Master en Big Data. Fundamentos Matemáticos del Análisis de Datos (FMAD).

Práctica 2

Fernando San Segundo

Curso 2021-22. Última actualización: 2021-09-16

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI
DEPARTAMENTO DE MATEMÁTICA APLICADA

Preliminares

Carga de librerías inicial:

```
library(tidyverse)
```

Cálculo de probabilidades en ficheros de datos.

- Vamos a empezar volviendo a las preguntas sobre el Teorema de Bayes que dejamos pendientes en la sesión 3, usando los datos de `spam` de la librería `kernlab`.

```
library(kernlab)
data(spam)
spam[1:4, c(5:10, 58)]
```

```
   our over remove internet order mail type
1 0.32 0.00   0.00     0.00  0.00 0.00 spam
2 0.14 0.28   0.21     0.07  0.00 0.94 spam
3 1.23 0.19   0.19     0.12  0.64 0.25 spam
4 0.63 0.00   0.31     0.63  0.31 0.63 spam
```

Recuerda las preguntas:

- ¿Cuál es la probabilidad de que un mensaje elegido al azar sea spam?
- ¿Cuál es la probabilidad de que un mensaje elegido al azar contenga la palabra *order*?
- *Sabiendo* que un mensaje es spam, ¿cuál es la probabilidad de que contenga la palabra *order*?
- Y ahora, usando la fórmula de Bayes, vamos a construir el programa antispam más simple del mundo: sabiendo que un mensaje contiene la palabra *order*, ¿cuál es la probabilidad de que sea spam?

Probabilidad de spam

- Vamos a usar `dplyr y a guardar el resultado en la variablepSpam`. Casi todo el trabajo que hay que hacer se puede hacer con código sencillo como este:

```
spam %>%
  count(type) %>%
  mutate(probs = n / sum(n))
```

```
     type    n     probs
1 nonspam 2788 0.6059552
2    spam 1813 0.3940448
```

En clase discutiremos modificaciones y alternativas sobre este código base. Modifícalo después para obtener la probabilidad de `order` y guárdala en `pOrder`.

Probabilidad de que contenga la palabra `order` *sabiendo* que es spam.

- Estas probabilidades *sabiendo algo* son siempre probabilidades condicionadas. Y el suceso que sabemos que ha ocurrido es el que condiciona. Así que en este caso se trata de calcular $P(\text{ contiene order} \,|\, \text{spam})$.
- La definición dice que esto es: $\dfrac{P(\text{spam} \cap \text{ contiene order})}{P(\text{ spam })}$. Antes de mirar una posible solución en la siguiente página, piensa en como calcular el numerador con `dplyr`. Guarda el resultado en `P_spam_y_order`.

- Esta es una de las (muchas) soluciones posibles:

```
P_spam_y_order = spam %>%
  summarise(prob = mean((order > 0) & (type == "spam"))) %>%
      .[1, 1]
```

- Y ahora la probabilidad condicionada $P(\text{spam} \mid \text{contiene order})$ es:

```
(P_order_si_spam = P_spam_y_order / pSpam)
```

```
[1] 0.3061224
```

- Vamos a comprobar el T. de Bayes calculando la condicionada recíproca $P(\text{spam} \mid \text{contiene order})$ de dos maneras. Una directa:

```
(P_spam_si_order = P_spam_y_order / pOrder)
```

```
[1] 0.7179819
```

y otra que es la aplicación directa del teorema:

```
P_spam_si_order =
   (P_order_si_spam * pSpam) / (P_order_si_spam * pSpam +  P_order_si_noSpam * pNoSpam)
```

Dejamos como ejercicio para clase el cálculo de las probabilidades que faltan en esta expresión. *Indicación:* una de ellas es trivial.

