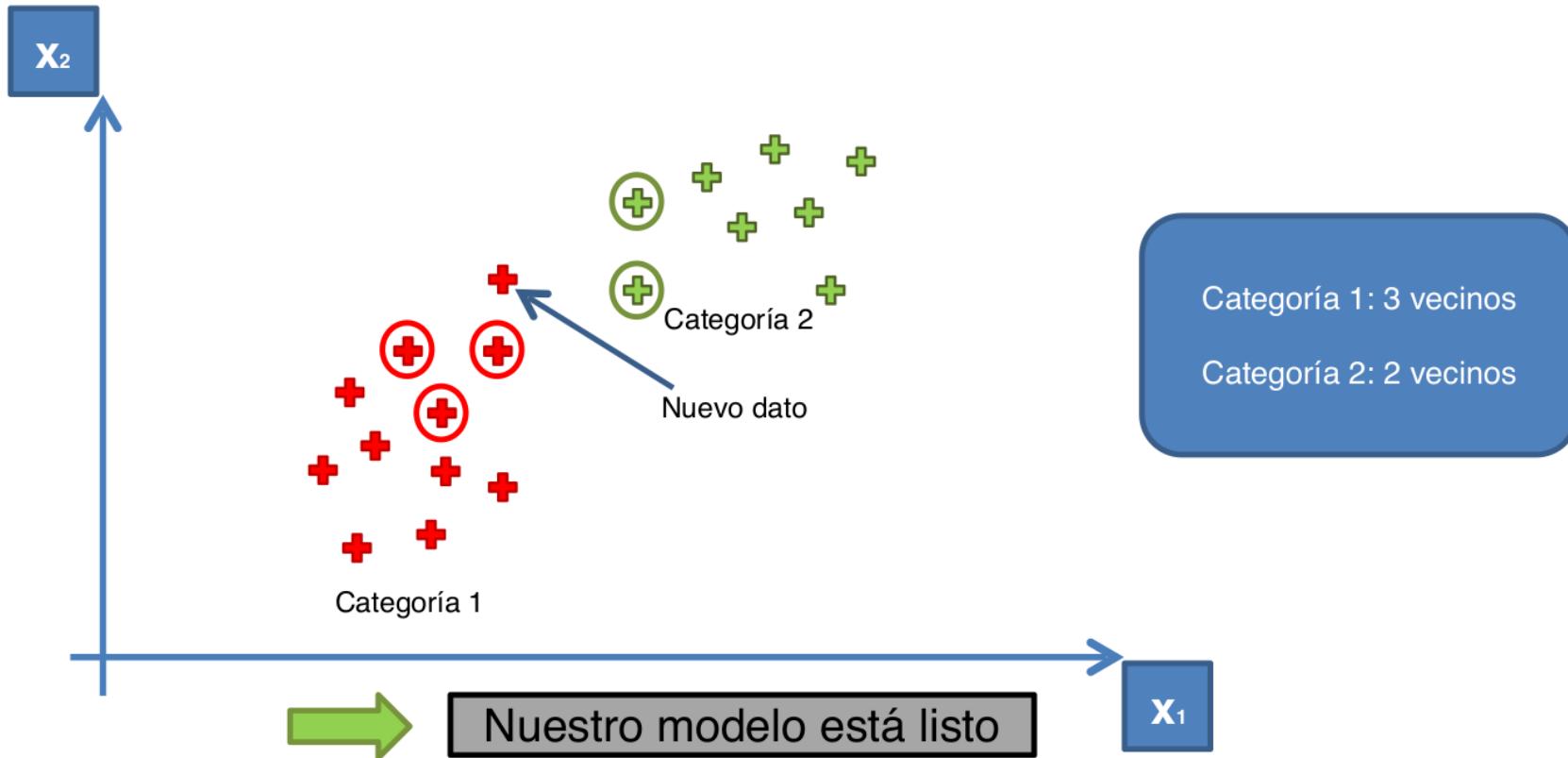
# Clasificación: K-NN (K-Nearest-Neighbors).

PASO 3: De entre esos K vecinos, contar el número de puntos de cada categoría



# Clasificación: K-NN (K-Nearest-Neighbors).

**PASO 4:** Asignar el nuevo dato a la categoría con más vecinos



# Clasificación: K-NN (K-Nearest-Neighbors).

```
# K - Nearest Neighbors (K-NN)
```

```
# Paso 1. Importar el dataset
```

```
dataset = read.csv('Datasets/Social_Network_Ads.csv')  
dataset = dataset[, 3:5]
```

```
# Paso 2. Dividir los datos en conjunto de entrenamiento y conjunto de test
```

```
# install.packages("caTools")  
library(caTools)  
set.seed(123)  
split = sample.split(dataset$Purchased, SplitRatio = 0.75)  
training_set = subset(dataset, split == TRUE)  
testing_set = subset(dataset, split == FALSE)
```

```
# Paso 3. Escalado de valores. Hay que escalar las columnas Age y EstimatedSalary.
```

```
training_set[,1:2] = scale(training_set[,1:2])  
testing_set[,1:2] = scale(testing_set[,1:2])
```

# Clasificación: K-NN (K-Nearest-Neighbors).

#Paso 4. Ajustar el clasificador con el conjunto de entrenamiento y hacer las predicciones con el conjunto de testing.

#En este punto la librería "class" cuenta con la función "knn" que se encarga de recibir el conjunto de training y testing,

#así como predecir automáticamente la categoría en función de la variable que se pretende predecir.

#La función knn recibirá los siguientes parámetros como mínimo:

#"train": Dataset que contiene el conjunto de entrenamiento. Se elimina la columna 3 que es precisamente la que se pretende predecir (Purchased).

#"test": Dataset que contiene el conjunto de testing. En este caso también se elimina la columna 3 ya que es la que se pretende predecir (Purchased).

#"cl": Representa la columna a predecir partiendo del dataset de entrenamiento.

#"k": Número de vecinos cercanos que se utilizarán para realizar la predicción/votación. Se recomienda un valor impar para evitar empates.

#En esencia el KNN sigue la misma mecánica de los modelos vistos anteriormente pero engloba todos los pasos.

#Realiza el entrenamiento y predicción en un único paso (todo se hace internamente en la función).

```
library(class)
y_pred = knn(train = training_set[,-3],
              test = testing_set[,-3],
              cl = training_set[,3],
              k = 5)
```

# Clasificación: K-NN (K-Nearest-Neighbors).

#Paso 5. Crear la matriz de confusión.

#Exactamente igual que lo visto en el modelo de clasificación logístico.

#Se intenta generar una tabla para saber el porcentaje de acierto y error con el dataset de prueba.

```
cm = table(testing_set[, 3], y_pred)
```

#Después de ejecutar la instrucción anterior se puede ver el valor la variable "cm".

```
# > cm  
# y_pred  
#   0   1  
# 0 59  5  
# 1  6 30
```

#En este caso concreto el modelo ha acertado en los valores 0 y 1 en 89 observaciones de 100 (89%) y se ha equivocado en 11.

# Clasificación: K-NN (K-Nearest-Neighbors).

#Paso 6. Visualización del conjunto de training

```
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = knn(train = training_set[,-3],
              test = grid_set,
              cl = training_set[,3],
              k = 5)
plot(set[, -3],
      main = 'K-NN (Conjunto de Entrenamiento)',
      xlab = 'Edad', ylab = 'Sueldo Estimado',
      xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

# Clasificación: K-NN (K-Nearest-Neighbors).

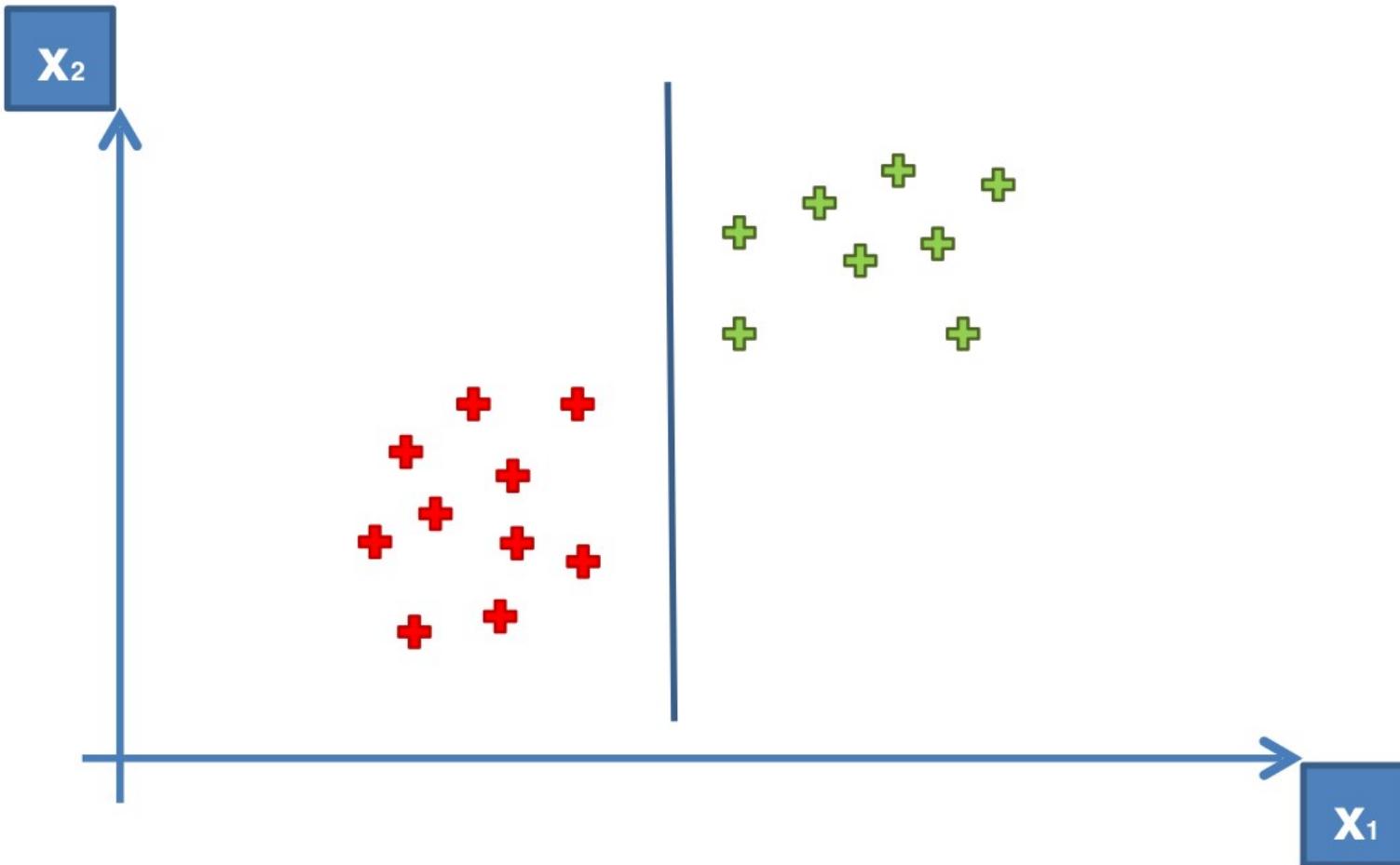
#Paso 6.1 Visualización del conjunto de testing

```
set = testing_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = knn(train = training_set[,-3],
              test = grid_set,
              cl = training_set[,3],
              k = 5)
plot(set[, -3],
      main = 'K-NN (Conjunto de Testing)',
      xlab = 'Edad', ylab = 'Sueldo Estimado',
      xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

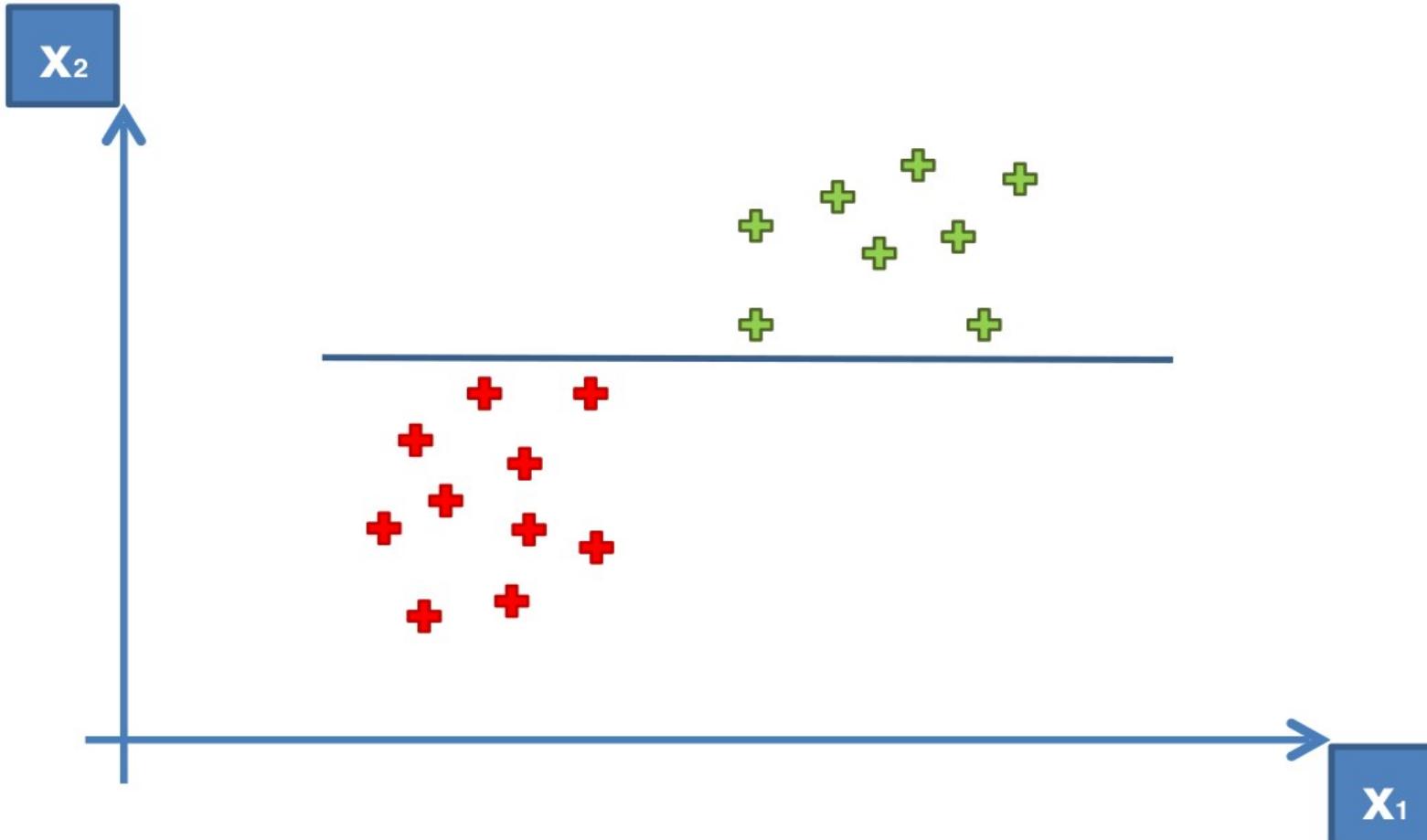
## Clasificación: SVM.

