PEC 2 - Desarrollo del trabajo - Fase 1

Endo-Mining: herramienta web para la búsqueda automatizada de genes potencialmente relacionados con la endometriosis a través de minería de textos

Jorge Vallejo Ortega

18/04/2021

Índice

1 Descripción del avance del proyecto

1.1 Grado de cumplimiento de los objetivos y resultados previstos en el plan de trabajo

Los objetivos planteados generales y específicos son:

1.1.1 Objetivos generales

1. Encontrar genes relacionados con la endometriosis aplicando técnicas de minería de textos.

1.1.2 Objetivos específicos

1. Desarrollar un script que permita realizar un procedimiento de minería de textos automáticamente, desde la recopilación de datos en bruto hasta la presentación de resultados.

2. Desarrollar una aplicación web implementando el script de minería de textos que resultó del objetivo anterior.

El **objetivo general** podríamos decir que está completo en un 60-70%. Hemos generado las rutinas necesarias para, a partir de palabras clave (en nuestro caso "endometriosis", aunque se pueden usar otras) y rangos de fechas, recuperar sumarios de publicaciones científicas de la base de datos PubMed. A partir de dichos sumarios, las mencionadas rutinas reconocen los genes que en ellos aparecen y los organizan tanto en forma de tabla de contingencias como en forma de diagrama de barras para mostrarlos al usuario.

Para completar el objetivo general en un 100% planeamos implementar las siguientes funcionalidades:

- Filtrado que mantenga fuera de la lista genes que hayan aparecido con una frecuencia estadísticamente no significativa y puedan ser falsos positivos.

- Ofrecer al usuario la descarga del listado de genes en forma de archivo CSV.

En cuanto a los **objetivos específicos**, consideramos que el desarrollo del script está completo en un 30-40% y la implementación en forma de aplicación web completa en un 20-30%.

En estos momentos el **script** es capaz de, a partir de palabras clave y un rango de fechas, recuperar información de la base de citas bibliográficas PubMed, generar un corpus de sumarios a partir de dicha información, extraer los símbolos HGNC[1] de los genes nombrados en el corpus y su frecuencia, y mostrar esta información en formato tabla y como gráfico de barras. Además calcula también la frecuencia de aparición de palabras en el corpus, mostrando las más frecuentes en forma de tabla y de gráfico de barras.

Para completar el script en un 100% según lo planeado necesitamos implementar lo siguiente:

- **Filtrado estadístico** que permita mostrar sólo aquellos genes que se encuentren sobrerrepresentados en la información recuperada, con respecto al total de citas en la base de datos PubMed.

- **Caracterización funcional** de la lista de genes según términos de ontología génica (GO, *gene ontology*).

- Representación de los resultados de frecuencia de genes y palabras en forma de **nubes de palabras**.

- **Visualización** de los resultado de caracterización funcional.

Por su parte, la **aplicación web** ha conseguido implementar las mismas funcionalidades que actualmente ofrece el script. Hemos podido publicar, en la plataforma de hosting Shinyapps.io, un prototipo esquemático (https://endo-mining.shinyapps.io/shinyapp/) que recoge input del usuario (palabras clave y rango de fechas), recupera información de la base de datos PubMed, la procesa y muestra los resultados en pantalla.

Para completar la aplicación web en un 100% según lo planeado necesitamos implementar lo siguiente:

[1] HUGO Gene Nomenclature Commitee (Comité de Nomenclatura de Genes de la HUGO), https://www.genenames.org/

- Las funciones que se incluirán próximamente en el script (filtrado estadístico, caracterización funcional, visualización de frecuencias en forma de nube de palabras y visualización de los resultados de caracterización funcional).

- **Mensajes de error** que sean de mayor utilidad al usuario (por ejemplo, sugiriendo aumentar o reducir el rango/especificidad de búsqueda cuando no se recuperen genes, o se recuperen demasiados resultados, respectivamente).

- Posibilidad de **descargar** resultados en formato CSV.

- Diseño web que combine **usabilidad** sencilla y **estética** agradable a la vista.

1.2 Justificación de los cambios

De momento, el único cambio con respecto a lo planeado es la inclusión en los resultados de las **frecuencias de las palabras**. Esto se ha hecho por dos razones. Por una parte, barajamos la posibilidad de incluir el **análisis de factores de riesgo no genéticos** como parte de los resultados de la minería de texto. Para realizar eso son necesarios los datos de frecuencia de palabras clave. A estas alturas del proyecto, aunque no hemos descartado completamente realizar dicho análisis de los factores de riesgo, consideramos poco probable llevarlo a cabo debido a restricciones en el tiempo disponible.