Tablas de contingencia

- Vamos a usar el fichero de datos de este enlace para practicar el lenguaje de las tablas de contingencia. El fichero contiene datos de `churn` (clientes que se dan de baja) de una compañía de telecomunicación.
- Usa dplyr y además usa como referencia el script de tablas de contingencia asociado a la Sesión 3 del curso para completar estos ejercicios:
  - Construye un factor binario `Contrato` con dos niveles llamados `mensual`, y `largo` según que los valores de `PaymentMethod` sean respectivamente `Month-to-month` o algún valor diferente (anual, bianual).
  - Haz una tabla de contigencia 2x2 de ese factor contrato frente al factor `Churn`.
  - Calcula la probabilidad de que un cliente se de de baja (`Churn` es 'Yes') sabiendo que su contrato es mensual.
  - Calcula la probabilidad de que un cliente tenga un contrato mensual sabiendo que se da de baja.
  - Calcula la probabilidad de que un cliente se de baja **y** tenga un contrato mensual.
  - Vamos a pensar en un modelo (extremadamente simplista) en el que usamos los valores de `Contrato` para predecir las bajas de los clientes. La *precision* (*accuracy*) del modelo es la tasa de aciertos (positivos verdaderos TP y negativos verdaderos TN). Calcula esa precisión para este modelo.
  - La *sensibilidad* de una prueba diagnóstica es la tasa de positivos verdaderos sobre el total de casos reales (total de enfermos). Calcula la sensibilidad cuando consideramos un contrato anual como un test positivo para *diagnosticar* el churn.
  - De forma análoga, la *especificidad* es la tasa de negativos verdaderos sobre el total de *sanos*. Calcula la especificidad en este ejemplo.
  - Mira los enlaces de la Wikipedia sobre Confusion matrix y Contingency table.

La binomial con R.

- Ver Sesión 3, página 15.
- Usaremos el Cuestionario 2 de Moodle para practicar con algunos ejemplos.

Datos limpios (tidy data).

- El Capítulo 12 de *R for Data Science* es una lectura casi obligada, ya que H. Wickham es el creador del concepto de *datos limpios*.
- Un conjunto de datos se considera *limpio* si cumple estas tres condiciones:
  1. Cada variable tiene su propia columna.
  2. Cada observación tiene su propia fila.
  3. Cada valor tiene su propia celda.

Ejemplos de datos no limpios.

- Por ejemplo, consideremos este conjunto de datos que corresponde al importe (en miles de euros) de ciertas operaciones de venta en tres tiendas de una cadena de supermercados.

```
(ventas <- tibble( tienda = c("A", "B", "C"),
                   enero = c(12, 23, 15),
                   febrero = c(16, 11, 19),
                   marzo = c(8, 12, 15)
                   ))
```

```
# A tibble: 3 x 4
  tienda enero febrero marzo
  <chr>  <dbl>   <dbl> <dbl>
1 A         12      16     8
2 B         23      11    12
3 C         15      19    15
```

Este conjunto de datos no es limpio, porque las filas no corresponden a observaciones y las columnas no corresponden a variables. Las variables de este conjunto de datos son `tienda`, `mes`, `ventas`. Una observación (fila) debería mostrar los valores conjuntos de las tres variables. Hay que convertir los valores de `mes` en una columna y eso hará la tabla más larga (y estrecha)

Limpiando la tabla con pivot_longer

- La función pivot_longer de *tidyR* hace precisamente eso:

```
(ventasTidy = ventas %>%
pivot_longer(enero:marzo, names_to = "mes"))
```

```
# A tibble: 9 x 3
  tienda mes     value
  <chr>  <chr>   <dbl>
1 A      enero      12
2 A      febrero    16
3 A      marzo       8
4 B      enero      23
5 B      febrero    11
6 B      marzo      12
7 C      enero      15
8 C      febrero    19
9 C      marzo      15
```

Esta función se usa cuando los nombres de algunas columnas de la tabla no son realmente variables, sino valores de una variable (que no aparece por su nombre en la tabla). Por ejemplo, algunas columnas pueden ser nombres de países, años, etc.