- En las SVM, tal como se ha visto se debe trazar una línea divisoria y definir los vectores de soporte. Además, existirá un margen máximo.
- La forma en la que se traza dicha línea hace parte del algoritmo de SVM y es precisamente lo que permite la clasificación de observaciones con características comunes.
- La clasificación se lleva a cabo partiendo precisamente de dichos vectores de soporte.

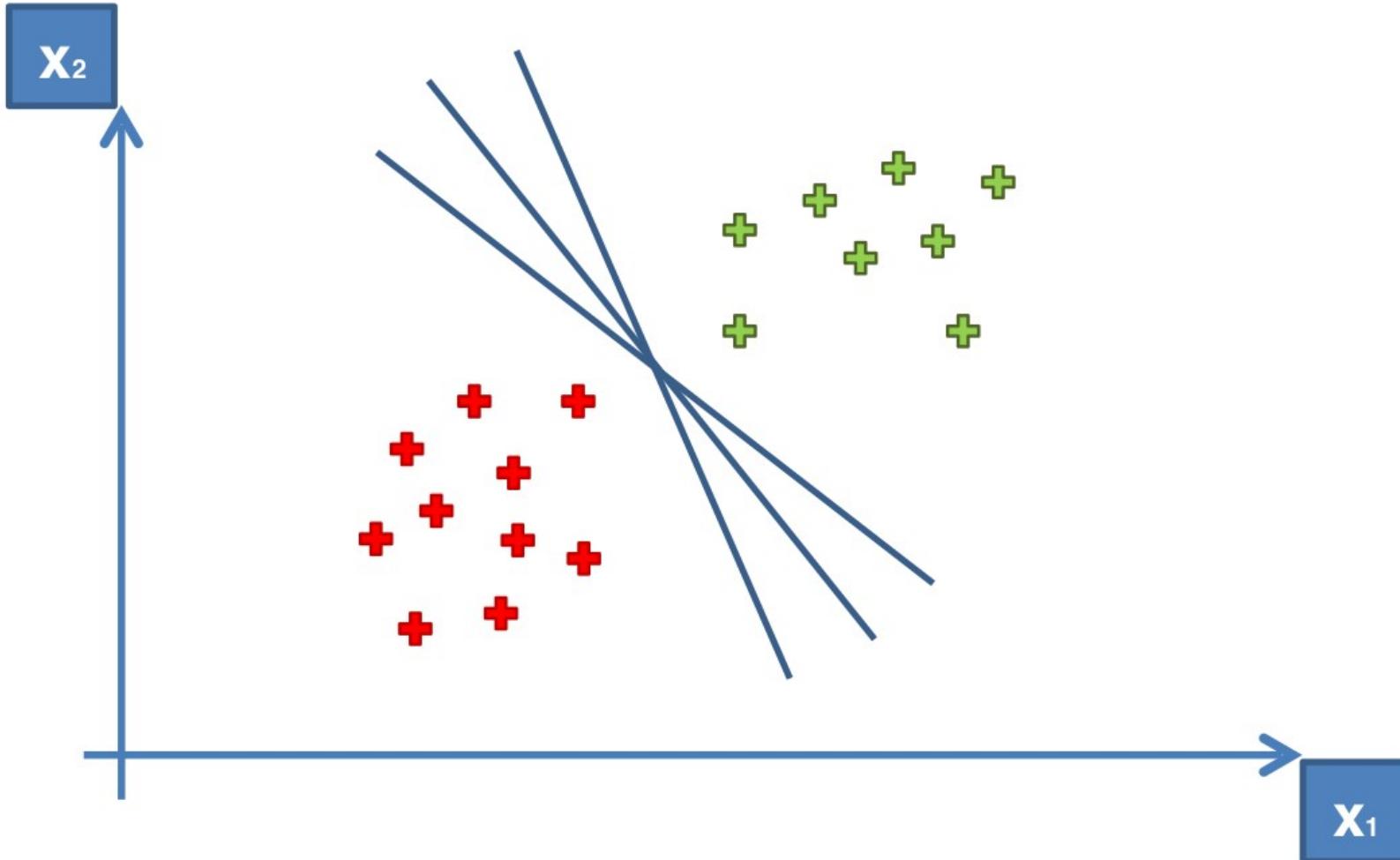
# Clasificación: SVM.



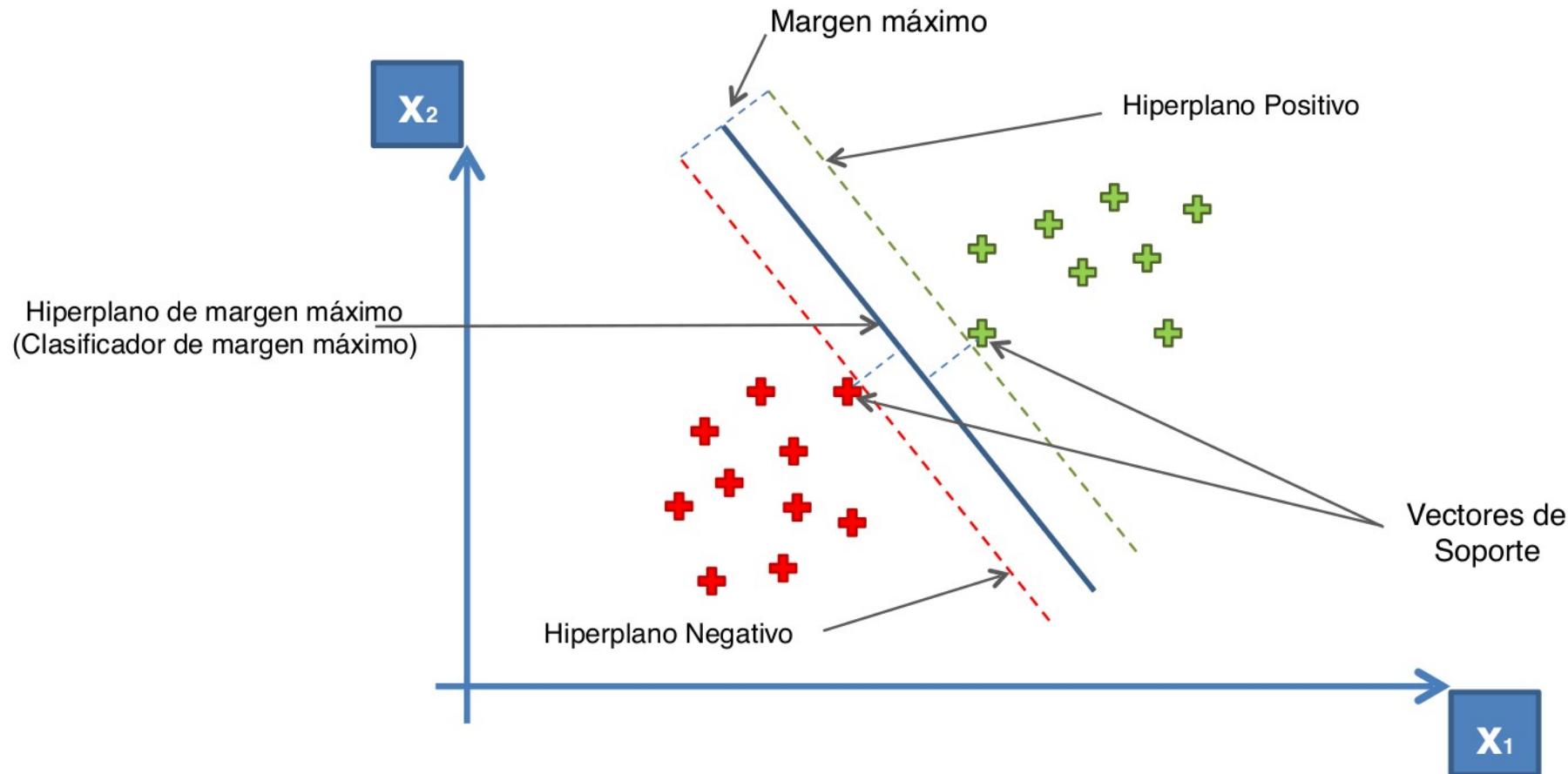
# Clasificación: SVM.



# Clasificación: SVM.



# Clasificación: SVM.



# Clasificación: SVM.

```
# SVM

#Paso 1. Importar el dataset
dataset = read.csv('Datasets/Social_Network_Ads.csv')
dataset = dataset[, 3:5]

#Paso 2. Dividir los datos en conjunto de entrenamiento y conjunto de test
# install.packages("caTools")
library(caTools)
set.seed(123)
split = sample.split(dataset$Purchased, SplitRatio = 0.75)
training_set = subset(dataset, split == TRUE)
testing_set = subset(dataset, split == FALSE)

#Paso 3. Escalado de valores
training_set[,1:2] = scale(training_set[,1:2])
testing_set[,1:2] = scale(testing_set[,1:2])

#Paso 4. Ajustar el SVM con el conjunto de entrenamiento.
#install.packages("e1071")
library(e1071)
classifier = svm(formula = Purchased ~ .,
                 data = training_set,
                 type = "C-classification",
                 kernel = "linear")
```

# Clasificación: SVM.

#Paso 5. Predicción de los resultados con el conjunto de testing

```
y_pred = predict(classifier, newdata = testing_set[,-3])
```

# Crear la matriz de confusión. Ver la tabla generada y determinar cuál ha sido la tasa de acierto

```
cm = table(testing_set[, 3], y_pred)
```

# Visualización del conjunto de entrenamiento

```
#install.packages("ElemStatLearn")
```

```
set = training_set
```

```
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
```

```
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
```

```
grid_set = expand.grid(X1, X2)
```

```
colnames(grid_set) = c('Age', 'EstimatedSalary')
```

```
y_grid = predict(classifier, newdata = grid_set)
```

```
plot(set[, -3],
```

```
     main = 'SVM (Conjunto de Entrenamiento)',
```

```
     xlab = 'Edad', ylab = 'Sueldo Estimado',
```

```
     xlim = range(X1), ylim = range(X2))
```

```
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
```

```
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
```

```
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

# Clasificación: SVM.

```
# Visualización del conjunto de testing
set = testing_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3],
      main = 'SVM (Conjunto de Testing)',
      xlab = 'Edad', ylab = 'Sueldo Estimado',
      xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

## Clasificación: SVM.

- La función “svm” recibe un argumento llamado “kernel”. En el ejemplo anterior se ha especificado el valor “lineal”, sin embargo admite otros valores entre los que se incluyen sigmoideo y polinomial.
- Los resultados del modelo mejoran, empeoran o se mantienen cambiando estos valores? Ver la documentación disponible **?svm**

# Clasificación: Naïve Bayes.

- Se basa en una técnica de clasificación estadística llamada “teorema de Bayes”.
- En ellos se asume que las variables predictoras son independientes entre sí. En otras palabras, que la presencia de una cierta característica en un conjunto de datos no está en absoluto relacionada con la presencia de cualquier otra característica.
- Se basan en el cálculo de la probabilidad ‘futura’ de que ocurra un cierto evento A, dados los eventos anteriores.

# Clasificación: Naïve Bayes.

# Naïve Bayes

#Paso 1. Importar el dataset

```
dataset = read.csv('Datasets/Social_Network_Ads.csv')  
dataset = dataset[, 3:5]
```

#Paso 2. Codificar la variable de clasificación como factor.

#Este paso es importante ya que la función naiveBayes recibe un factor para la variable dependiente.

```
dataset$Purchased = factor(dataset$Purchased, levels = c(0,1))
```

#Paso 3. Dividir los datos en conjunto de entrenamiento y conjunto de test

```
# install.packages("caTools")  
library(caTools)  
set.seed(123)  
split = sample.split(dataset$Purchased, SplitRatio = 0.75)  
training_set = subset(dataset, split == TRUE)  
testing_set = subset(dataset, split == FALSE)
```

#Paso 4. Escalado de valores

```
training_set[,1:2] = scale(training_set[,1:2])  
testing_set[,1:2] = scale(testing_set[,1:2])
```

# Clasificación: Naïve Bayes.

#Paso 5. Ajustar el clasificador con el conjunto de entrenamiento.

```
#install.packages("e1071")
```

#Tal como se ha visto en el paso 2, el parámetro "y" de la función naiveBayes recibe como argumento un factor, #por lo tanto en el dataset lo que se ha hecho es precisamente generar un factor con las posibles categorías para la variable dependiente "Purchased" #que en este caso son simplemente 0 y 1.

```
library(e1071)
```

```
classifier = naiveBayes(x = training_set[,-3],  
                         y = training_set$Purchased)
```

#Paso 6. Predicción de los resultados con el conjunto de testing

```
y_pred = predict(classifier, newdata = testing_set[,-3])
```

#Paso 7. Crear la matriz de confusión

```
cm = table(testing_set[, 3], y_pred)
```

#Ahora al visualizar en la terminal el contenido de la variable cm se puede apreciar lo siguiente:

```
# > cm
```

```
# y_pred
```

```
# 0 1
```

```
# 0 57 7
```

```
# 1 7 29
```

#Lo que indica que 86% de las observaciones han sido correctamente predecidas y 14% no.

