Curso : MACHINE LEARNING CON R

Tema : Preprocesamiento de Datos y Primeros Modelos de Predicción y Clasificación

Docente : Victor Henostroza

**GUIA DE LABORATORIO 1**

**Objetivos del Procesamiento de Datos:**

1. **Desarrollar nuestra plantilla de Preprocesamiento de Datos.**
2. **Desarrollar nuestro primer modelo de Predicción con Regresión Lineal Simple y una variante con Regresión Lineal Polinómica.**
3. **Desarrollar nuestro primer modelo de Clasificación con Regresión Logística.**

**---------------------------------------------------------------------------------------------------------**

**Objetivo 1:**

La idea de desarrollar una plantilla de procesamiento es que el alumno pueda tenerla para aplicarla a modo de base para los futuros desarrollos de sus modelos de machine learning.

1. **Importar el dataset en nuestro script (crea el dataset con la función read.csv)**

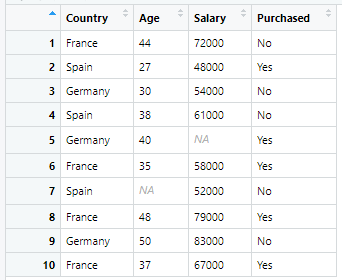
dataset = read.csv("Data.csv", header = TRUE, sep = ",", dec = ".",

na.strings = "", stringsAsFactors = FALSE)

**A tener en cuenta:**

* Los argumentos de la función read.csv deben ir dentro de los paréntesis y separados por comas (esto es común a casi todas las funciones de R).
* **El primer argumento** es el nombre de nuestro archivo csv tal cual está grabado en disco (es importante asegurarse que el directorio de trabajo está marcado en la carpeta que contiene el archivo, caso contrario se necesitará especificar toda la ruta).
* **header** indica si el archivo tiene o no cabecera con los nombres de las variables, para nuestro caso ponemos TRUE porque si existen cabeceras.
* **Sep** indica el tipo de separador que usan los datos, en nuestro caso la coma (algunos archivos pueden usar tabuladores o punto y coma, entre otros).
* **na.strings** indica que si hay espacios vacíos en las variables categóricas las convierte en NAs (esto para estandarizar los datos perdidos).
* **stringsAsFactors** indica si se desea o no que se tome como factores a los valores categóricos encontrados, para nuestro caso indicamos que no con FALSE.

Después de aplicar la sentencia anterior nuestro dataset quedará mostrado de la siguiente manera:

****

De la tabla, podemos apreciar que existen variables numéricas y también categóricas, además vemos que hay presencia de valores perdidos NA (not available) en las variables Age (Edad) y Salary (Salario). Este dataset es pequeño para facilitar la visualización de los cambios en el mismo, pero para datasets más grandes podemos hacer las siguientes sentencias para poder visualizar los NAs:

**#Analizando el comportamiento de los NAs en las variables**

#install.packages("VIM")

library(VIM)

aggr(dataset,

col= c('green', 'red'), #colores del gráfico

numbers = TRUE, #indicador de proporciones mostradas por números

sortVars = TRUE, #ordena variables por apariencia de NAs de más a menos

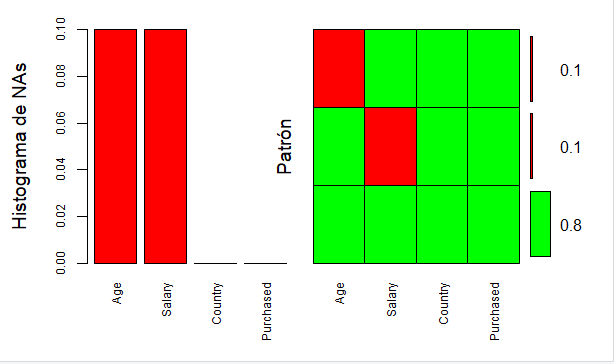
labels = names(dataset), #pone las etiquetas de acuerdo al nombre de las columnas

cex.axis = 0.75, #ancho de barras

gap = 1, #distancia entre gráficos

ylab = c("Histograma de NAs", "Patrón")) #título de las variables

**# Con dichas sentencias, obtendremos el siguiente gráfico:**

****

El gráfico izquierdo muestra el histograma con las frecuencias relativas de NAs, como vimos, solo había un caso faltante tanto en la columna Age así como en la columna Salary, por lo tanto dicha frecuencia es de 10% para cada una de dichas variables.