Otra situación distinta con datos `not tidy`

- Este otro ejemplo usa un conjunto de datos `pacientes` que contiene información sobre los pacientes de una clínica: un código que identifica al paciente, los meses que hace que esa persona es paciente y el número de visitas que ha realizado a la clínica. Y tampoco es un conjunto de datos limpio porque hay variables distintas almacenadas en una misma columna. Para verlo mejor: ¿en qué unidades se miden los valores de la columna `valor`?

```
# A tibble: 8 x 3
  codigo tipo    valor
  <chr>  <fct>   <int>
1 A      meses       3
2 A      visitas     9
3 B      meses       3
4 B      visitas     8
5 C      meses       1
6 C      visitas     4
7 D      meses       5
8 D      visitas     8
```

  En este caso para representar las variables `visitas` y `meses` debemos colocar cada una en su propia columna, haciendo la tabla más ancha (y corta).

Limpiando la tabla con `pivot_wider`

- Y de nuevo *tidyR* viene al rescate con una función que hace precisamente eso:

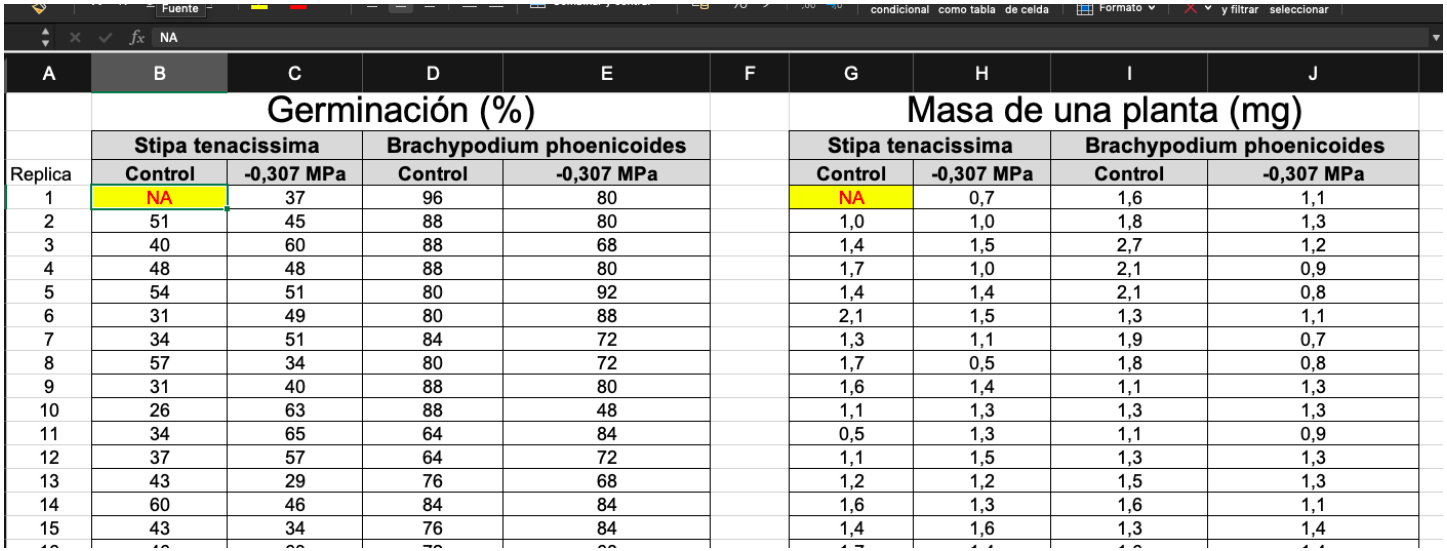
```
pacientes %>%
  pivot_wider(names_from = tipo,
              values_from = valor)
```

```
# A tibble: 4 x 3
  codigo meses visitas
  <chr>  <int>   <int>
1 A          3       9
2 B          3       8
3 C          1       4
4 D          5       8
```

Usamos esta función cuando una observación está repartida en varias filas de la tabla o cuando una columna contiene nombres de variables.

Más ejemplos. . .