# Clasificación: Naïve Bayes.

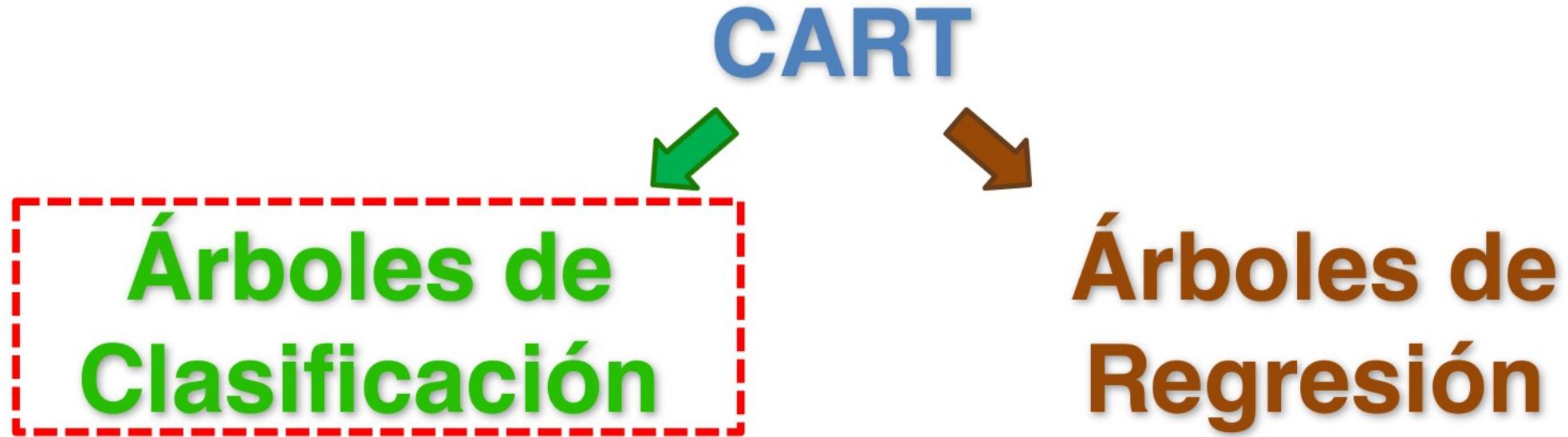
```
#Paso 8. Visualización del conjunto de entrenamiento
#install.packages("ElemStatLearn")
library(ElemStatLearn)
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3],
      main = 'Naïve Bayes (Conjunto de Entrenamiento)',
      xlab = 'Edad', ylab = 'Sueldo Estimado',
      xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

# Clasificación: Naïve Bayes.

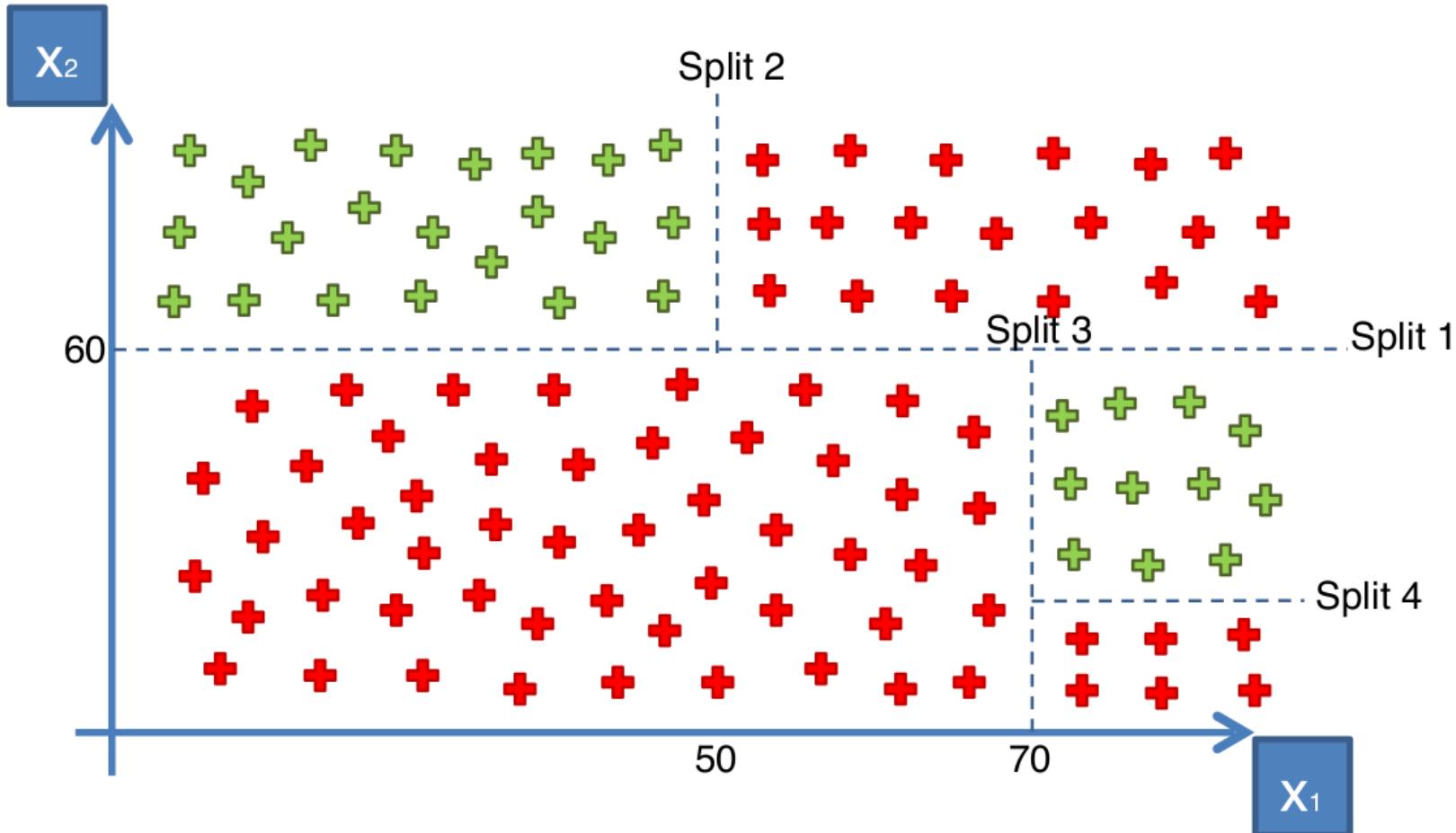
#Paso 8.1 Visualización del conjunto de testing

```
set = testing_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3],
      main = 'Naïve Bayes (Conjunto de Testing)',
      xlab = 'Edad', ylab = 'Sueldo Estimado',
      xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

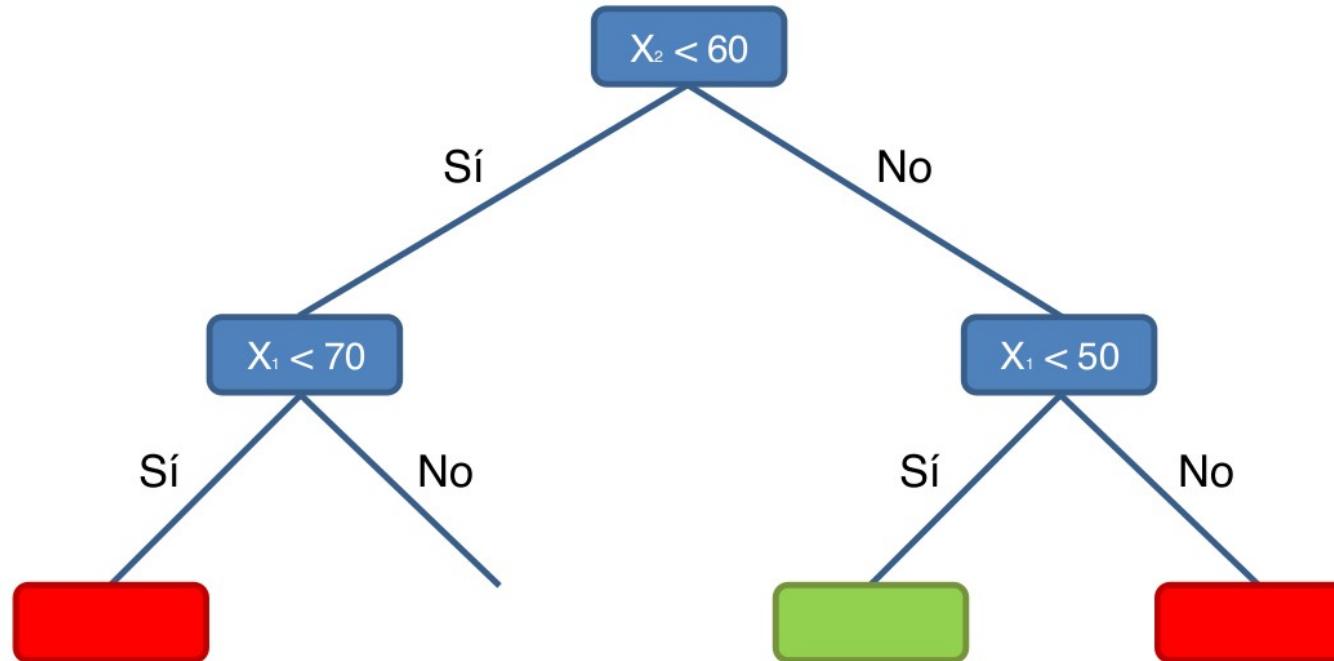
# Clasificación: árboles de decisión.



# Clasificación: árboles de decisión.



# Clasificación: árboles de decisión.



# Clasificación: árboles de decisión.

- **Método antiguo**
- **Se ha mejorado con los años**
  - **Random Forest**
  - **Gradient Boosting**
  - **etc.**

# Clasificación: árboles de decisión.

# Clasificación con Árboles de Decisión

#Paso 1. Importar el dataset

```
dataset = read.csv('Datasets/Social_Network_Ads.csv')  
dataset = dataset[, 3:5]
```

#Paso 2. Codificar la variable de clasificación como factor

```
dataset$Purchased = factor(dataset$Purchased, levels = c(0,1))
```

#Paso 3. Dividir los datos en conjunto de entrenamiento y conjunto de test

```
# install.packages("caTools")  
library(caTools)  
set.seed(123)  
split = sample.split(dataset$Purchased, SplitRatio = 0.75)  
training_set = subset(dataset, split == TRUE)  
testing_set = subset(dataset, split == FALSE)
```

# Clasificación: árboles de decisión.

#Paso 4. Ajustar el clasificador con el conjunto de entrenamiento.

#Se debe utilizar la librería rpart (recursive partitioning and regression tree) tal como se ha visto en los ejemplos de regresión,

# sin embargo en este caso se utilizará para clasificación de datos.

```
#install.packages("rpart")
```

```
library(rpart)
```

```
classifier = rpart(formula = Purchased ~ .,  
                    data = training_set)
```

#Paso 5. Con el clasificador, que representa el entrenamiento que se ha realizado con los datos de training y el modelo seleccionado (rpart)

# se procede a predecir los resultados con el conjunto de testing

# Como resulta evidente, para la predicción se debe eliminar la columna correspondiente a la variable dependiente (Purchased).

```
y_pred = predict(classifier, newdata = testing_set[,-3], type = "class")
```

#NOTA: Ver qué pasa si no se incluye el parámetro type= "class". Este parámetro asigna una clasificación lógica (1 o 0)

```
#y_pred = predict(classifier, newdata = testing_set[, -3])
```

#Resulta conveniente ver la variable "y\_pred" que como se podrá comprobar tiene un valor distinto para cada observación y no hay "0" y "1".

#en su lugar, cada observación enseña la probabilidad de estar en el factor 0 o 1.

```
# > y_pred
```

```
# 0      1  
# 2  0.96703297 0.03296703  
# 4  0.96703297 0.03296703  
# 5  0.96703297 0.03296703
```

# Clasificación: árboles de decisión.

#Paso 6. Crear la matriz de confusión  
cm = table(testing\_set[, 3], y\_pred)  
#> cm  
# y\_pred  
# 0 1  
# 0 56 8  
# 1 6 30

#Paso 7. Visualización del conjunto de entrenamiento  
set = training\_set  
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.05)  
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 250)  
grid\_set = expand.grid(X1, X2)  
colnames(grid\_set) = c('Age', 'EstimatedSalary')  
y\_grid = predict(classifier, newdata = grid\_set, type = "class")  
plot(set[, -3],  
 main = 'Árbol de Decisión (Conjunto de Entrenamiento)',  
 xlab = 'Edad', ylab = 'Sueldo Estimado',  
 xlim = range(X1), ylim = range(X2))  
contour(X1, X2, matrix(as.numeric(y\_grid), length(X1), length(X2)), add = TRUE)  
points(grid\_set, pch = '.', col = ifelse(y\_grid == 1, 'springgreen3', 'tomato'))  
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))

# Clasificación: Random Forest.

- En este caso se seleccionan un número aleatorio determinado de puntos en el dataset seleccionado, que típicamente será el de entrenamiento. Se aplica la técnica de árboles de decisión sobre esos N puntos del dataset y se genera el árbol de decisión sobre cada uno de ellos.
- Dado que la técnica de “Random Forest” se encarga de seleccionar un conjunto determinado de árboles para construir un “bosque”, lo que se ha indicado anteriormente se debe hacer N veces, siendo N el número de árboles que se deben generar sobre los puntos seleccionados.
- A continuación, para medir la eficiencia del modelo, se intenta clasificar cada nuevo punto utilizando el dataset de testing y cada uno de los arboles del bosque debe “votar” sobre si ese nuevo punto pertenece a una categoría u otra.

**EN RESUMEN, LOS PASOS SON:**

# Clasificación: Random Forest.

**PASO 1:** Seleccionar un número aleatorio K de puntos del Conjunto de Entrenamiento.



**PASO 2:** Construir el Árbol de Decisión asociado a esos K puntos de datos.



**PASO 3:** Elegir un número Ntree de árboles que queremos construir y repetir los PASOS 1 y 2.



**PASO 4:** Para clasificar un nuevo punto, hacer que cada uno de los Ntree árboles elabore la predicción de a qué categoría pertenece y asignar el nuevo punto a la categoría con más votos.

# Clasificación: Random Forest.

# Clasificación con Random Forest

#Paso 1. Importar el dataset

```
dataset = read.csv('Datasets/Social_Network_Ads.csv')  
dataset = dataset[, 3:5]
```

#Paso 2. Codificar la variable de clasificación como factor

```
dataset$Purchased = factor(dataset$Purchased, levels = c(0,1))
```

#Paso 3. Dividir los datos en conjunto de entrenamiento y conjunto de test

```
# install.packages("caTools")  
library(caTools)  
set.seed(123)  
split = sample.split(dataset$Purchased, SplitRatio = 0.75)  
training_set = subset(dataset, split == TRUE)  
testing_set = subset(dataset, split == FALSE)
```

#Paso 4. Escalado de valores

```
training_set[,1:2] = scale(training_set[,1:2])  
testing_set[,1:2] = scale(testing_set[,1:2])
```

# Clasificación: Random Forest.

#Paso 5. Ajustar el Random Forest con el conjunto de entrenamiento.