El gráfico derecho muestra el patrón de los valores NAs para todas las columnas, en nuestro caso no hay dos o más NAs por fila (solo hay uno por fila), por lo tanto solo se pintan con rojo los cuadrados que se muestran, apreciándose que el 80% de los datos están libres de NAs.

**Observación:** Si deseamos ver en la consola cuales son los nombres de las variables de nuestro dataset usamos la siguiente sentencia:

> names(dataset)

[1] "Country" "Age" "Salary" "Purchased"

"

**Observación:** Si deseamos cambiar los nombres de las columnas de nuestro dataset a unos que sean más amigables para nosotros usamos la siguiente instrucción:

dataset2 = read.csv("Data.csv", header = TRUE, col.names = c("país", "edad", "salario",

"compra\_o\_no\_compra"))

* **El argumento col.names permite que se le indiquen cuáles serán los nombres de las columnas separados por comas.**

**Observación:** Si deseamos ver en la consola las 3 primeras filas de datos de nuestro dataset utilizaremos la siguiente sentencia:

> head(dataset, 3)

Country Age Salary Purchased

1 France 44 72000 No

2 Spain 27 48000 Yes

3 Germany 30 54000 No

1. **Llenado de valores perdidos (NAs) en el Dataset**

Debemos reemplazar los datos perdidos con la media (datos numéricos) o con un valor aleatorio tomado de la misma variable (datos categóricos). Para lograrlo haremos las siguientes instrucciones:

**Caso 1: Rellenando NAs con valores centrales (media)**

**#Rellenando NAs en columna Age con la media de sus datos disponibles.**

dataset$Age = ifelse(is.na(dataset$Age),

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

dataset$Age)

**#Rellenando NAs en columna Salary con la media de sus datos disponibles.**

dataset$Salary = ifelse(is.na(dataset$Salary),

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

dataset$Salary)

**Observación:** Se han actualizado los valores de las columnas Age y Salary dependiendo de si el valor de dicha variable es NA o no, si es NA se reemplaza con la media, si no lo es se deja como estaba.

**#ifelse**(proposición a validar, instrucciones a hacer si es verdadera, instrucciones a hacer si es falsa)

**#na.rm = TRUE** (elimina internamente a los valores NA para poder realizar el cálculo de la media.

**Caso 2: Rellenando NAs en la variable con un valor aleatorio de los que tiene disponibles**

**#Rellenar los valores NA con valores aleatorios de la variable Salary (opcional).**

# Primero crearemos una función que impute los NAs de una variable

rand.impute = function(x) {

# missing contiene un vector de valores T/F dependiendo del NA de x

missing = is.na(x)

#n.missing contiene cuantos valores son NA dentro de x

n.missing = sum(missing)

#x.obs son los valores conocidos que tienen dato diferente de NA en x

x.obs = x[!missing]

#por defecto, devolverá lo mismo que había entrado por parámetro

imputed = x

#en los valores que faltaban, los reemplazamos por una muestra de los que si conocemos (MAS)

imputed[missing] = sample(x.obs, n.missing, replace = TRUE)

return (imputed)}

#sample(vector de datos a muestrear, número de aleatorios totales, replace =con reemplazo)

#Llamando a la función para ser aplicada a la columna Salary

dataset$Salary = rand.impute(dataset$Salary)

