R para Procesamiento del Lenguaje Natural

Prácticas curso 2025-2026

Mariano Rico (mariano.rico@upm.es)

Documento creado el 07/11/2025

Tabla de contenidos

1 Las prácticas

1.1 Práctica Grupo 1

Análisis de ficheros de entrenamiento

El dataset **XLsum** es muy utilizado para hacer sistemas de generación automática de resúmenes.

Descarga manualmente el fichero `spanish_XLSum_v2.0.tar.bz2` que se encuentra en esta página web de HuggingFace. Ocupa 66 MB en disco. Este fichero está comprimido (extensión `tar.bz2`), y se puede descomprimir en linux con el comando `tar -xf archivo.tar.bz2` (en Windows puedes usar el programa `winRAR`). Una vez descomprimido (se elimina el original por defecto) tendrás una carpeta `spanish_XLSum_v2.0` con tres archivos que ocupan 249MB. Cada uno de estos tres ficheros tiene la extensión `.jsonl`.

Si abres con un editor el fichero `spanish_train.jsonl` verás que se compone de líneas que empiezan con `{` y terminan con `}`, con varios *campos* entre comillas dobles: `"id"`, `"url"`, `"title"`, `"summary"` y `"text"`. El formato de los ficheros `jsonl` se describe aquí, pero parece bastante intuituvo. Solo hay que tener cuidado con las comillas que aparecen en el interior de las cadenas de textos, porque van **escapadas** (`\"`).

1) Haced un programa `train_qcorpus.R` que lea el fichero `spanish_train.jsonl` y cree un corpus `quanteda` en el que cada documento sea el campo `"text"` de cada linea, y que tenga por docvars el resto de campos. Salva el objeto creado a un fichero (usa `saveRDS()`) llamado `spanish_train.qcorpus.rds`.
2) Haced un programa `train_stats.R` que lea el corpus `spanish_train.qcorpus.rds` (usa `readRDS()`) y calcule, para cada documento: (1) cuántas palabras contiene el campo `title`, (2) cuántas palabras contiene el campo `summary`. Muestra el resultado de estos (dos) cálculos mediante (dos) histogramas. Para el cálculo del número de palabras usa `spacyr`. Compara los resultados si usas `udpipe` en lugar de `spacyr` para identificar las palabras.
3) Haced un programa `train_analysis.R` que lea el corpus `spanish_train.qcorpus.rds` (usa `readRDS()`) y calcule, para cada documento: (1) cuántos verbos (convertidos a su forma en infinitivo) del campo `title` aparecen en el texto del documento, (2) cuántos verbos (convertidos a su forma en infinitivo) del campo `summary` aparecen en el texto del documento. Muestra el resultado de estos (dos) cálculos mediante (dos) histogramas.

1.2 Práctica Grupo 2

Trasteando con ficheros CONLL-U

Descarga el fichero CONLL-U que se encuentra aquí. Este fichero tiene 58.198 líneas y ocupa 3.44 MB en disco. Este fichero no es *puro CoNLL-U*, por lo que no puede ser leído por `udpipe_read_conllu()`. Hay que leerlo como un fichero tsv (*tab separated values*) con la función `read.table(file = 'el_fichero', sep = '\t', header = FALSE)`.

Haz los programas necesarios para responder a las siguientes preguntas:

1. ¿Cuántas frases se analizan en este fichero?
2. ¿Cuántos verbos distintos se utilizan?
   - Haz un histograma con las formas verbales encontradas.
3. Haz un programa `connlu_nlp.R` que reciba por argumentos el nombre del fichero collu de entrada y el nombre de un fichero donde se escribirán las respuestas a las preguntas anteriores.
4. Ejecuta el programa `connlu_nlp.R` con el fichero CONLL-U que se encuentra aquí y adjunta el fichero resultado

1.3 Práctica Grupo 3

Palabras encadenadas

En el juego de las palabras encadenadas, un jugador elegido al azar dice una palabra. El jugador que *es mano* (al que le toca jugar siguiendo un turno fijo) debe decir una palabra cuya primera silaba sea la última sílaba de la palabra anterior. Se producirán cadenas del estilo. `polo --> loro --> rocoso--> soso...`. Gana el jugador que diga una palabra que no pueda continuase, por ejemplo, `codorniz`, `flexo`, `malandrín`.