#Para aplicar el modelo de random forest basta con instalar e importar el modulo "randomForest"  
#y utilizar la función "randomForest".

#La función requiere que se envíen los parámetros X e Y que representan las variables independientes  
#y la variable dependiente respectivamente.

#Además, se envía el parámetro "ntree" que permite indicar cuántos árboles de clasificación  
#se deben generar para el modelo.

```
#install.packages("randomForest")
```

```
library(randomForest)
```

```
classifier = randomForest(x = training_set[,-3],  
                           y = training_set$Purchased,  
                           ntree = 10)
```

#Paso 6. Predicción de los resultados con el conjunto de testing

```
y_pred = predict(classifier, newdata = testing_set[,-3])
```

#Paso 7. Crear la matriz de confusión

```
cm = table(testing_set[, 3], y_pred)
```

# Clasificación: Random Forest.

```
#Paso 8. Visualización del conjunto de entrenamiento
#install.packages("ElemStatLearn")
library(ElemStatLearn)
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3],
      main = 'Random Forest (Conjunto de Entrenamiento)',
      xlab = 'Edad', ylab = 'Sueldo Estimado',
      xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

# Clasificación: Random Forest.

#Paso 8.1 Visualización del conjunto de testing

```
set = testing_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3],
      main = 'Random Forest (Conjunto de Testing)',
      xlab = 'Edad', ylab = 'Sueldo Estimado',
      xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

# **FAQs sobre modelos de clasificación.**

**¿Cuáles son los pros y los contras de cada modelo?**

**¿Cómo sé qué modelo debo elegir para resolver mi problema?**

# ¿Cuáles son los pros y contras de cada modelo?

Classification Model	Pros	Cons
Logistic Regression	Probabilistic approach, gives informations about statistical significance of features	The Logistic Regression Assumptions
K-NN	Simple to understand, fast and efficient	Need to choose the number of neighbours k
SVM	Performant, not biased by outliers, not sensitive to overfitting	Not appropriate for non linear problems, not the best choice for large number of features
Kernel SVM	High performance on nonlinear problems, not biased by outliers, not sensitive to overfitting	Not the best choice for large number of features, more complex
Naive Bayes	Efficient, not biased by outliers, works on nonlinear problems, probabilistic approach	Based on the assumption that features have same statistical relevance
Decision Tree Classification	Interpretability, no need for feature scaling, works on both linear / nonlinear problems	Poor results on too small datasets, overfitting can easily occur
Random Forest Classification	Powerful and accurate, good performance on many problems, including non linear	No interpretability, overfitting can easily occur, need to choose the number of trees

## ¿Cómo puedo mejorar cada uno de estos modelos ?

Se deben ajustar los parámetros que permiten mejorar la eficacia de los modelos. Existen dos tipos de parámetros:

- **parámetros normales:** Son aquellos en los que el modelo aprende, como los coeficientes de la regresión lineal.
- **los hiperparámetros del algoritmo:** Son parámetros en los que el algoritmo no aprende, si no que son fijos y forman parte de las ecuaciones de los modelos. Encontrar el valor óptimo es parte del ajuste de parámetros.

# ¿Cómo sé qué modelo debo elegir para resolver mi problema?

Primero hay que averiguar si el problema es o no es lineal.

**Si es lineal**, intentar un modelo de Regresión Logística o bien SVM.

**Si no es lineal**, K-NN, Naïve Bayes, Árboles o Random Forest.

Desde un punto de vista empresarial:

Regresión Logística o Naïve Bayes para ordenar las predicciones por probabilidad. Por ejemplo, para crear un ranking de clientes desde el más probable al menos probable que compre un producto.

# ¿Cómo sé qué modelo debo elegir para resolver mi problema?

Si el problema de empresa es lineal, mejor utiliza la regresión logística, y si no lo es, intentar con Naïve Bayes.

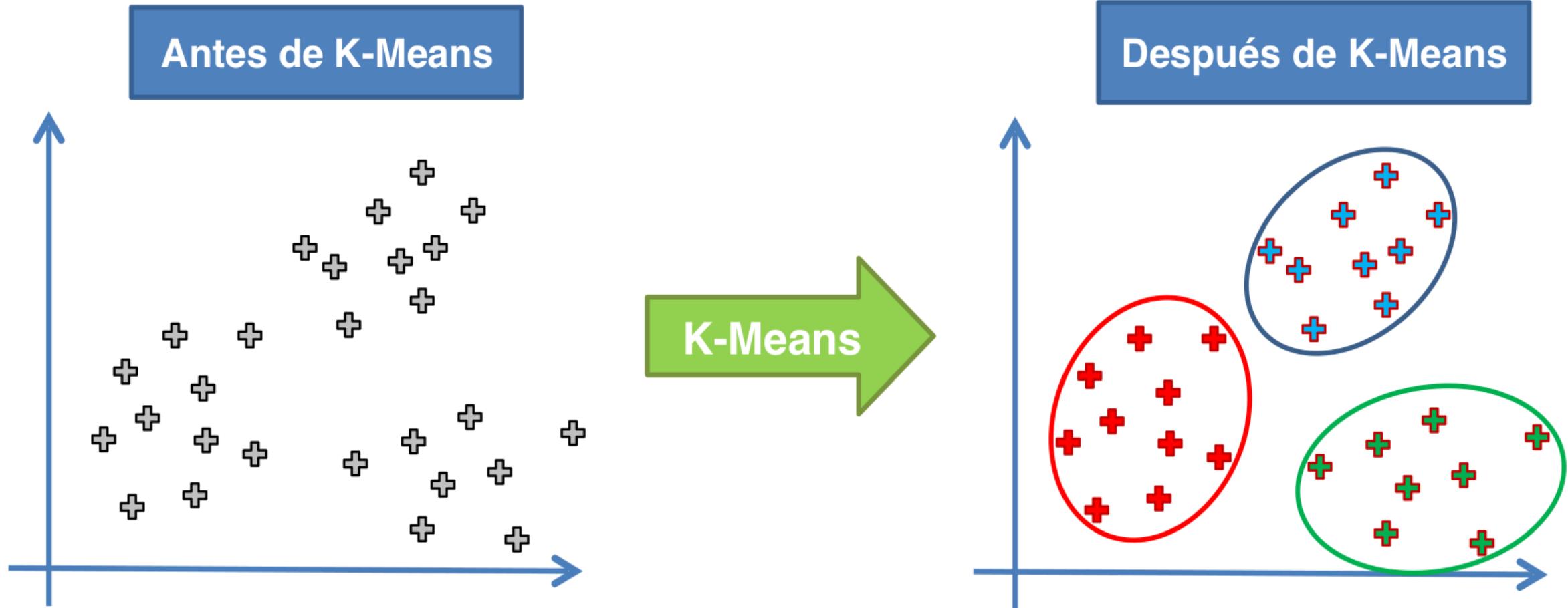
SVM para predecir a qué segmento pertenece un cliente. Los segmentos pueden ser cualquier conjunto de características que definan a los clientes, como los que se identifican en clustering.

Finalmente, Árboles o Random Forest cuando se busque un mejor resultado de predicción y preocupe menos la interpretación de los modelos.

# Modelos de ML basados en Clustering

- Clustering es un proceso similar al de clasificación, pero con un fundamento diferente. En el Clustering no se conocen las categorías previamente, si no que se intenta crear una segmentación de tus propios datos en grupos más o menos homogéneos, es decir, se generan patrones.
- Los algoritmos de clustering pueden producir segmentos inesperados como estructuras o agrupaciones que un humano no se habría imaginado pero el modelo es capaz de encontrarlos.
- En esta formación se implementarán los siguientes algoritmos de Clustering para Machine Learning: **K-Means** y **Jerárquico**

# Modelos de ML basados en Clustering:k-means



# Modelos de ML basados en Clustering:k-means

**PASO 1:** Elegir el número K de clusters



**PASO 2:** Seleccionar al azar K puntos, los baricentros (no necesariamente de nuestro dataset)



**PASO 3:** Asignar cada punto al baricentro más cercano

→ Esto formará los K clusters



**PASO 4:** Calcular y asignar el nuevo baricentro de cada cluster



**PASO 5:** Reasignar cada punto de los datos a su baricentro más cercano.

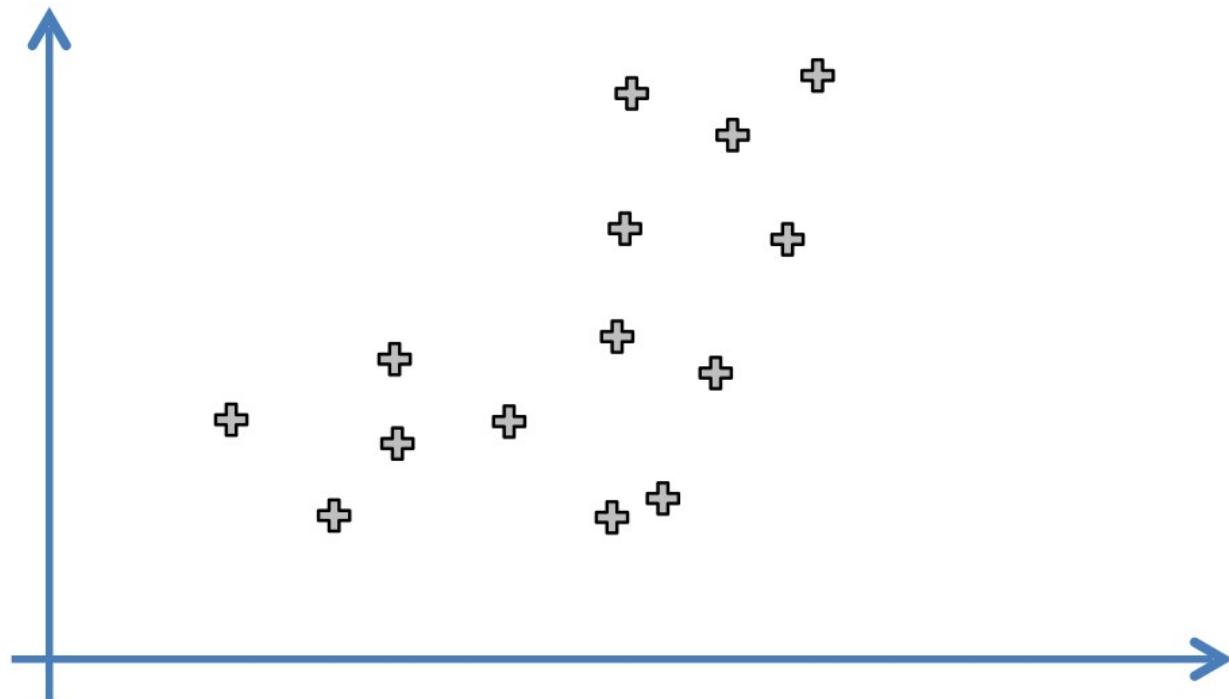
Si ha habido nuevas asignaciones, ir al PASO 4, si no ir FIN.



**El Modelo está Listo**

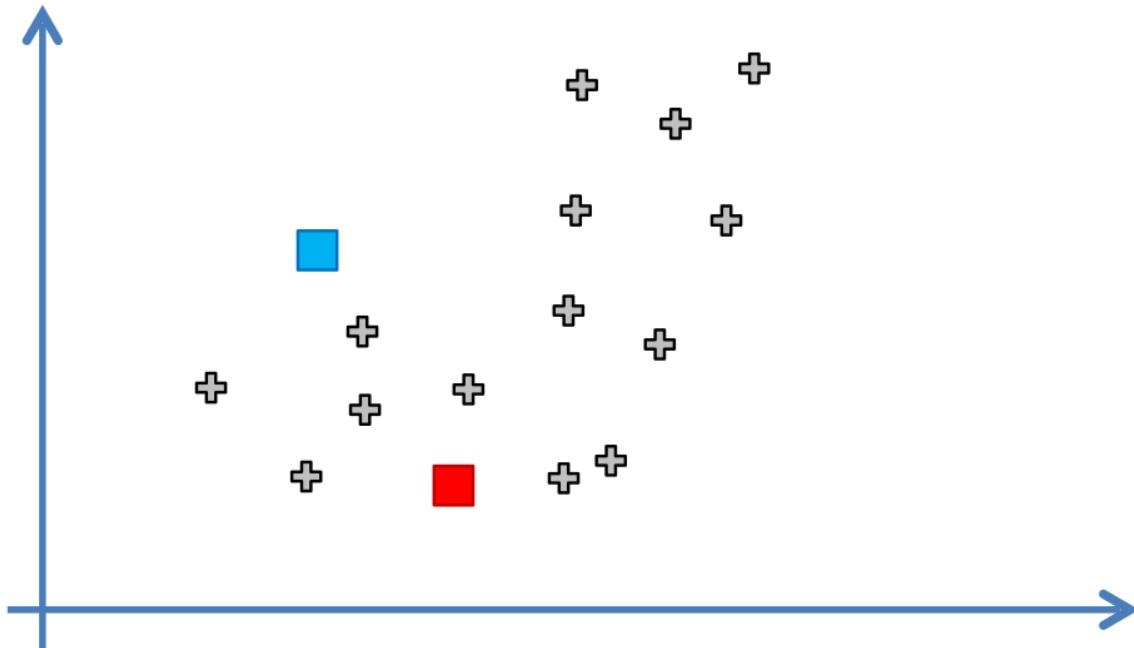
# Modelos de ML basados en Clustering:k-means