**Caso 3: Eliminando las filas que contengan algún NA en sus variables (no recomendable)**

**# Crearemos un dataset nuevo a partir de una copia del archivi csv desde cero**

dataset1 = read.csv("Data.csv", header = TRUE, sep = ",", dec = ".",

na.strings = "", stringsAsFactors = FALSE)

dataset1 = na.omit(dataset1)

**Observación:** Si se desea eliminar variables creadas a partir de otras o viceversa podemos hacer las siguientes instrucciones:

dataset1 = dataset #Creamos un nuevo dataset idéntico al primero

borrar = c("Salary","Purchased") #Creamos un vector con las columnas a eliminar

dataset1 = dataset1[ , !(names(dataset1) %in% borrar)] #Elimina las columnas finalmente

1. **Generación de factores con datos categóricos**

En algunas ocasiones necesitaremos modificar las variables categóricas a números para que puedan ser evaluadas con algoritmos de predicción, con esto se pasa cada categoría a un número entero. La manera de hacerlo es la siguiente:

# factor(variable, niveles del factor, etiquetas del factor)

# Codificando la variable "Country"

dataset$Country = factor(dataset$Country,

levels = c("France", "Spain", "Germany"),

labels = c(1, 2, 3))

# Codificando la variable "Purchased"

dataset$Purchased = factor(dataset$Purchased,

levels = c("No", "Yes"),

labels = c(0,1))

1. **Generación de variables dummy con datos categóricos**

Para las variables categóricas que tengan más de dos posibles valores (como nuestra variable country que tiene 3 categorías) es necesario generar nuevas variables que contengan ceros y unos (el número de variables es uno menos al total de categorías para evitar multicolinealidad), los ceros deben ir donde no se activa la variable en la fila y los unos en donde si está activa. Para lograrlo, haremos las siguientes instrucciones:

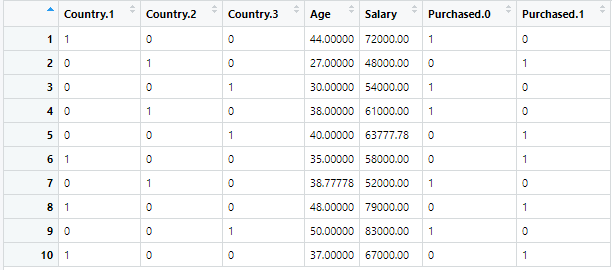
# Generando variables dummy para las variables categorícas (opcional)

# install.packages("dummies")

library(dummies)

dummed\_dataset = dummy.data.frame(dataset, names = c("Country", "Purchased"), sep = ".")

Tras esta instrucción nuestro dataset Nuevo quedaría dela siguiente forma:



De la tabla, observamos que la variable country se ha dividido en 3 3 columnas que contienen ceros y unos, respectivamente para la no presencia o si presencia de cada categoría en la columna inicial (lo mismo para la variable Purchased).

1. **Generación de agrupaciones con datos numéricos**

En algunas ocasiones necesitaremos generar grupos con datos numéricos de las variables para poder trabajarlas con algoritmos de clasificación, para esto podemos categorizar dichas variables, existen dos formas las cuales tratamos a continuación:

**# Caso1: Categorizar la variable Salary de un nuevo dataset con cortes elegidos**

dataset2 = read.csv("Data.csv", header = TRUE, sep = ",", dec = ".",

na.strings = "", stringsAsFactors = FALSE)

bp <- c(-Inf, 50000, 75000, Inf) #vector de cortes

names <- c("Low", "Average", "High") #vector de etiquetas

# cut(vector a cortar, breaks = puntos de corte, labels = etiquetas)

dataset2$Salary1 <- cut(dataset2$Salary, breaks = bp, labels = names)

**# Caso 2: Categorizar la variable Salary con número de cortes indicados por el usuario**

dataset2$Salary2 <- cut(dataset2$Salary, breaks = 4,

labels = c("Level 1", "Level 2", "Level 3", "Level 4"))