- En otro ejemplo, unos biólogos querían hacer unos análisis de unos datos de germinación de unas plantas obtenidos en un estudio de campo. Por comodidad a la hora de recoger los datos, estos estaban organizados en un *estadillo* que a su vez se reflejaba en una tabla Excel como la de la figura:



	Germinación (%)					Masa de una planta (mg)			
	Stipa tenacissima		Brachypodium phoenicoides			Stipa tenacissima		Brachypodium phoenicoides	
Replica	Control	-0,307 MPa	Control	-0,307 MPa		Control	-0,307 MPa	Control	-0,307 MPa
1	NA	37	96	80		NA	0,7	1,6	1,1
2	51	45	88	80		1,0	1,0	1,8	1,3
3	40	60	88	68		1,4	1,5	2,7	1,2
4	48	48	88	80		1,7	1,0	2,1	0,9
5	54	51	80	92		1,4	1,4	2,1	0,8
6	31	49	80	88		2,1	1,5	1,3	1,1
7	34	51	84	72		1,3	1,1	1,9	0,7
8	57	34	80	72		1,7	0,5	1,8	0,8
9	31	40	88	80		1,6	1,4	1,1	1,3
10	26	63	88	48		1,1	1,3	1,3	1,3
11	34	65	64	84		0,5	1,3	1,1	0,9
12	37	57	64	72		1,1	1,5	1,3	1,3
13	43	29	76	68		1,2	1,2	1,5	1,3
14	60	46	84	84		1,6	1,3	1,6	1,1
15	43	34	76	84		1,4	1,6	1,3	1,4

  en este conjunto de datos ni filas ni columnas corresponden a observaciones ni variables de una manera limpia.

- **Ejercicio:** Descarga el fichero students3.csv, ábrelo con R y piensa qué problemas tiene esta tabla de datos.

Comentarios adicionales sobre tidyR.

- La librería *tidyR* se ha actualizado recientemente y cuando busques información en la red es inevitable que te cruces con los nombres de las funciones de la versión previa. En concreto `gather` era la predecesora de `pivot_longer` y `spread` hacía el trabajo que ahora hace `pivot_wider`.
- Aunque las dos clases de `pivot` son la parte central de *tidyR*, existen otras funciones auxiliares. En concreto queremos destacar `separate` y `unite`. Son especialmente útiles para trabajar con columnas que contienen varias variables agrupadas con algún formato. Por ejemplo, puede ser conveniente separar una columna con fechas como 1988-02-15 en tres columnas año, mes, día. Volveremos sobre ellas después de aprender sobre fechas y expresiones regulares.
- Es muy recomendable ver los esquemas gráficos que aparecen en el resumen de tidyR elaborado por RStudio para ver gráficamente el efecto de las operaciones `gather` y `spread`.

Un ejemplo con gather (obsolescente).

- La tabla de datos `USArrests` de la librería `datasets` comienza así:

```
           Murder Assault UrbanPop Rape
Alabama      13.2     236       58 21.2
Alaska       10.0     263       48 44.5
Arizona       8.1     294       80 31.0
Arkansas      8.8     190       50 19.5
California    9.0     276       91 40.6
Colorado      7.9     204       78 38.7
```

  Fijate en que hay tres columnas que en realidad contienen la variable *tipo de delito* (tasa por 100000 habitantes). Vamos a usar `gather` para crear una variable llamada `felony` a partir de esas tres columnas:

```
USArrests %>%
  gather("Murder", "Assault", "Rape",
  key = "Felony",
  value = "ratePer100K") %>%
  sample_n(4)
```

```
  UrbanPop Felony ratePer100K
1       80 Murder        12.7
2       45   Rape        12.8
3       80   Rape        22.9
4       66 Murder         2.7
```

- **Ejercicio:** comprueba el cambio de dimensiones de la tabla y reescribe este ejemplo con `pivot_longer`.