PASO 1: Elegir el número K de clusters: K = 2



# Modelos de ML basados en Clustering:k-means

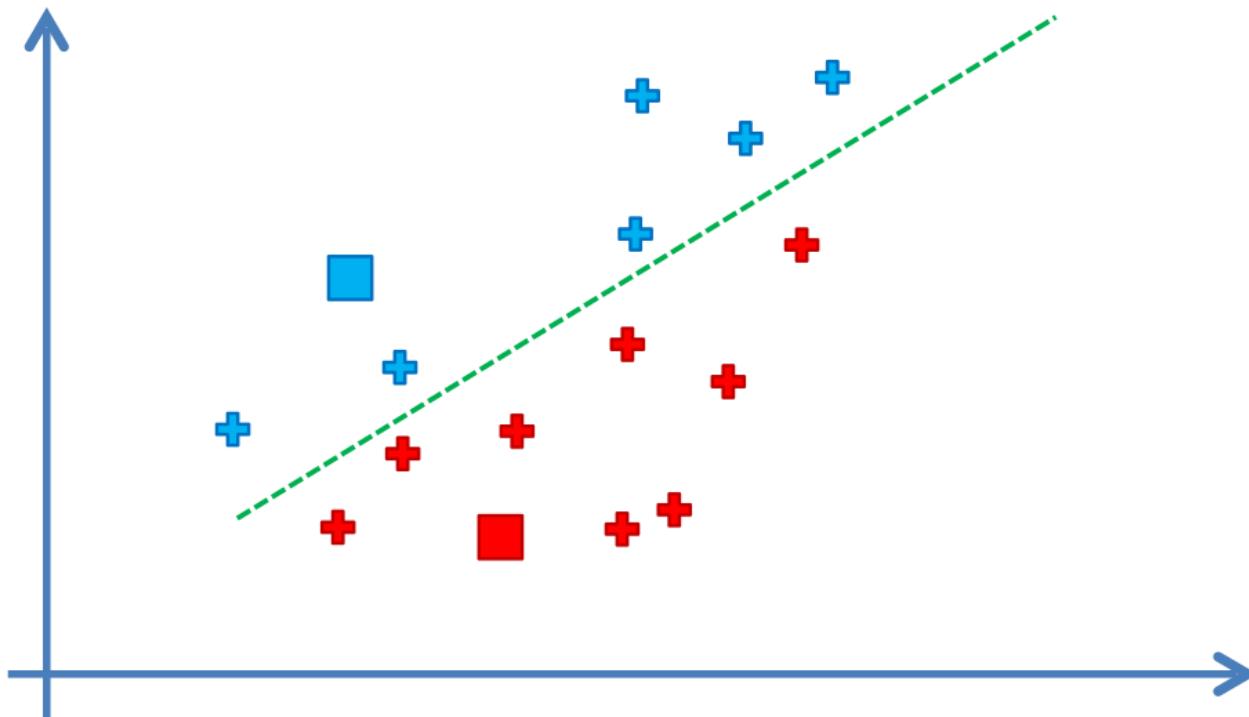
**PASO 2:** Seleccionar al azar K puntos, los baricentros (no necesariamente de nuestro dataset)



# Modelos de ML basados en Clustering:k-means

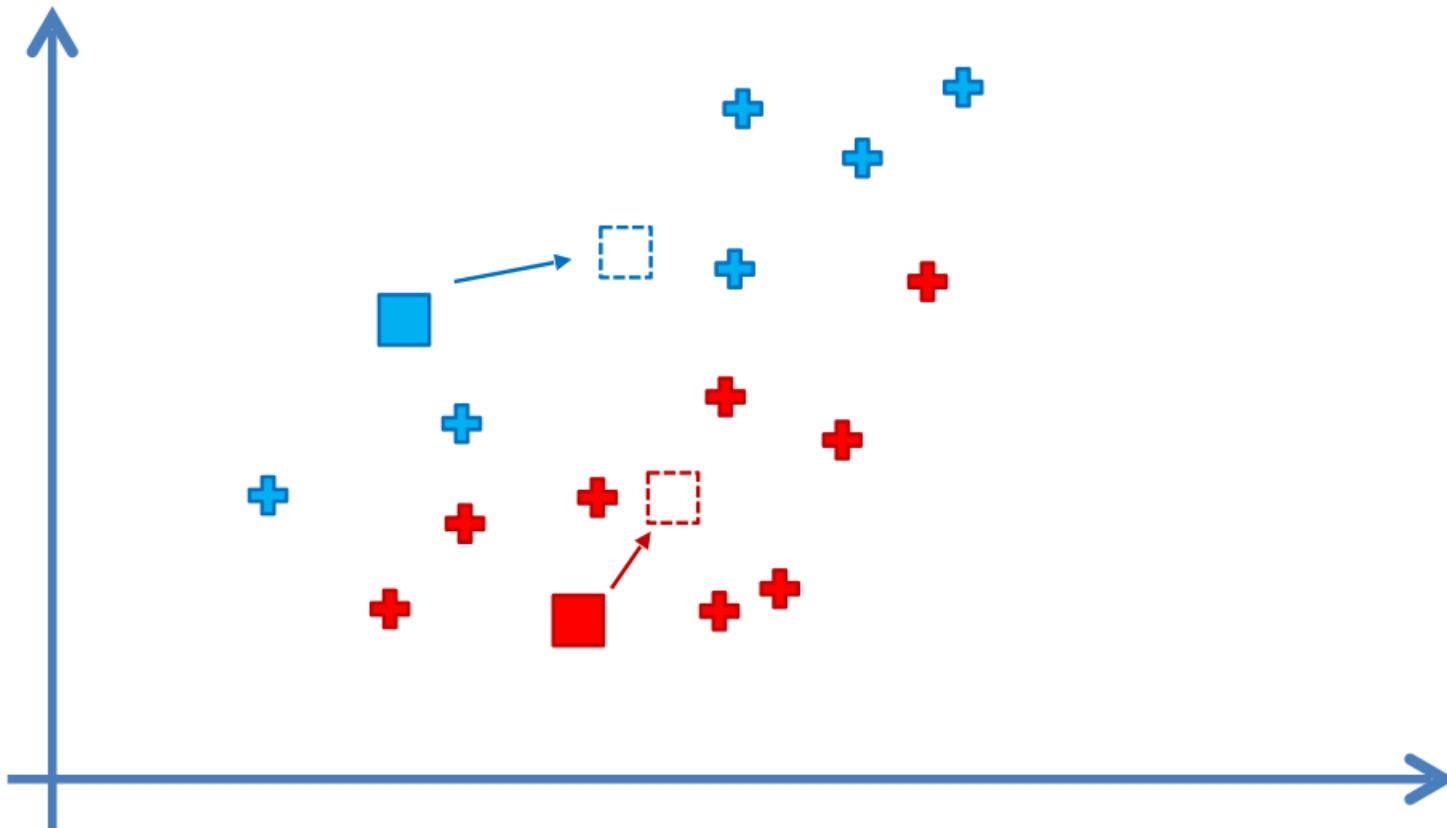
PASO 3: Asignar cada punto al baricentro más cercano

→ Esto formará los K clusters



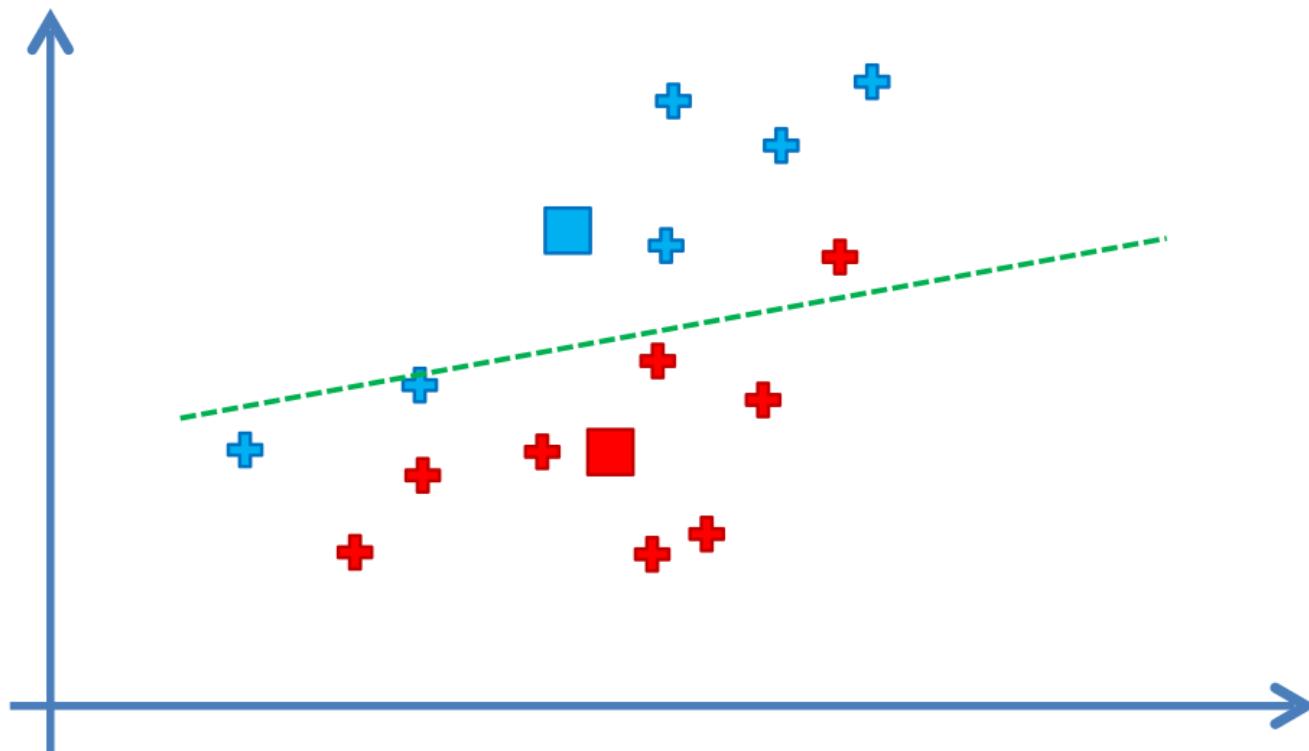
# Modelos de ML basados en Clustering:k-means

**PASO 4:** Calcular y asignar el nuevo baricentro de cada cluster



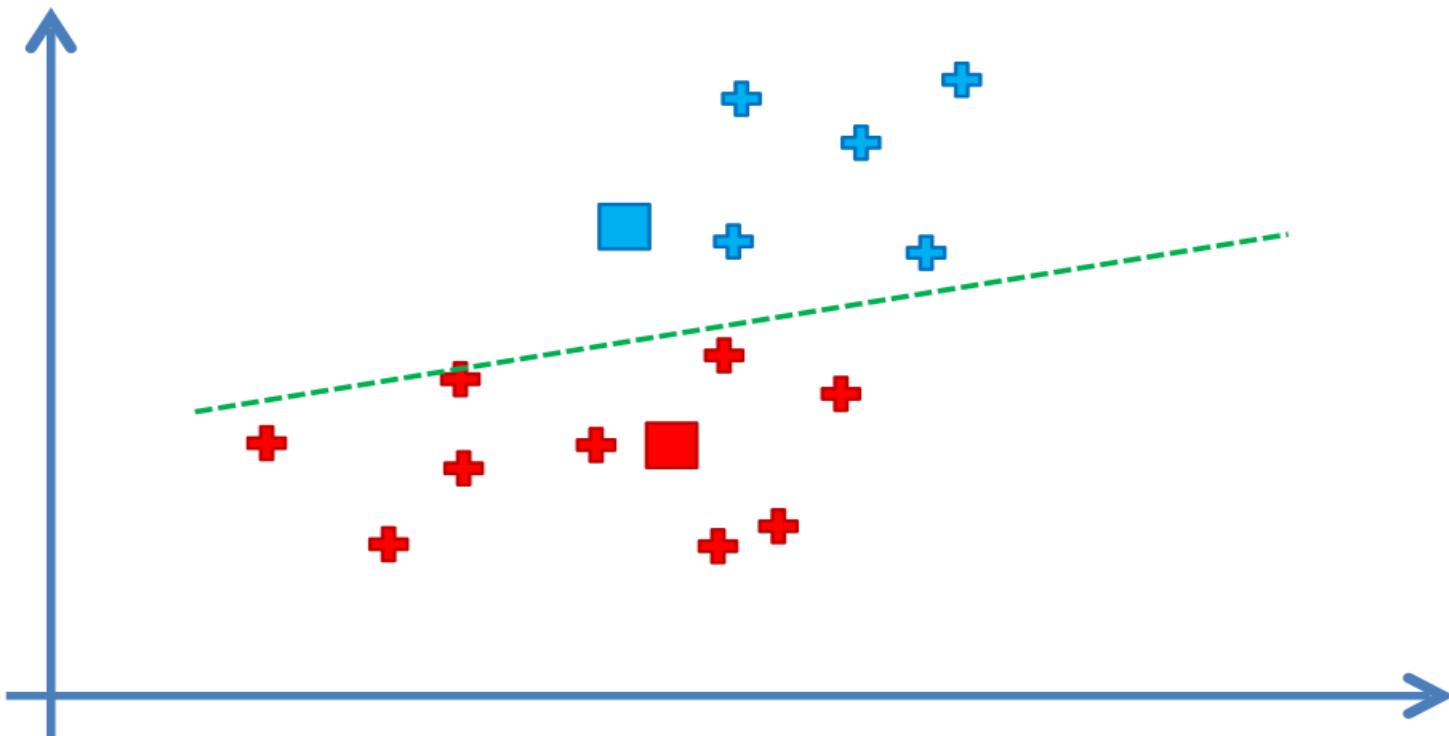
# Modelos de ML basados en Clustering:k-means

**PASO 5:** Reasignar cada punto de los datos a su baricentro más cercano.  
Si ha habido nuevas asignaciones, ir al PASO 4, si no ir FIN.