1. **Dividir el dataset en conjunto de entrenamiento y de testeo**

Para poder validar el modelo, se debe dividir el conjunto de datos de entrada en dos partes, una para entrenar los algoritmos (conjunto de entrenamiento) y otra para validarlos (conjunto de testeo). Para esto utilizaremos el archivo Data.csv y haremos las siguientes instrucciones:

# install.packages("caTools")

library(caTools)

set.seed(123) #fija los números aleatorios que partirán los dos conjuntos

split = sample.split(dataset$Purchased, SplitRatio = 0.8)

#crear variables independientes

training\_set = subset(dataset, split == TRUE)

#crea variable dependiente

testing\_set = subset(dataset, split == FALSE)

**Observación:** El objeto creado con nombre split contiene un vector con valores true y false, donde los true son los valores que serán usados para entrenar y los false serán usados para testear.

> split

[1] TRUE TRUE TRUE TRUE TRUE FALSE TRUE TRUE FALSE TRUE

1. **Normalizado o escalada de variables**

La normalización de variables no es obligatoria, pero para la gran mayoría de casos que se presenten es recomendable hacerlo, sobre todo si se usará algún algoritmo basado en distancias, esto debido a que si existen diferencias significativas en las varianzas entre las variables predictores se daría mayor peso a las variables con mayor varianza y eso afectaría las conclusiones. Para realizar la normalización de variables haremos las siguientes sentencias:

**# scale(variables a normalizar del dataset)**

training\_set[,2:3] = scale(training\_set[,2:3]) #escalado de conjunto de entrenamiento

testing\_set[,2:3] = scale(testing\_set[,2:3]) #escalado de conjunto de testeo

**Observación:** La notación para el manejo de datasets por filas y columnas es la siguiente:

**#Dataset[fila\_1:fila\_n,columna\_1:columna:n]**

#Dataset[,2:5] #selecciona todas las filas y las columnas desde la 2da hasta la 5ta

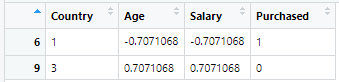
#Dataset[,-4] #selecciona todas las filas y todas las columnas con excepción de la 4ta

**Conclusión:**

Finalizando el laboratorio, hemos logrado generar un dataset final que sí se podrá utilizar como entrada para algún algoritmo de machine learning, pues ya tenemos todas las variables codificadas o categorizadas, además, de que las variables numéricas ya están normalizadas, también, hemos completado todas las filas sin NAs. Sin embargo, aún podemos explorar más a fondo la data final que tenemos con técnicas que explicaremos en el siguiente laboratorio para que finalmente quede un dataset más representativa aún de nuestros casos.

**A continuación, se muestra nuestros datasets de entrenamiento y testeo finales:**

****

****

**Objetivo 2:**

Ahora haremos nuestro primer modelo de predicción, para el cual nos valdremos de un dataset que muestra los diferentes puestos de una empresa por niveles y su respectivo salario de acuerdo al mercado. Haremos un modelo de regresión lineal simple y otro de regresión lineal polinómica para ver con cual de estos modelos podemos predecir mejor. Utilizaremos esta vez todo el dataset para entrenar nuestro modelo y con el este decidir cuánto pagarle a un futuro empleado que viene con 6,5 años de experiencia a la empresa puesto que ha sido consultor independiente y no ha ocupado un puesto equivalente a los del dataset.