En R, el paquete `sylly` (en CRAN) soporta el español y tiene funciones para partir una palabra en sus sílabas. Si te pica la curiosidad, aquí hay una magnífica explicación de cómo funciona el (complejo) proceso de guionado de palabras en español.

Para este juego, las palabras válidas serán las que están en el diccionario que se describe a continuación, y la partición válida de sílabas será la que diga `sylly`.

Ejemplo de uso de `sylly`:

```
library(sylly)

available.sylly.lang() #Debe salir en la lista sylly.es

#Solo se debe instalar una vez
install.sylly.lang("es")

library (sylly.es)
sampleText <- paste (c("Yo fui a EGB", #Se ponía con tilde hasta los años 80
                       "El guion de la película era buena", #Sin tilde en 2010 (truhan, fie, liais...)
                       "Solo los lunes estoy solo", #También 2010
                       "Me duele el pie", #La extremidad
                       "Dile que píe"     #Verbo piar.
                      ), collapse = " "
                     )
palabras <- unlist(strsplit(sampleText, " "))
hyph.txt.es <- hyphen(palabras,
                      hyph.pattern="es",
                      min.length = 2, #Por defecto es 4 caracteres
                      quiet = TRUE #Quita progress bar
                     )
#OJO hyphen() devuelve un objeto complejo (S4), con campos lang, desc, y hyphen.
print(hyph.txt.es@hyphen[["word"]])
```

```
##  [1] "Yo"          "fui"         "a"           "EGB"         "El"
##  [6] "guion"       "de"          "la"          "pe-lí-cu-la" "era"
## [11] "bue-na"      "So-lo"       "los"         "lu-nes"      "es-toy"
## [16] "so-lo"       "Me"          "due-le"      "el"          "pie"
## [21] "Di-le"       "que"         "píe"
```

Hay palabras mal partidas, como `píe`, que debería tener dos sílabas según llevatilde.es.

1. Haz un programa `encadena.R` que, dada una palabra, haga una encadenación de palabras.
2. Haz un programa `encadenadas_interactivo.R` que juegue con el usuario, permitiéndole meter palabras (válidas, claro), y que intente buscar palabras *ganadoras*.
3. Haz un programa `supercadena.R` que elija palabras al azar del diccionario y calcule la longitud de la cadena más larga. (opcional) 4. ¿Podrías encontrar *anillos* (secuencia que te lleva a la palabra inicial)? ¿Cuál sería el anillo más grande?

Reglas de juego:

1. No se pueden usar palabras de menos de 5 letras
2. No se pueden usar nombres propios (ni personas, ni topónimos). En general, ninguna palabra del diccionario que empiece por mayúscula.

1.3.1 Un diccionario del español

Usad el diccionario que he dejado en Moodle. Este diccionario es el resultado de usar el programa `spell` para el español, sobre los diccionarios y reglas de afijos que se usan en OpenOffice y sistemas de código abierto como GNU. Este diccionario tiene las palabras de un diccionario habitual más las formas verbales (los verbos en todos sus tiempos), pero carece de muchos topónimos (e.g. Galicia, Extremadura, no están, pero sí tiene Madrid o Barcelona), y hay palabras repetidas (e.g. parís, del verbo parir, está dos veces). El diccionario es el fichero `dic_es.txt`, tiene 1.250.793 palabras con sus tildes y eñes bien puestas (UTF-8), y ocupa 14MB sin comprimir (como 3 fotos).

1.4 Práctica Grupo 4

Ayuda al poeta

En COES podemos encontrar una recopilación de enlaces a textos en español. Seleccionamos la antología de sonetos recopilada por José Benito.