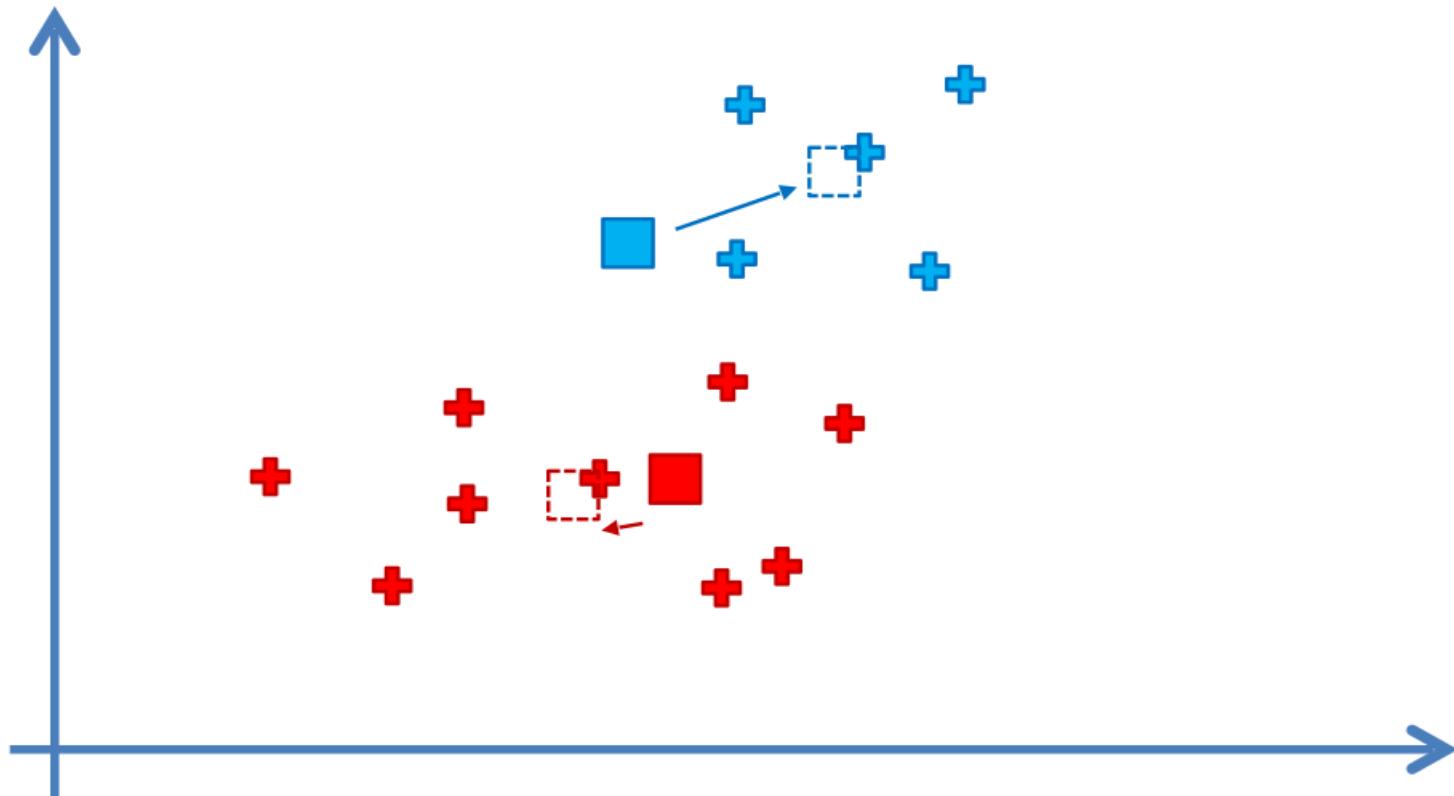
# Modelos de ML basados en Clustering:k-means

**PASO 5:** Reasignar cada punto de los datos a su baricentro más cercano.  
Si ha habido nuevas asignaciones, ir al PASO 4, si no ir FIN.



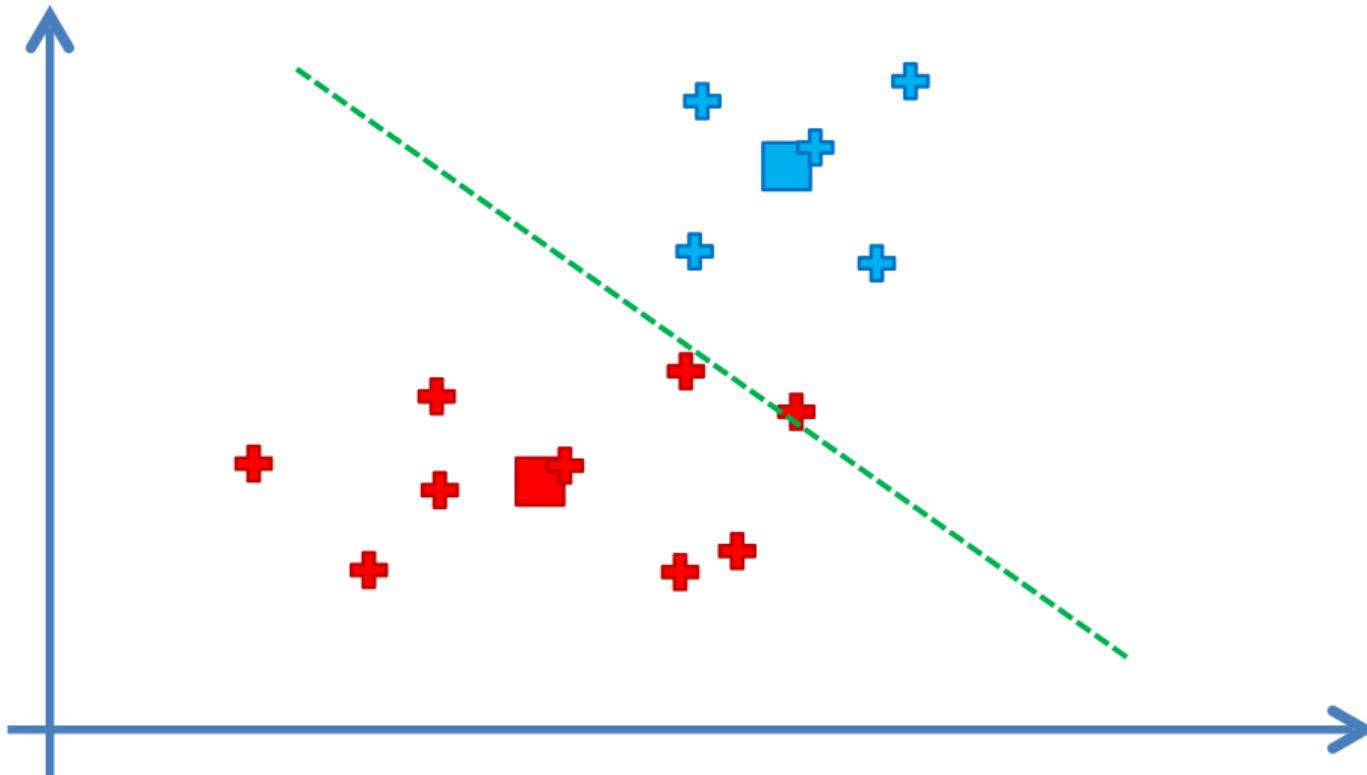
# Modelos de ML basados en Clustering:k-means

**PASO 4:** Calcular y asignar el nuevo baricentro de cada cluster



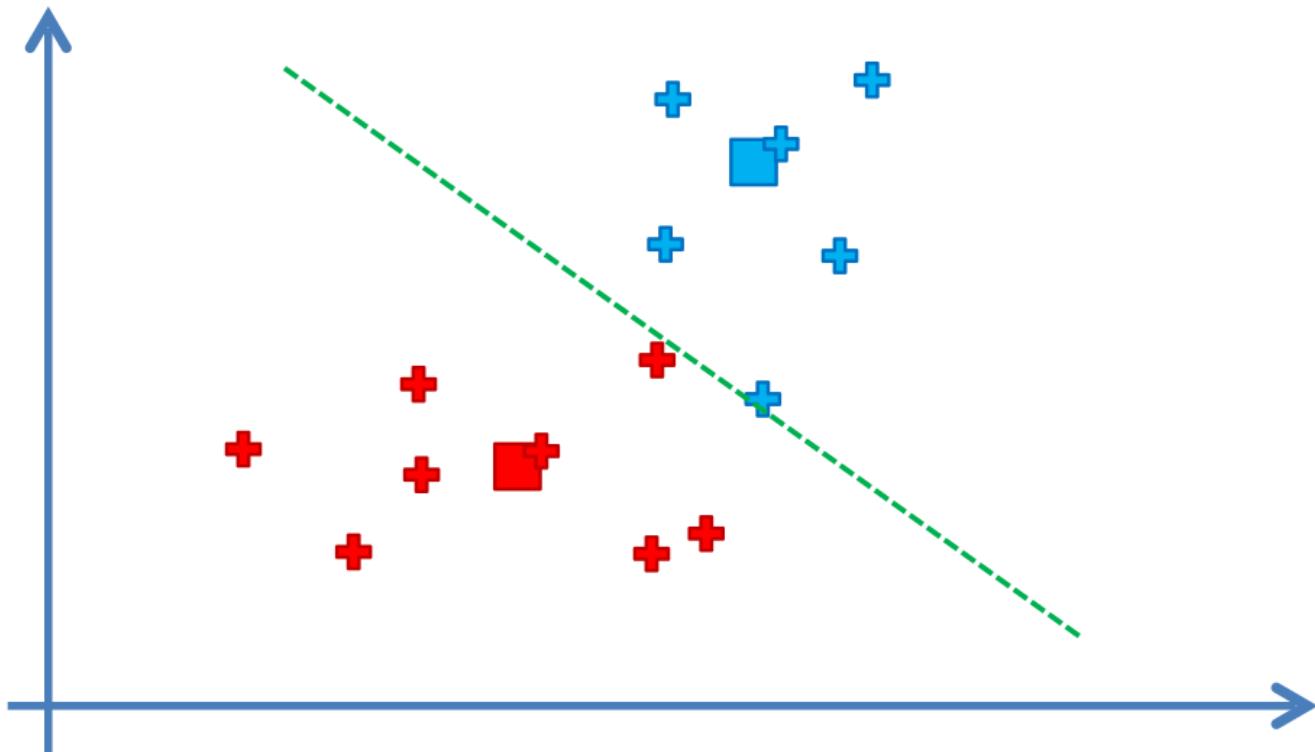
# Modelos de ML basados en Clustering:k-means

**PASO 5:** Reasignar cada punto de los datos a su baricentro más cercano.  
Si ha habido nuevas asignaciones, ir al PASO 4, si no ir FIN.



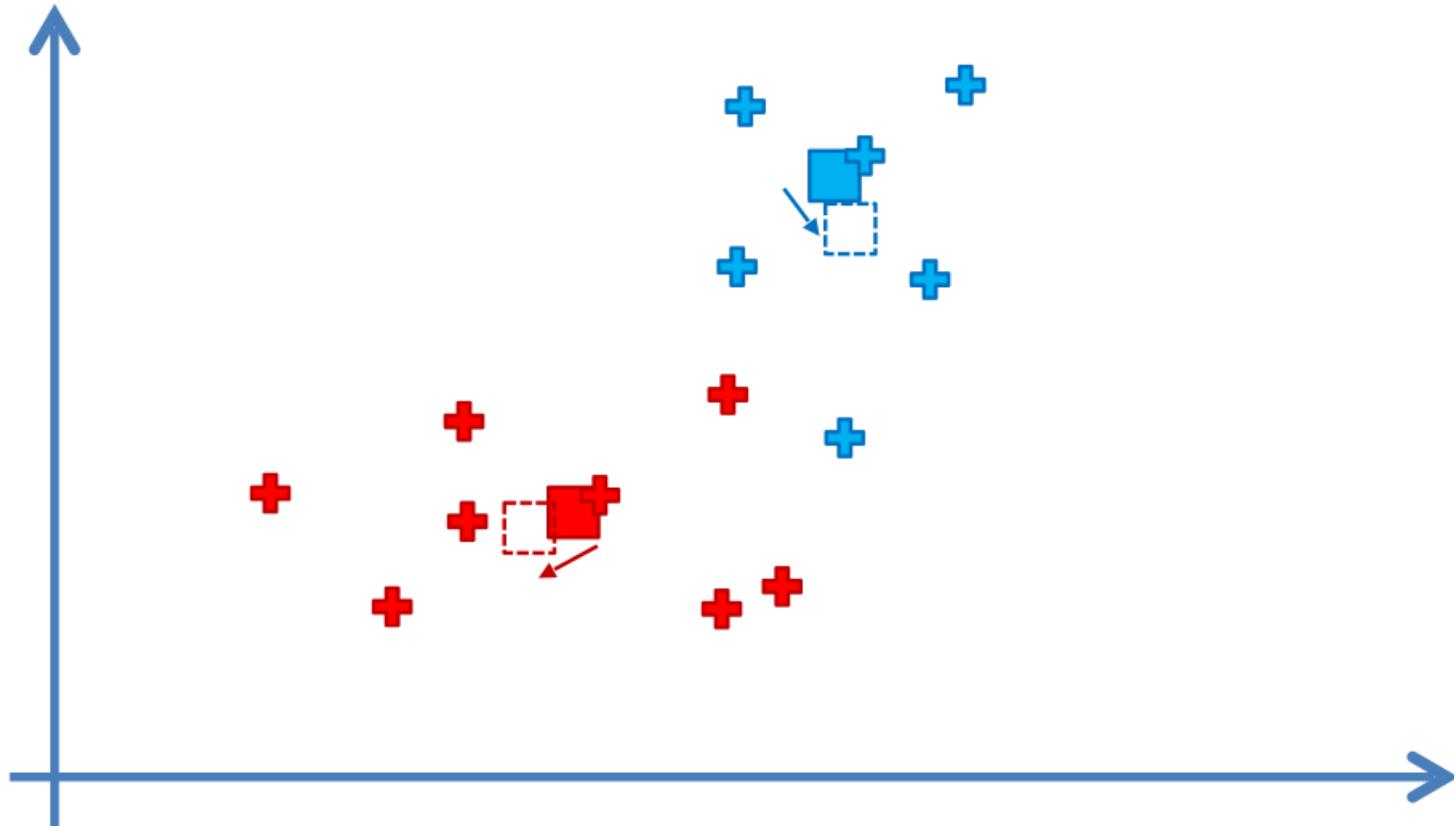
# Modelos de ML basados en Clustering:k-means

**PASO 5:** Reasignar cada punto de los datos a su baricentro más cercano.  
Si ha habido nuevas asignaciones, ir al PASO 4, si no ir FIN.