En segundo lugar, ofrecer al usuario una visualización de las palabras más frecuentes en el corpus da una **idea general e intuitiva** del material recuperado en la búsqueda. La recuperación y representación de esos datos, sin ser trivial, es bastante sencilla y consideramos que valía la pena implementarlo como una herramienta simple para ayudar a la comprensión y familiarización con los datos.

2 Relación de las actividades realizadas

2.1 Actividades previstas en el plan de trabajo

2.1.1 Tarea 1. Exploración de paquetes para minería de textos

Mediante búsquedas en internet y consultas en foros especializados hemos encontrado varios paquetes en R enfocados a minería de textos. De estos, algunos son generales y otros están diseñados alrededor del acceso a datos de PubMed y el NCBI. La siguiente tabla lista ambos tipos de paquetes:

PubMed/NCBI	General
easyPubMed	*lsa*
pubmed.mineR	*tidytext*
bibliometrix	*tm*
rentrez	
RISmed	

Después de consultar los manuales de referencia de los diferentes paquetes en la página del CRAN[2] decidí utilizar dos paquetes que se adaptan bien a las necesidades del proyecto: *easyPubMed* para la consulta en PubMed y descarga de citas, y *pubmed.mineR* para generar y manipular el corpus.

2.1.1.1 *easyPubMed*

El paquete *easyPubMed* es una interfaz que permite usar R para interactuar con las **Entrez Programming Utilities**, las API[3] públicas que permiten el acceso programático a las bases de datos Entrez (PubMed,

[2] *The Comprehensive R Archive*. Red de servidores web y ftp que almacenan documentación y código para R. Es la página de referencia para la descarga de R y de sus paquetes más comunes. En la página dedicada a cada paquete es posible consultar el manual de referencia respectivo en formato pdf.

[3] Siglas en inglés de interfaz de programación de aplicaciones (*application programming interface*), que se refiere al conjunto de funciones y protocolos que un programa ofrece para poder ser usado por otro programa diferente.

PMC, Gene, Nuccore y Protein). Las funciones de este paquete permiten la descarga por lotes de grandes volúmenes de registros, y el procesado básico del resultado de las búsquedas en PubMed (Fantini 2019).

La función principal que he usado de este paquete es `batch_pubmed_download()`, que permite realizar una búsqueda en PubMed y descargar los resultados en forma de ficheros. Los resultados se pueden descargar en formato XML o TXT en lotes de hasta 5.000 registros. Estos resultados en formato texto conforman los datos sobre los que posteriormente se aplicarán las funciones del paquete *pubmed.mineR*.

Como ejemplo he descargado los registros correspondientes al término de búsqueda "endometriosis" con fecha de publicación posterior a 2019:

```
query <- "endometriosis AND 2020/01/01:3000/12/31[dp]"

batch_pubmed_download(pubmed_query_string = query,
                      dest_dir = "intermediateData/",
                      dest_file_prefix = "last_endometriosis",
                      format = "abstract",
                      batch_size = 5000
                      )
```

```
[1] "PubMed data batch 1 / 1 downloaded..."
[1] "last_endometriosis01.txt"
```

El comportamiento de la función `batch_pubmed_download()` se puede ajustar mediante diferentes opciones, algunas de las cuales se pueden ver en este ejemplo. Mediante *pubmed_query_string* especificamos los términos con los que se efectuará la búsqueda en PubMed. Puede ser una búsqueda sencilla sólo con los términos de búsqueda o, como en el ejemplo, contener etiquetas (ej. [dp], fecha de publicación) y operadores booleanos (ej. AND).

El directorio de destino se puede elegir mediante la opción *dest_dir*, y la opción *dest_file_prefix* nos permite elegir el prefijo que se añadirá a cada uno de los ficheros creados para almacenar los resultados.

Con la opción *batch_size* podemos elegir el número máximo de registros incluidos en cada fichero (entre 1 y 5.000). Por último, la opción *format* nos da la oportunidad de elegir el formato en el que recibiremos los datos. El formato XML es rico en información y permite un procesado posterior más complejo del corpus primario (p.ej. subdividiéndolo por fecha, por autor, u otras opciones). Para este proyecto sin embargo he elegido una de las opciones en formato texto, ya que no necesitaremos realizar ese tipo de procesado del corpus y además genera ficheros que, conteniendo el mismo número de registros, ocupan menos espacio de memoria.