Los sonetos tienen una estructura muy concreta: son 14 líneas (versos), cada verso tiene 11 sílabas (endecasílabos), con estructura fija en los 8 primeros versos (2 cuartetos ABBA ABBA) y estructuras más flexibles en los 6 últimos (2 tercetos) como (1) CDE CDE (2) CDE DCE o (3) CDC DCD.

1. Haz un programa `poeta.R` que obtenga por agumentos el nombre del fichero que contiene un soneto. El programa debe indicar si el soneto cumple la métrica. Utiliza el paquete `sylly` descrito en la práctica 2.
2. Haz un programa `poeta_interactivo.R` que juegue con el usuario, permitiéndole meter versos, y que el programa proponga palabras que cumplan la métrica.
3. Haz un programa `poeta_web.R` que analice una página web e identifique sonetos contenidos en la página. Prueba con la antología de sonetos recopilada por José Benito. (Opciona) Hay licencias métricas para relajar las restricciones, como la diéresis, el hiato, la sinéresis o la sinalefa. ¿Puedes incluirla en tus programas?

1.5 Práctica Grupo 5

El juego de cambiar una letra o permutar

Empieza indicando un tamaño de palabra. Por ejemplo: 5. Empieza con una palabra de 5 letras. Por ejemplo: `truco`. Los jugadores (dos o más) van, alternativamente, cambiando una única letra o permutando las letras. O se cambia una letra o se hace una permutación, no se puden hacer ambas cosas. La palabra resultante debe estar en el diccionario.

Ejemplo de partida: `trapo --> potra --> potro --> tropo --> tropa --> troca --> ...`

1. Haz un programa `camper.R` que, dada una palabra, haga una secuencia de palabras siguiendo el procedimiento descrito.
2. Haz un programa `camper_interactivo.R` que juegue con el usuario, permitiéndole meter palabras (válidas, claro), y que intente buscar palabras *ganadoras*.
3. Haz un programa `supercamper.R` que elija palabras al azar del diccionario y calcule la longitud de la secuencia más larga.
4. ¿Podrías encontrar *anillos* (secuencia que te lleva a la palabra inicial)? ¿Cuál sería el anillo más grande (por ejemplo, para "`truco`")?

1.6 Práctica Grupo 6

Buscador de enfermedades

Generar un buscador para el que, dada una noticia, documento o corpus, encuentre enfermedades clasificadas por tipos. Los tipos que se contemplan son: dolor, alteración, infección o cáncer. Se pueden plantear nuevos

tipos de interés. La clasificación de estas enfermedades se realiza mediante los afijos utilizados en medicina: `_algia` para dolor, `hipo_`/`hiper_` para alteraciones, `_itis` para infección y `_oma` para cáncer.

La búsqueda debe contemplar que dichas palabras sean nombres (no adjetivos que tengan dichos afijos) y que se pueda devolver el sintagma nominal completo que describa la enfermedad (e.g. fascitis necrosante). Por último, devolver dicha enfermedad en su forma invariable/lema.

(Opcional) Buscar la definición de esa enfermedad en un diccionario médico.

Datasets:

Articulos médicos en español aquí

Un ejemplo de API REST que pueden explotar para el diccionario es https://www.cun.es/diccionario-medico/terminos/abdominocentesis.

En general sería: `https://www.cun.es/diccionario-medico/terminos/{CONCEPTO}`.

1.7 Práctica Grupo 7

Crawler básico para THC

The Conversation (THC) es un publicador de contenidos científicos divulgativos. Tiene buena reputación por la calidad de sus artículos, en buena parte porque están escritos por científicos en lugar de por periodistas.

Si usas un navegador para visitar la página de entrada a THC en español, verás que al comienzo de la página hay una lista horizontal de categorías (Ciencia + Tecnología, Cultura, Economía, etc..). Pulsando en una de estas categorías obtienes un conjunto de artículos de esa categoría. Por ejemplo, en la categoría "Ciencia + Tecnología" hoy obtengo primero el artículo "El oro de la Prehistoria[...]". El contenido cambia periódicamente, así que si no obtienes este artículo no pasa nada, lo importante es el procedimiento.