**# Regresión Simple y Polinómica**

**# Importar el dataset**

dataset = read.csv('Position\_Salaries.csv')

dataset = dataset[, 2:3]

**# Ajustar Modelo de Regresión Lineal con el Conjunto de Datos**

lin\_reg = lm(formula = Salary ~ .,

data = dataset)

**# Ajustar Modelo de Regresión Polinómica con el Conjunto de Datos**

**# Creamos tres columnas extra que tengan la variable Level al cuadrado, al cubo y a la cuarta**

dataset$Level2 = dataset$Level^2

dataset$Level3 = dataset$Level^3

dataset$Level4 = dataset$Level^4

poly\_reg = lm(formula = Salary ~ .,

data = dataset)

**# Visualización del modelo lineal**

# install.packages("ggplot2")

library(ggplot2)

ggplot() +

geom\_point(aes(x = dataset$Level , y = dataset$Salary),

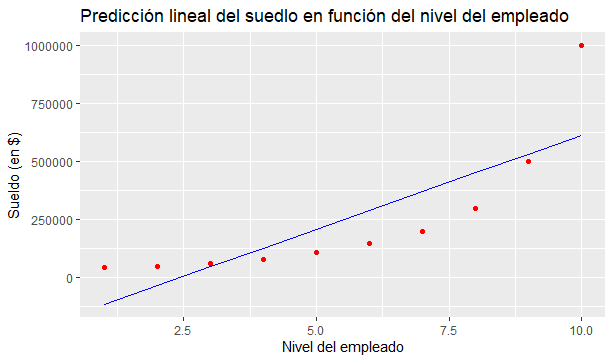
color = "red") +

geom\_line(aes(x = dataset$Level, y = predict(lin\_reg, newdata = dataset)),

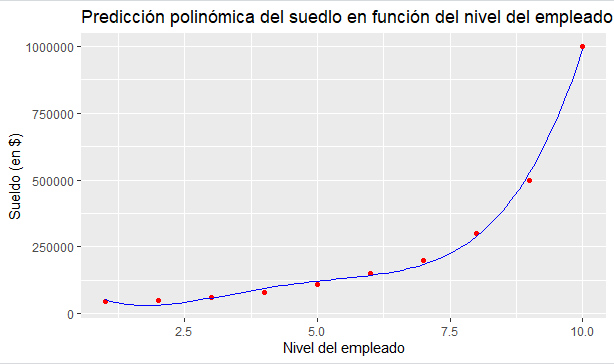
color = "blue") +

ggtitle("PredicciÃ³n lineal del suedlo en funciÃ³n del nivel del empleado") +

xlab("Nivel del empleado") + ylab("Sueldo (en $)")



Del gráfico, apreciamos que nuestros datos se dispersan demasiado de la gráfica de predicción, inclusive muy por debajo de cero para el sueldo cuando el empleado es nuevo (0 años de experiencia) y para el nivel 10 está muy por debajo también del valor real de sueldo.



Del gráfico, observamos que para este modelo de predicción con potencias para la variable independiente si se están ajustando los valores predichos de sueldo a los reales, tanto el nivel 0 como el nivel 10 están recibiendo una remuneración más cercana a la real.

**# Predicción de nuevos resultados con Regresión Lineal**

y\_pred = predict(lin\_reg, newdata = data.frame(Level = 6.5))

> y\_pred

1

330378.8 **# Valor muy por encima del que debería pagársele al empleado**

**# Predicción de nuevos resultados con Regresión Polinómica**

y\_pred\_poly = predict(poly\_reg, newdata = data.frame(Level = 6.5,

Level2 = 6.5^2,

Level3 = 6.5^3,

Level4 = 6.5^4))

> y\_pred\_poly

1

158862.5 **# Valor mucho más cercano al real a pagar al empleado**

**Conclusión:**

Vemos que nuestro modelo de regresión lineal polinómica resultó ser mejor que el de regresión lineal simple para predecir el ingreso de un nuevo dato y que en general se ajustó mejor a los datos reales.

**Objetivo 3:**

Para hacer nuestro modelo de clasificación con regresión logística utilizaremos el dataset “Social\_Network\_Ads”, el cual nos muestra las columnas edad (Age) y salario anual (EstimatedSalary) que serán nuestras variables independientes. La variable a clasificar será Compra (Purchased) la cual es 1 cuando sí compran un artículo y 0 cuando no compran dicho artículo.