Ejemplo de spread (obsolescente)

- Vamos a usar los datos de `table2`, un ejemplo incluido con el `tidyverse` (se muestra el comienzo, la tabla tiene dimensiones 12, 4):

```
# A tibble: 4 x 4
  country      year type           count
  <chr>       <int> <chr>          <int>
1 Afghanistan  1999 cases            745
2 Afghanistan  1999 population 19987071
3 Afghanistan  2000 cases           2666
4 Afghanistan  2000 population 20595360
```

  Aplicamos

```
table2 %>%
  spread(key = "type", value = "count")
```

```
# A tibble: 6 x 4
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

  y ahora las variables de `cases` y `population` ya tienen columnas propias.

- **Ejercicio:** comprueba el cambio de dimensiones de la tabla y reescribe este ejemplo con `pivot_wider`

Fechas en R. Lubridate.

- Las fechas (y horas) son un tipo de dato muy frecuente pero a menudo difícil de gestionar, por los diferentes formatos que se usan. En el ecosistema del `tidyverse` disponemos de la librería `lubridate` para facilitar ese tipo de operaciones. Recomendamos la lectura del Capítulo 16 de *R for Data Science* o el Capítulo 8 de (Boehmke 2016). También la consulta del resumen de 'lubridate' elaborado por RStudio.
- Una de las operaciones básicas es la conversión de una fecha en formato texto al tipo de datos fecha que usa `lubridate`. Existen diversas funciones de conversión para los formatos más habituales. En este ejemplo, después de examinar el texto hemos optado por la función `dmy_hm` (de *day-month-year_hour_minute*):

```
library(lubridate)
fechaTexto = "21-07-1969 02:56UTC"
(fecha = dmy_hm(fechaTexto))
```

```
[1] "1969-07-21 02:56:00 UTC"
```

De la misma forma, existen funciones como `ydm`, `hms`, etc. para extraer la información necesaria de la mayoría de formatos de texto que encontrarás.

Acceso a las componentes de una fecha y operaciones con fechas.

- Una vez que disponemos de una fecha en el formato correcto, también podemos extraer de ella las partes que la componen:

```
day(fecha)
```

```
[1] 21
```

```
month(fecha)
```

```
[1] 7
```

```
hour(fecha)
```

```
[1] 2
```

- También es muy fácil hacer operaciones con fechas. Por ejemplo, ¿qué fecha se obtiene si sumamos 100 días a la fecha de ejemplo que venimos usando?

```
fecha + days(100)
```

```
[1] "1969-10-29 02:56:00 UTC"
```

¿Y si le sumamos 1000 horas y 42 minutos?

```
fecha + hours(1000)  + minutes(42)
```

```
[1] "1969-08-31 19:38:00 UTC"
```

El 29 de octubre de ese mismo año se envio el primer mensaje a través de Arpanet, el precursor de Internet. ¿Cuántos días separan ambas fechas?

```
fecha2 = dmy_hm("29-10-1969 00:00")
fecha2 - fecha
```

```
Time difference of 99.87778 days
```

Strings y regex con stringr

- La librería stringr es un componente del tidyverse que pemite trabajar de forma eficiente con variables de tipo character y que ofrece una sintaxis más homogénea que R base, que además está orientada al uso de *expresiones regulares*.
- Ver el Capítulo 14 de (R for Data Science, Wickham and Grolemund 2016). Y no dejes de descargarte la referencia rápida de stringr.
- Las funciones de stringr tienen nombres que empiezan siempre por str_. Por ejemplo, para ver el número de caracteres que componen un string:

```
palabras = c("Wahrheit", "ist", "Feuer", "und",
             "Wahrheit", "reden", "heißt", "leuchten", "und",
             "brennen")
str_count(palabras)
```

```
 [1] 8 3 5 3 8 5 5 8 3 7
```

- Hemos usado varias veces paste a lo largo del curso para combinar strings, pero con stringr se puede usar también str_c. Dos ejemplos:

```
str_c(palabras, collapse = " ")
```

```
[1] "Wahrheit ist Feuer und Wahrheit reden heißt leuchten und brennen"
```

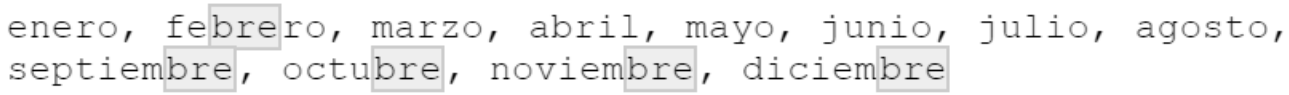
```
str_c("20",1:9, sep = "0")
```

```
[1] "2001" "2002" "2003" "2004" "2005" "2006" "2007" "2008"
[9] "2009"
```

Expresiones regulares

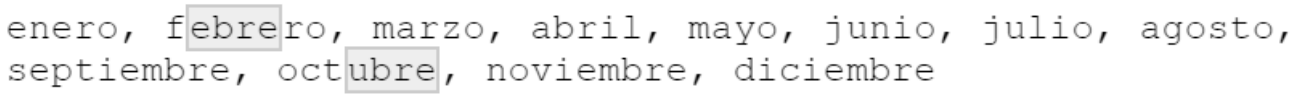
- Vamos a usar `str_view` para localizar todas la apariciones de un patrón en un string dado:

```
meses = "enero, febrero, marzo, abril, mayo, junio, julio,     agosto, septiembre, octubre, noviembre, dic
patron = "bre"
str_view_all(meses, patron) # requiere instalar htmlwidgets
```



- Vamos ahora a pedir que el patrón aparezca precedido por una vocal, creando una clase de caracteres con corchetes:

```
patron2 = "[aeiou]bre"
str_view_all(meses, patron2)
```



- Hay muchas más formas de definir patrones para seleccionar strings de tipos diversos. Por ejemplo para seleccionar exactamente dos consonantes (el guión selecciona caracteres consecutivos). Ejecuta este código y pregúntate qué ha pasado con `mbr`:

```
patron3 = "[b-df-hj-np-tv-z]{2}"
str_view_all(meses, patron3)
```

Usando las expresiones regulares con funciones de `stringr`

- Una vez que definimos un patrón con una expresión regular podemos usarlo en las funciones de `stringr`. Por ejemplo, podemos remplazarlo:

```
patron3 = "[b-df-hj-np-tv-z]{2}"
str_replace_all(meses, patron3, "##")
```

```
[1] "enero, fe##ero, ma##o, a##il, mayo, junio, julio, ago##o,  se##ie##re, o##u##e, novie##re, dicie##re"
```

Y opciones más sofisticadas. El siguiente patrón identifica dos caractéres que sean dígitos o letras seguidos (no espacios ni puntuación) e intercambia el orden de esos dos caracteres:

```
patron4 = "(\\w)(\\w)"
# str_view_all(meses, patron4) # Ejecuta para ver como funciona
str_replace_all(meses, patron4, "\\2\\1")
```

```
[1] "nereo, efrbreo, amzro, bairl, amoy, ujino, ujilo, gasoot,  estpeibmer, coutrbe, onivmerbe, idicmerbe"
```

- Las expresiones regulares (disponibles en muchos lenguajes de programación y en herramientas bash como grep) junto con las funciones de `stringr` forman una combinación muy potente para el trabajo con textos desde R. Por ejemplo, el vector `words` contiene una colección de 980 palabras comunes en inglés. ¿Cuáles contienen dos eses? Usando `stringr`, expresiones regulares y el operador %>% se tiene:

```
words %>% str_detect(pattern = "(s){2}") %>% words[.]
```

```
 [1] "across"   "address"  "associate" "assume"
 [5] "business" "class"    "cross"     "discuss"
 [9] "dress"    "express"  "glass"     "guess"
[13] "issue"    "less"     "miss"      "necessary"
[17] "pass"     "possible" "press"     "pressure"
[21] "process"  "unless"
```

Referencias

Boehmke, B. C. (2016). *Data Wrangling with R* (p. 508). Springer. https://doi.org/10.1007/978-3-319-45599-0

Wickham, H., & Grolemund, G. (2016). *R for data science: import, tidy, transform, visualize, and model data*. O'Reilly Media, Inc.