Si guardamos el artículo (como .html) y lo abrimos con un editor de texto, podemos observar que el texto del artículo se encuentra entre `<p>` y `</p>` a partir de la línea donde aparece `itemprop="articleBody"`. Usando esta información, extrae el texto del artículo. Podrás observar que en el texto obtenido hay elementos como `<a ref="xxxx"... </a>`. Elimínalo, así como cualquier otro tipo de elemento de marcado HTML, para quedarte con el texto **limpio**.

1. Haz un programa `thc_extract.R` que lea un fichero .html y genere el texto del artículo.

Observa que al final del artículo hay un sección titulada "También le podría interesar", que tiene artículos relacionados.

2. Extiende el programa anterior para que también se muestren las URLs de los artículos relacionados. NOTA: estas URLs pueden ser difíciles de identificar. Si se te resisten, utiliza la extensión para Chrome `SingleFile`.

3. Haz un programa `thc_crawler.R` que tenga por argumentos: (1) la URL de un artículo, (2) el tiempo que esperará el programa antes de acceder a cada URL (para no saturar al servidor web), y (3) el número de artículos que se van a extraer a partir de este. La salida será un objeto corpus `quanteda` en el que cada documento sea un artículo y que tenga por docvars la URL. NOTA: Alternativamente, descarga los artículos de cada categoría y crea un corpus quanteda. Observa que cada categoría tiene una URL distinta pero constante; por ejemplo, la categoría "Ciencia + Tecnología" tiene la URL `https://theconversation.com/es/ciencia`, la categoría "Cultura" tiene la URL `https://theconversation.com/es/cultura`, etc.

Observa que al principio de cada artículo hay elementos `<meta name="xxx"... content="yyyy">`. De aquí puedes obtener la fecha de publicación (`<meta name="pubdate"...`), una lista de palabras clave (`<meta name="news_keywords"...`), la descripción (es un resumen breve), el autor y la institución.

4. Modifica el programa anterior para incluir esta información como docvars.

Analiza las siguientes cuestiones:

4.1. ¿Tu crawler puede escapar de "referencias circulares" (cuando en los artículos relacionados está el artículo inicial)? Si no lo hace, ¿cómo podría hacerlo?.

4.2. Deja funcionando tu crawler "toda una noche", con un tiempo entre artículos de 1 segundo, y un artículo inicial de la sección de "Ciencia + Tecnología". ¿Cuántos artículos has obtenido?. ¿Cuál es el autor con más publicaciones?. ¿Y la institución que más publica?. Haz una nube de palabras con las palabras clave de los artículos obtenidos. Muestra una gráfica (línea temporal) con las fechas de las publicaciones. Alternativamente, si no pudiste hacer el `crawler`, responde las mismas preguntas sobre el corpus `quanteda` que creaste en el apartado 3.

1.8 Práctica Grupo 8

La RAE (Real Academia de la Lengua) gestiona el CORPES, el corpus del Español del siglo XXI. El corpus no está disponible (sus textos están protegidos por derechos de autor), pero la aplicación web permite hacer consultas. Sabemos que el corpus se compone de unos 410 millones de *tokens*, y en su página publican listados de frecuencias.

Descarga 10 000 formas ortográficas más frecuentes. Verás que tiene tres columnas. La primera es la forma (ortográfica), lo que nosotros llamamos *token*, la segunda es la frecuencia absoluta (en realidad es el número de veces que aparece la forma en el corpus), y la tercera es la frecuencia normalizada (número de ocurencias por cada millón de tokens de corpus).

1. Comprueba que las formas siguen la ley de Zipf: haz una gráfica log-log donde se vea que los datos están sobre una recta. 1.1. ¿Hay formas repetidas? Muestras cuántas y cuáles son mostrando una lista que tenga en cada uno de sus elementos: la forma, la línea del fichero en la que se encuentra, su frecuencia absoluta, y para cada copia la línea y la frecuencia. 1.2. Número de duplicados no exactos. Por ejemplo, la forma `hoy` puede aparcer como `HOY` o como `Hoy`. Consideraremos `hoy` la forma básica y el resto formas duplicadas no exactas. ¿Cuántas hay? 1.3. ¿Hay alguna forma con caracteres no españoles? Haz una función que, para un texto dado, detecte si hay carateres no españoles. ¿Cuántas formas tienen carateres no españoles?