2.1.1.2 *pubmed.mineR*

El paquete *pubmed.mineR*, para el lenguaje **R**, se ha desarrollado específicamente para facilitar la minería de textos en el ámbito de la investigación biomédica; concretamente, aplicada a los sumarios (*abstracts*) de artículos incluidos en las citas de la base de datos PubMed. Para este fin incluye multitud de herramientas que implementan algoritmos de minería de textos, o que usan herramientas ya existentes en otros paquetes (Rani, S.Ramachandran, and Shah 2014). En esta sección comentaré, de entre las muchas funciones contenidas en el paquete, tan sólo aquellas que resultarán de utilidad en este proyecto.

En primer lugar, para constituir el **corpus primario**, utilizaremos la función `readabs()` sobre el archivo en formato texto que contiene los registros resultado de nuestra búsqueda previa. El resultado es un objeto tipo S4 con tres *slots* que contienen, respectivamente, la información referente la cita del artícula (nombre de la revista, fecha, número, ect.), el texto del sumario y el código PMID[4] del artículo.

[4]PubMed ID; número de identificación único asignado a cada una de las referencias incluidas en la base de datos PubMed.

```
last_abstracts <- readabs("intermediateData/last_endometriosis01.txt")
```

```
str(last_abstracts, vec.len = 1, nchar.max = 50)
```

```
## Formal class 'Abstracts' [package "pubmed.mineR"] with 3 slots
##   ..@ Journal : chr [1:2276] "1. Eur J Obstet Gynecol Reprod Bi"| __truncated__ ...
##   ..@ Abstract: chr [1:2276] "10.1016/j.ejogrb.2021.02.022. [Ep"| __truncated__ ...
##   ..@ PMID    : num [1:2276] 33756338 ...
```

Disponemos de dos importantes funciones para el **reconocimiento de entidades**. La función `gene_atomization()` reconoce los nombres de los genes (en su codificación como símbolo HGNC) y los extrae del del corpus primario además de calcular sus frecuencias.

```
last_genes <- gene_atomization(last_abstracts)
```

```
head(last_genes)
```

```
##      Gene_symbol    Genes
## [1,] "AMH"       "anti-Mullerian hormone"
## [2,] "ARID1A"    "AT-rich interaction domain 1A"
## [3,] "CPP"       "ceruloplasmin pseudogene"
## [4,] "KRAS"      "KRAS proto-oncogene, GTPase"
## [5,] "BDNF"      "brain derived neurotrophic factor"
## [6,] "MALAT1"    "metastasis associated lung adenocarcinoma transcript 1"
##      Freq
## [1,] "128"
## [2,] "58"
## [3,] "56"
## [4,] "32"
## [5,] "30"
## [6,] "26"
```

Por otro lado, la función `word_atomizations()` es más general. Disgrega el texto en palabras y las ordena según su frecuencia. Excluye los espacios, la puntuación y las palabras más comunes del idioma inglés.

```
last_words <- word_atomizations(last_abstracts)
```

```
head(last_words)
```

```
##               words Freq
## 13469 endometriosis 7236
## 32774         women 2943
## 24159      patients 2778
## 29603         study 2047
## 23866          pain 1560
## 27215       results 1382
```

Finalmente, la función `tdm_for_lsa()`, a partir de un vector de términos, encuentra la frecuencia de cada término en cada uno de los sumarios del corpus primario. Devuelve una **matriz documento-término** con las frecuencias de los términos buscados, en la que cada fila representa uno de los términos y cada columna representa un documento (en este caso, un sumario). Esta matriz se puede usar posteriormente para realizar un análisis semántico latente. Su utilidad para este proyecto es la exploración de factores de riesgo no genético.

```
tdm <- tdm_for_lsa(last_abstracts,
                   c("age", "gender", "woman", "women", "man",
                     "men", "smoking", "relationships", "relation"))
```

```
tdm[, 1:10]
```

```
##               [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## age              3    0    0    0    2    2    1    1    1     1
## gender           0    0    0    0    0    0    0    0    0     0
## woman            0    0    1    0    0    1    1    0    0     0
## women           10    0    0    0    3    0    0    0    0     3
##  man             0    0    0    0    0    0    0    0    0     0
##  men             0    0    0    0    0    0    0    0    0     0
## smoking          0    0    0    0    0    0    0    0    0     0
## relationships    0    1    0    0    0    0    0    0    0     0
## relation         0    1    0    0    0    0    0    0    0     0
```