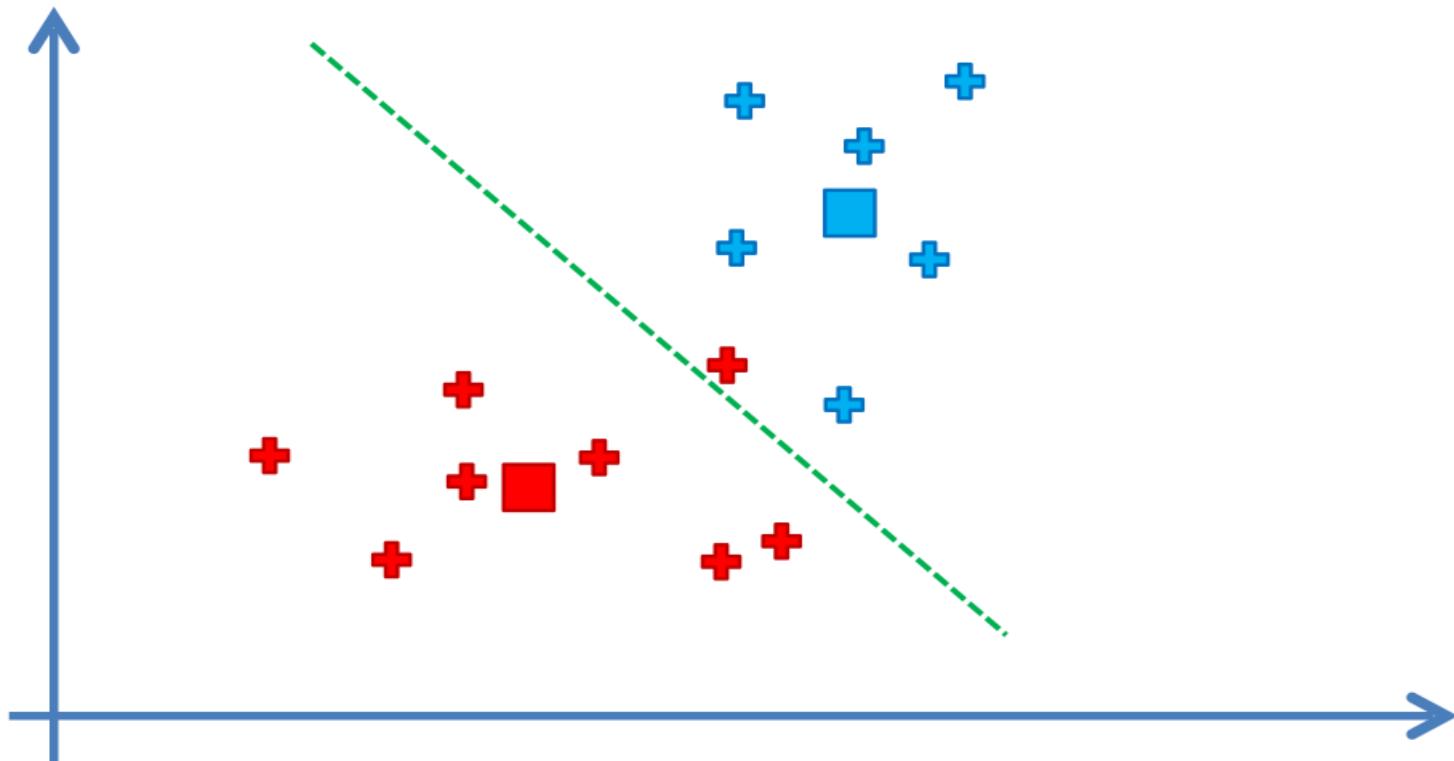
# Modelos de ML basados en Clustering:k-means

**PASO 4:** Calcular y asignar el nuevo baricentro de cada cluster



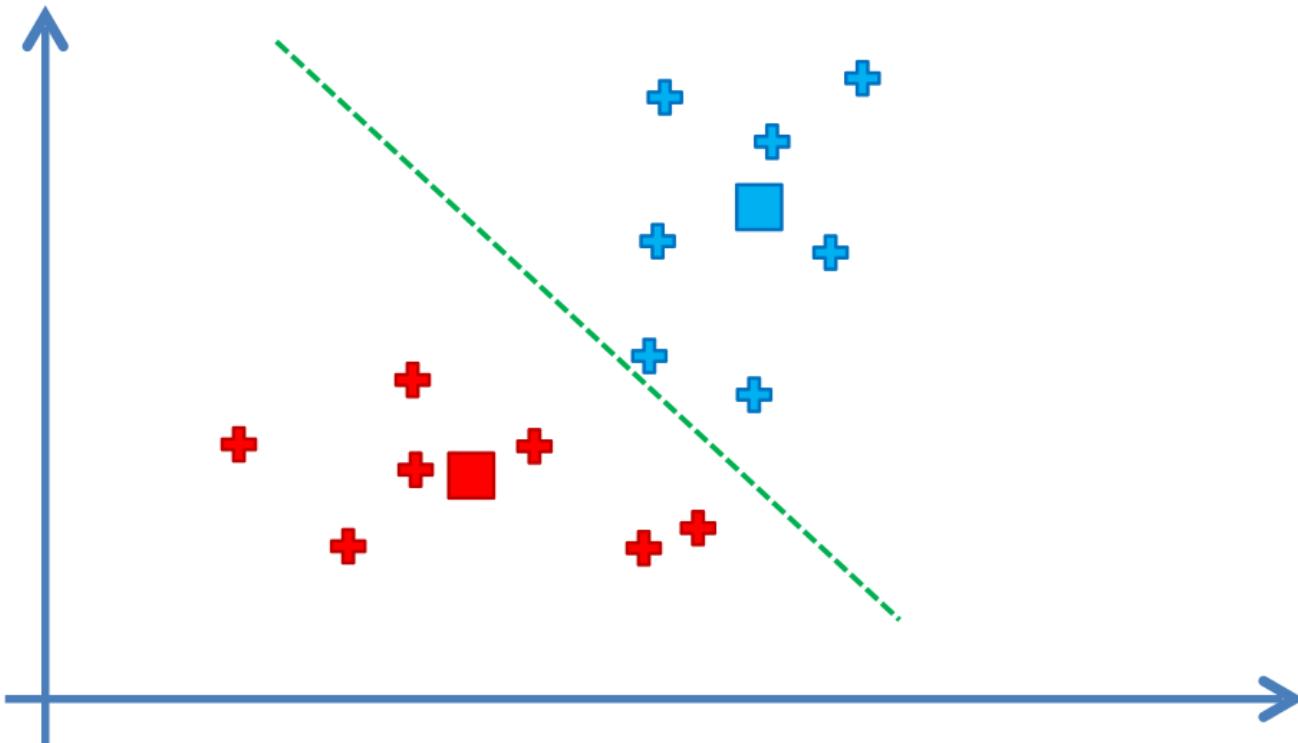
# Modelos de ML basados en Clustering:k-means

**PASO 5:** Reasignar cada punto de los datos a su baricentro más cercano.  
Si ha habido nuevas asignaciones, ir al PASO 4, si no ir FIN.



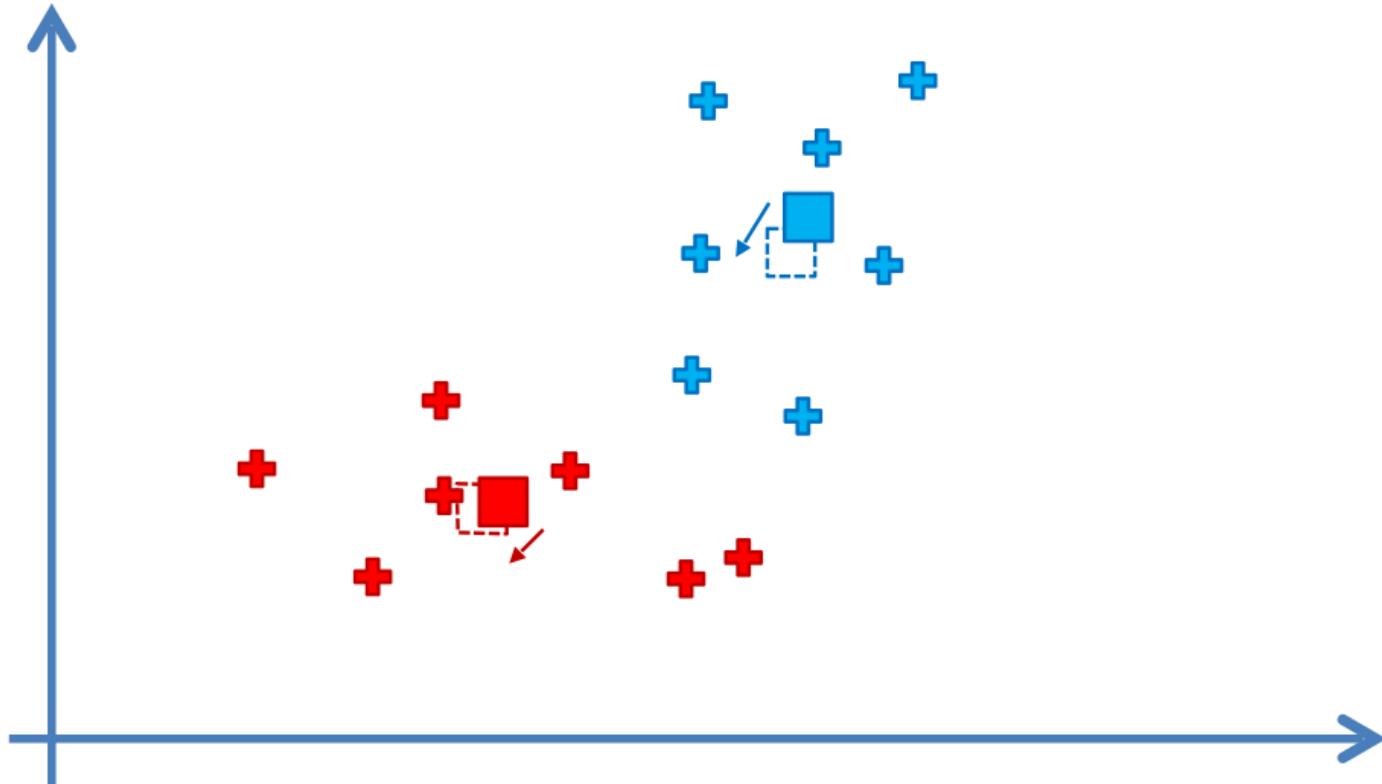
# Modelos de ML basados en Clustering:k-means

**PASO 5:** Reasignar cada punto de los datos a su baricentro más cercano.  
Si ha habido nuevas asignaciones, ir al PASO 4, si no ir FIN.



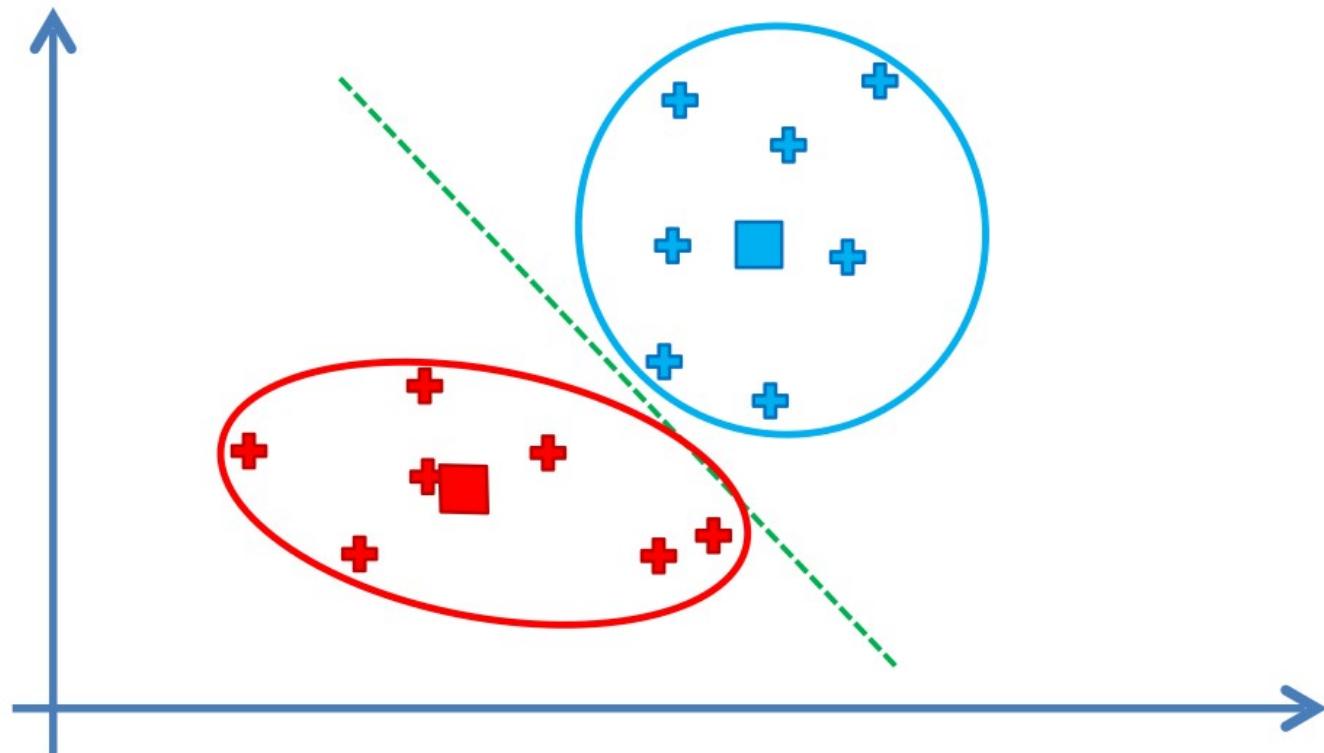
# Modelos de ML basados en Clustering:k-means

**PASO 4:** Calcular y asignar el nuevo baricentro de cada cluster



# Modelos de ML basados en Clustering:k-means

**FIN:** El modelo está listo



# Modelos de ML basados en Clustering:k-means

## Selección del número óptimo de clusters.

El principal problema de los métodos de Clustering es la elección del número de Clusters, ya que las agrupaciones de datos pueden ser muy heterogéneas (pocos Clusters); o datos que siendo muy similares se agrupen en Clusters diferentes (muchos Clusters).

Existen mecanismos para seleccionar el número de clusters óptimo, entre los que destacan: elbow method (método del codo), Calinsky, Affinity Propagation (AP), Gap y Dendrogramas.

# Modelos de ML basados en Clustering:k-means

## **Selección del número óptimo de clusters.**

El método del codo es uno de los más extendidos cuando se aplica el K-Means. Este método se basa en la aplicación de la técnica Within Clusters Summed Squares (WCSS). El objetivo es generar un conjunto de valores que representan la inercia del algoritmo “k-means” tras cada ejecución del k-means con un número determinado de clusters utilizando la misma semilla de aleatoriedad. Dicho conjunto de valore se visualiza posteriormente para encontrar el “punto de inflexión” en donde la curva se estabiliza.