**# Regresión Logística**

**# Importar el dataset**

dataset = read.csv('Social\_Network\_Ads.csv')

**# Nos qudamos solo con las columnas de la 3 a la 5 por ser las dos primeras no significativas**

dataset = dataset[, 3:5]

**# 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)

**# Como tenemos 400 casos, nos quedamos con 300 para entrenar y 100 para testear**

training\_set = subset(dataset, split == TRUE)

testing\_set = subset(dataset, split == FALSE)

**# Escalado de valores**

**# Necesitamos escalar porque existe mucha diferencia entre las varianzas de las variables**

training\_set[,1:2] = scale(training\_set[,1:2])

testing\_set[,1:2] = scale(testing\_set[,1:2])

**# Ajustar el modelo de regresión logística con el conjunto de entrenamiento.**

**# Utilizamos family =binomial porque nuestra respuesta tiene dos valores 0 y 1**

classifier = glm(formula = Purchased ~ .,

data = training\_set,

family = binomial)

**# Predicción de los resultados con el conjunto de testing**

prob\_pred = predict(classifier, type = "response",

newdata = testing\_set[,-3])

> prob\_pred

2 4 5 9 12 18

0.0212876085 0.0125086559 0.0035905256 0.0029370720 0.0080066561 0.2549200385

19 20 22 29 32 34

0.3227290281 0.4506350758 0.5857219587 0.0118599288 0.2342624813 0.0096172875

35 38 45 46 48 52

La variable prob\_pred tiene las probabilidades de ocurrencia de compra, ahora necesitamos traducir esas probabilidades a ceros y unos para poder compararlas con los valores reales.

**y\_pred = ifelse(prob\_pred> 0.5, 1, 0)**

# y\_pred será 1 cuando prob\_pred sea mayor a 0.5 y será 0 en caso contrario

> y\_pred

2 4 5 9 12 18 19 20 22 29 32 34 35 38 45 46 48 52 66 69 74

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

75 82 84 85 86 87 89 103 104 107 108 109 117 124 126 127 131 134 139 148 154

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

**# Crear la matriz de confusión**

cm = table(testing\_set[, 3], y\_pred)

> cm

y\_pred

0 1

0 57 7

1 10 26

De la sentencia anterior, obtenemos la cantidad de aciertos sobre el total de la siguiente forma:

**(57+26) / (57+7+10+26) = 0.83**

Es decir, hemos acertado en la clasificación del 83% de los casos.

**# 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)',

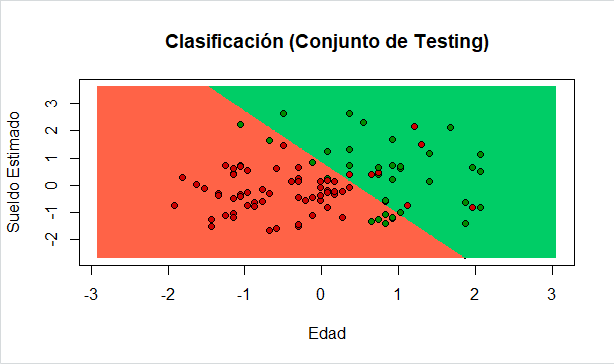
xlab = 'Edad', ylab = 'Sueldo Estimado',

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'))



Del gráfico, apreciamos dos zonas de colores, la zona verde corresponde a donde se espera que se compre el artículo y la zona roja donde no se espera dicha compra. Los puntos verdes son los que han sido predichos como compras y los rojos como no compras, por lo tanto, apreciamos que la mayoría de ellos están cayendo en las zonas correctas de clasificación y los que no corresponden a los puntos mal clasificados según la matriz de confusión.

**Conclusión:**

Gracias a nuestro modelo de regresión logística logramos clasificar correctamente al 83% de los datos del conjunto de testing, este porcentaje es bastante bueno para un modelo de clasificación.