2. Haz lo mismo de antes pero con la Lista total de frecuencias.

1.9 Práctica Grupo 9

1) Usa el código penal en html de aquí: https://www.conceptosjuridicos.com/codigo-penal/ Tiene esta estructura:

- Título preliminar, con 9 artículos (del 1 al 9)
- Libro I (disposiciones generales)
  - Títulos I a VII
    - Cada título tiene artículos (del 10 al 137)
- Libro II (delitos y sus penas) *Títulos I al XXIV
  - Cada título tiene artículos (del 138 al 616 quater)

Hay que hacer un procesado general de la página html para sacar los links a los artículos (del 1 al 616 quater). Luego procesar cada página del artículo para sacar el texto limpio (con `\n`'s, por ejemplo, cada 50 caracteres, para que se pueda leer).

Como no podemos tener jerarquías de docs en quanteda, los nombres de docs deben tener esta forma (e.g. art. 38): `LIBRO I.Título III.Capítulo I.Sección III.Artículo 39` Además, el corpus debe tener docvars para `Libro`, `Título`, `Capítulo`, `Sección`, `Artículo` y `Contexto`.

De esta forma, cada documento quanteda tendrá el texto del artículo, así como una docvar con su contexto. Por ejemplo, para el Artículo 39 tendría este contexto (una sola cadena con los correspondientes `n`):

```
LIBRO I: Disposiciones generales sobre los delitos, las personas responsables, las penas, medidas de se
```

```
Título III: De las penas
Capítulo I: De las penas, sus clases y efectos
Sección III: De las penas privativas de derechos
```

El documento del artículo 39 sería así:

```
Son penas privativas de derechos:

a) La inhabilitación absoluta.

b) Las de inhabilitación especial para empleo o cargo público, profesión, oficio, industria o comercio,

c) La suspensión de empleo o cargo público.

...

j) La privación de la patria potestad.
```

2) Usa el corpus generado en el apartado anterior para mostrar un dendrograma de similaridad entre artículos. ¿Qué dos artículos son los más parecidos (menor distancia euclídea en la matriz de distancias)? Primero usa solo los documentos, y luego los contexto+documento (concatenado).

3) Usa el corpus generado en el apartado anterior para identificar entidades nombradas (sintagmas nominales) en los doumentos. Compara las entidades nombradas obtenidas con estas dos librerías:

- spacyr (por ejemplo, usando `spacy_extract_entity()`)
- udpipe (por ejemplo, usando `keywords_rake()`)

Muestra las 20 entidades nombradas más frecuentes identificadas por `spacyr` y por `udpipe`. ¿Cuántas son comunes a ambas librerías?

1.10 Práctica Grupo 10

1) Usa el código civil en html de aquí: https://www.conceptosjuridicos.com/codigo-civil/ Tiene esta estructura: Título preliminar, con 5 capítulos (I a V) y 15 artículos (del 1 al 15)

- Libro I
  - Títulos I a XII
    - Cada título tiene artículos (del 16 al 332)
- Libro II *Títulos I al VII
  - Cada título tiene capítulos, y estos, tienen artículos (del 333 al 608)
- Libro III *Títulos I al III
  - Cada título tiene capítulos, y estos, tienen artículos (del 609 al 1087)
- Libro IV *Títulos I al XVII
  - Cada título tiene capítulos, y estos, tienen artículos (del 1088 al 1975, y una disposición final)

Por ejemplo, este sería el "camino" hasta el artículo 530 del código civil:

LIBRO II.Título VII.Capítulo I.Sección I.Artículo 530

Lo interesante es tener todo el contexto del artículo. Algo como esto:

LIBRO II. DE LOS ANIMALES, DE LOS BIENES, DE LA PROPIEDAD Y DE SUS MODIFICACIONES Título VII: De las servidumbres Capítulo I: De las servidumbres en general Sección I: De las diferentes clases de servidumbres que pueden establecerse sobre las fincas Artículo 530

Haced un procesado general de la página html para sacar los links a los artículos. Luego procesar cada página del artículo para sacar el texto timpio (con `\n`'s, por ejemplo, cada 50 caracteres, para que se pueda leer).