# Modelos de ML basados en Clustering:k-means

# Clustering con K-means

#Paso 1. Importar los datos

```
dataset = read.csv("Datasets/Mall_Customers.csv")
```

#Se seleccionan solamente aquellas variables que se deben agrupar utilizando el mecanismo de agrupación (clustering)

```
X = dataset[, 4:5]
```

#Paso 2. Método del codo

#El método del codo representa la fórmula estadística WCSS (Within-Cluster-Sum-of-Squares) que permite encontrar el número de centroides (agrupaciones) óptimo.

#Se trata de un mecanismo que suma las distancias de cada observación con respecto a su correspondiente centroide y las eleva al cuadrado.

#El objetivo es encontrar cuál es número de centroides que seleccionados al azar, aporta un número adecuado de observaciones clasificadas o agrupadas.

```
set.seed(1234)
```

```
wcss = vector()
```

```
for (i in 1:10){
```

```
  wcss[i] <- sum(kmeans(X, i)$withinss)
```

```
}
```

#Paso 3. Visualizar el método del codo para ver el valor adecuado para el algoritmo kmeans

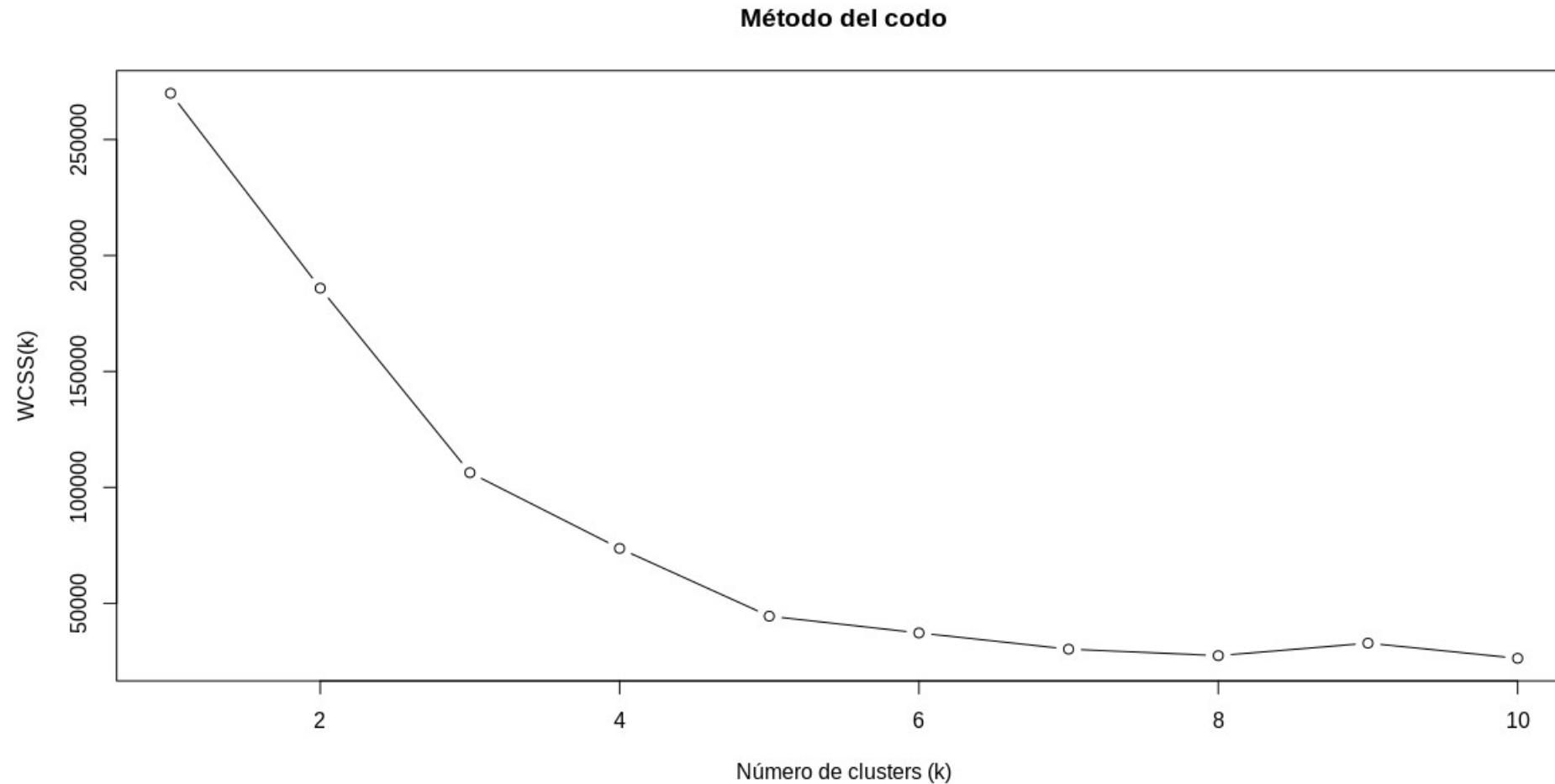
```
plot(1:10, wcss, type = 'b', main = "Método del codo",
     xlab = "Número de clusters (k)", ylab = "WCSS(k)")
```

#Paso 4. Aplicar el algoritmo de k-means con k óptimo generado por el método del codo, que en este caso, tal como se pudo ver en el gráfico anterior es 5.

```
set.seed(5678)
```

```
kmeans <- kmeans(X, 5, iter.max = 300, nstart = 10)
```

# Modelos de ML basados en Clustering:k-means



# Modelos de ML basados en Clustering:k-means

#Paso 4. Aplicar el algoritmo de k-means con k óptimo generado por el método del codo, que en este caso, tal como se pudo ver en el gráfico anterior es 5.  
set.seed(5678)

```
kmeans <- kmeans(X, 5, iter.max = 300, nstart = 10)
```

#Paso 5. Visualización de los clusters.

#La librería cluster permite visualizar las agrupaciones de clusters generadas por algoritmos como k-means.

```
#install.packages("cluster")
library(cluster)
clusplot(X, #Dataframe con todas las observaciones
          kmeans$cluster, #Enseña los clusters a los que pertenece cada punto.
          lines = 0, #No unir por líneas los puntos.
          shade = TRUE, #Marcar los clusters.
          color = TRUE, #Pintar colores.
          labels = 2, #Etiquetar los puntos. Ver la documentación para ver cómo se etiqueta el plot.
          plotchar = FALSE, #Enseñar símbolos diferentes para cada cluster (no)
          span = TRUE, #Aparece cada cluster representado con una elipse.
          main = "Clustering de clientes",
          xlab = "Ingresos anuales",
          ylab = "Puntuación (1-100)")
)
```

# Modelos de Clustering jerárquico

- Se basa en un modelo aditivo o aglomerativo, en el que se seleccionan clusters al azar (que representan observaciones del conjunto de datos) y posteriormente se van creando clusters en función de la distancia entre cada punto.
- Los clusters se van juntando poco a poco en la media en la que se encuentran más cercanos y este comportamiento se repite hasta que solamente queda un único cluster, que representa el conjunto de datos completo.
- Y ahora, la pregunta lógica sería, para qué hacer esto si el resultado final es equivalente a tener un único cluster con todos los datos del dataset? La respuesta a esta pregunta está en que durante cada iteración del modelo, se pueden representar los clusters en dendrogramas y medir las distancias euclidianas más largas y determinar el número de agrupaciones o clusters óptimo.

# Modelos de Clustering jerárquico

# Clustering Jerárquico

#Paso 1: Importar los datos.

```
dataset = read.csv("Datasets/Mall_Customers.csv")
```

#Se seleccionan solamente aquellas variables que se deben agrupar utilizando el mecanismo de agrupación (clustering)

```
X = dataset[, 4:5]
```

#Paso 2: Utilizar el dendrograma para encontrar el número óptimo de clusters.

#Se utiliza la función "hclust" que precisamente permite crear un cluster jerárquico.

#La función recibe una matriz de distancias con el objetivo de determinar la distancia euclíadiana entre cada observación del dataset.

#En el algoritmo de kmeans era necesario encontrar el número óptimo de centros para crear las agrupaciones.

#En este caso dicho concepto no existe, por lo tanto lo que se hace para determinar el número de clusters es utilizar la varianza

#de todos los puntos con un método conocido como ward

```
dendrogram = hclust(dist(X,  
                           method = "euclidean"),  
                           method = "ward.D")
```

#Paso 3: Visualizar el dendrograma.

```
plot(dendrogram,  
      main = "Dendrograma",  
      xlab = "Clientes del centro comercial",  
      ylab = "Distancia Euclídea")
```

# Modelos de Clustering jerárquico

#Paso 3: Visualizar el dendrograma.

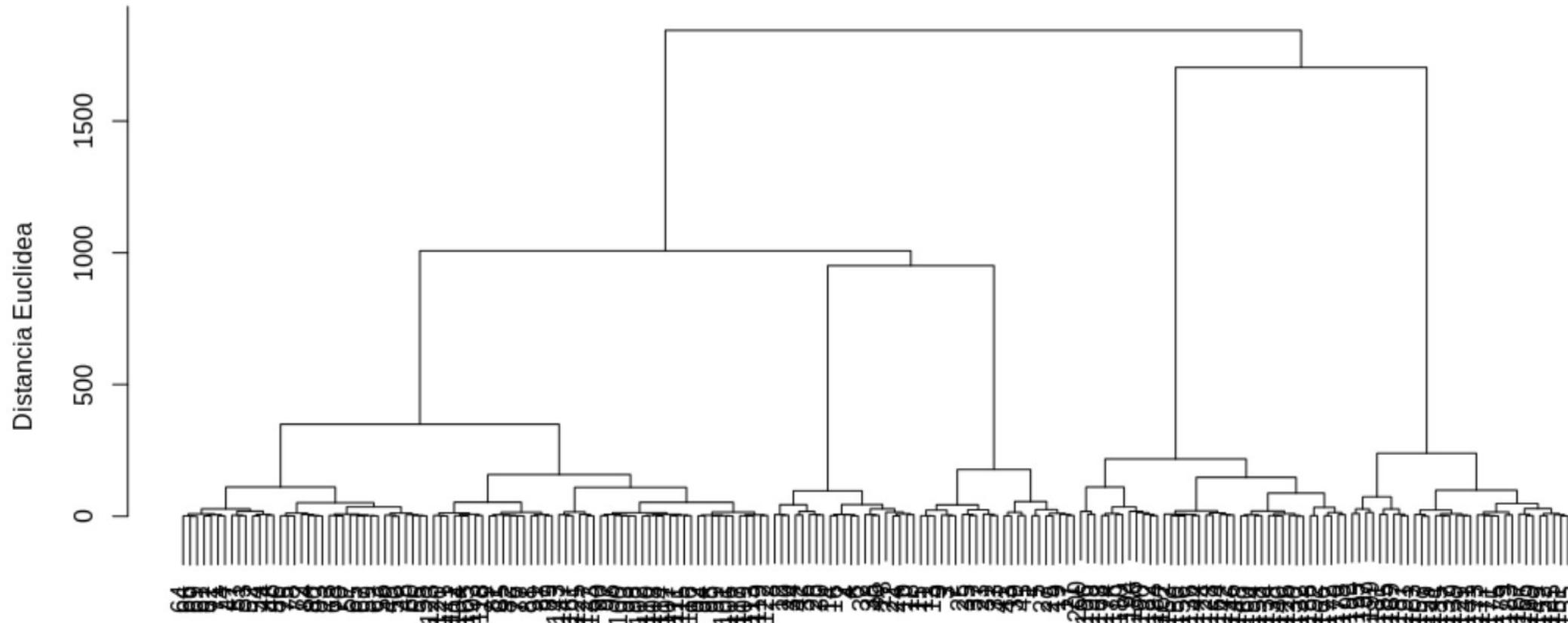
```
plot(dendrogram,
      main = "Dendrograma",
      xlab = "Clientes del centro comercial",
      ylab = "Distancia Euclídea")
```

#Paso 3.1: Análisis del dendrograma.

```
# En este caso, es necesario determinar la línea de corte. Como norma general, las agrupaciones deben ser lo más homogéneas posibles.
# Para ello, se debe seleccionar la distancia más larga sin cortes.
# Hay un pequeño "truco para esto". Basta simplemente con extender todas las líneas horizontales que ha generado el dendrograma y a continuación,
# seleccionar la línea vertical más larga sin cortes (es decir, la distancia euclídea más larga).
# En el caso del ejemplo anterior es difícil verlo porque las líneas parecen muy iguales, pero se selecciona finalmente la distancia entre ~300 y ~900.
# Una vez seleccionada la distancia euclídea más larga, se cuentan las líneas paralelas verticales a esta, lo que representa el número de clusters,
# que en el caso de este ejemplo es 5.
```

# Modelos de Clustering jerárquico

Dendrograma



# Modelos de Clustering jerárquico

```
# Paso 4: Generar el cluster jerárquico como se ha hecho antes
hc = hclust(dist(X, method = "euclidean"),
             method = "ward.D")
# Con la función "cutree" se corta el árbol en el número de clusters determinado.
# El resultado es un vector en donde se indica cada observación en qué cluster ha quedado (se verá un número de cluster entre 1 y 5 en éste caso)
y_hc = cutree(hc, k=5)
#> y_hc
# [1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 3 1
# [46] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
# [91] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
# [136] 4 5 4 5 4 5 4 3 4 5 4 3 4 5 4 5 4 5 4 5 4 3 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4
# [181] 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4
```

#Paso 5. Visualizar los clusters.  
#Se utilizan las mismas instrucciones que se han visto para kmeans utilizando la librería de visualización "cluster".

```
#install.packages("cluster")
library(cluster)
clusplot(X,
         y_hc,
         lines = 0,
         shade = TRUE,
         color = TRUE,
         labels = 2,
         plotchar = FALSE,
         span = TRUE,
         main = "Clustering de clientes",
         xlab = "Ingresos anuales",
         ylab = "Puntuación (1-100)")
```

# Diferencias modelos de clustering

Clustering Model	Pros	Cons
K-Means	Simple to understand, easily adaptable, works well on small or large datasets, fast, efficient and performant	Need to choose the number of clusters
Hierarchical Clustering	The optimal number of clusters can be obtained by the model itself, practical visualisation with the dendrogram	Not appropriate for large datasets