Como no podemos tener jerarquías de docs en quanteda, los nombres de docs deben tener esta forma (eg. art 530): `LIBRO II.Título VII.Capítulo I.Sección I.Artículo 530` Además, el corpus debe tener docvars para `Libro`, `Título`, `Capítulo`, `Sección`, `Artículo` y `Contexto`.

De esta forma, cada documento quanteda tendrá el texto del artículo, así como una docvar con su contexto. Por ejemplo, para el Artículo 530 tendría este contexto (una sola cadena con los correspondientes `n`):

```
LIBRO II. DE LOS ANIMALES, DE LOS BIENES, DE LA PROPIEDAD Y DE SUS MODIFICACIONES
Título VII: De las servidumbres
Capítulo I: De las servidumbres en general
Sección I: De las diferentes clases de servidumbres que pueden establecerse sobre las fincas
```

El documento del artículo 530 sería así:

```
La servidumbre es un gravamen impuesto sobre un inmueble en beneficio de otro perteneciente a distinto

El inmueble a cuyo favor está constituida la servidumbre se llama predio dominante; el que la sufre, pr
```

2) Usa el corpus generado en el apartado anterior para mostrar un dendrograma de similaridad entre artículos. ¿Qué dos artículos son los más parecidos (menor distancia euclídea en la matriz de distancias)? Primero usa solo los documentos, y luego los contexto+documento (concatenado).

3) Usa el corpus generado en el apartado anterior para identificar entidades nombradas (sintagmas nominales) en los documentos. Compara las entidades nombradas obtenidas con estas dos librerías:

- spacyr (por ejemplo, usando `spacy_extract_entity()`)
- udpipe (por ejemplo, usando `keywords_rake()`)

Muestra las 20 entidades nombradas más frecuentes identificadas por `spacyr` y por `udpipe`. ¿Cuántas son comunes a ambas librerías?

2 Instrucciones de envío

Los miembros de cada grupo deben elegir a la persona que

1. enviará la *entrega* a Moodle dentro del plazo de entrega (**antes del 23 de diciembre**)
2. enviará un email a `mariano.rico@upm.es` con el asunto `Práctica PLN GradoCD grupoX` (reemplazando `X` por el número del grupo)

Esa *entrega* será un fichro `.zip` de nombre `prácticaGX` (reemplazando `X` por el número del grupo) que contendrá:

1. Un pdf con la memoria, generado a partir de un fichero .Rmd, que deberá contener:

- El número del grupo
- Los emails de los miembros del grupo
- Para cada pregunta de la práctica, el código fuente y el resultado. Si el resultado ocupa más de 20 líneas, solo se mostrarán las 5 primeras y las 5 últimas, y unos puntos suspensivos entre medias para indicar que hay más líneas. También una *explicación* de las partes más relevantes de cada programa, y las partes que más os han costado (indicando el motivo y la solución adoptada). Se valorará la claridad de las explicaciones, la claridad del código fuente, y la calidad de las pruebas realizadas al código.
- Una lista de tareas relizadas por cada miembro del grupo. Si una tarea se ha realizado entre varias personas, indicad el porcentaje de autoría.

2. EL fichero .Rmd con el que se generó el fichero pdf.

3. Una carpeta de nombre `fuentes` con el código fuente de los programas (al menos un fichero `.R` por pregunta).

4. Una carpeta de nombre `datos` con los datos usados por los programas.

5. Una carpeta de nombre `pruebas` con el código fuente de los programas usados para probar las funciones y/o los programas que responden a las preguntas.
6. Una carpeta de nombre `resultados` con los resultados (ficheros, figuras, etc.) generados por los programas.