2.1.2 Tarea 2. Búsqueda en PubMed para entender la construcción de la consulta

PubMed es un recurso en línea de acceso público y gratuito consistente en una base de datos - en continuo crecimiento - que incluye más de 32 millones de citas y abstracts de literatura biomédica, tanto artículos (*MEDLINE* y *PubMed Central*) como libros (*Bookshelf*). Esta base de datos - en línea desde 1996 - fue desarrollada y sigue siendo mantenida por el Centro Nacional para la Información Biotecnológica (*National Center for Biotechnology Information*, NCBI), que forma parte de la Biblioteca Nacional de Medicina de los E.E.U.U. (*U.S. National Library of Medicine*, NLM) de los Institutos Nacionales de Salud (*National Institutes of Health*, NIH). Esta base de datos está especializada en publicaciones centradas en campos científicos relacionados con la salud y la biomedicina (National Library of Medicine (2021a), National Library of Medicine (2021b)).

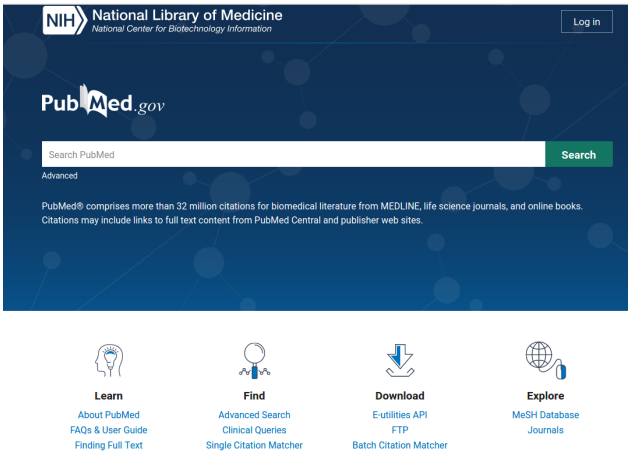

Figura 1: Página principal de PubMed. La barra de búsqueda destaca en medio de la imagen, indicando la finalidad principal de esta página.

Para realizar búsquedas es posible utilizar una serie de etiquetas con las que especificar si el término que hemos escrito debe buscarse como parte del título (etiqueta [TI]), el autor ([AU]), la revista ([TA]) o cualquier otro campo dentro de una larga lista que podemos consultar en la sección de ayuda de PubMed[5]. También es posible usar los operadores booleanos AND, OR y NOT. Sin embargo, no es necesario emplear las

[5] https://pubmed.ncbi.nlm.nih.gov/help/#search-tags

etiquetas ni los operadores, ya que el motor de búsqueda de la página puede crear por sí mismo búsquedas complejas a partir de sólo las palabras clave introducidas en el campo de búsqueda.

El algoritmo que construye las búsquedas a partir de nuestras palabras clave (*Automatic Term Mapping*) contrasta dichas palabras clave contra diferentes tablas de traducción de términos. En este orden: tabla de traducción de temas, tabla de traducción de revistas, índice de autores e índice de investigadores (colaboradores). Cuando se encuentra una coincidencia para el término o la frase, dicha coincidencia se añade a la búsqueda y no se continúa en la siguiente tabla de traducción.

La tabla de temas relaciona - entre otras cosas - las diferentes formas ortográficas del inglés americano y el británico, formas singulares y plurales, sinónimos, términos fuertemente relacionados, nombres de medicamentos genéricos y sus nombres comerciales, y el vocabulario controlado incluido en el tesauro MeSH (*Medical Subject Headings*).

La tabla de revistas contiene y relaciona el título completo de las revistas, sus abreviaciones y sus números ISSN.

El índice de autores y el índice de investigadores contienen el nombre, iniciales y nombre completo de los autores incluidos en la base de datos.

Primero de todo, realicé una búsqueda en PubMed (https://pubmed.ncbi.nlm.nih.gov/). La más básica posible y más general, sin ningún filtro, con la palabra clave "endometriosis". El resultado fueron 29,361 citas, desde 1927 hasta 2021.

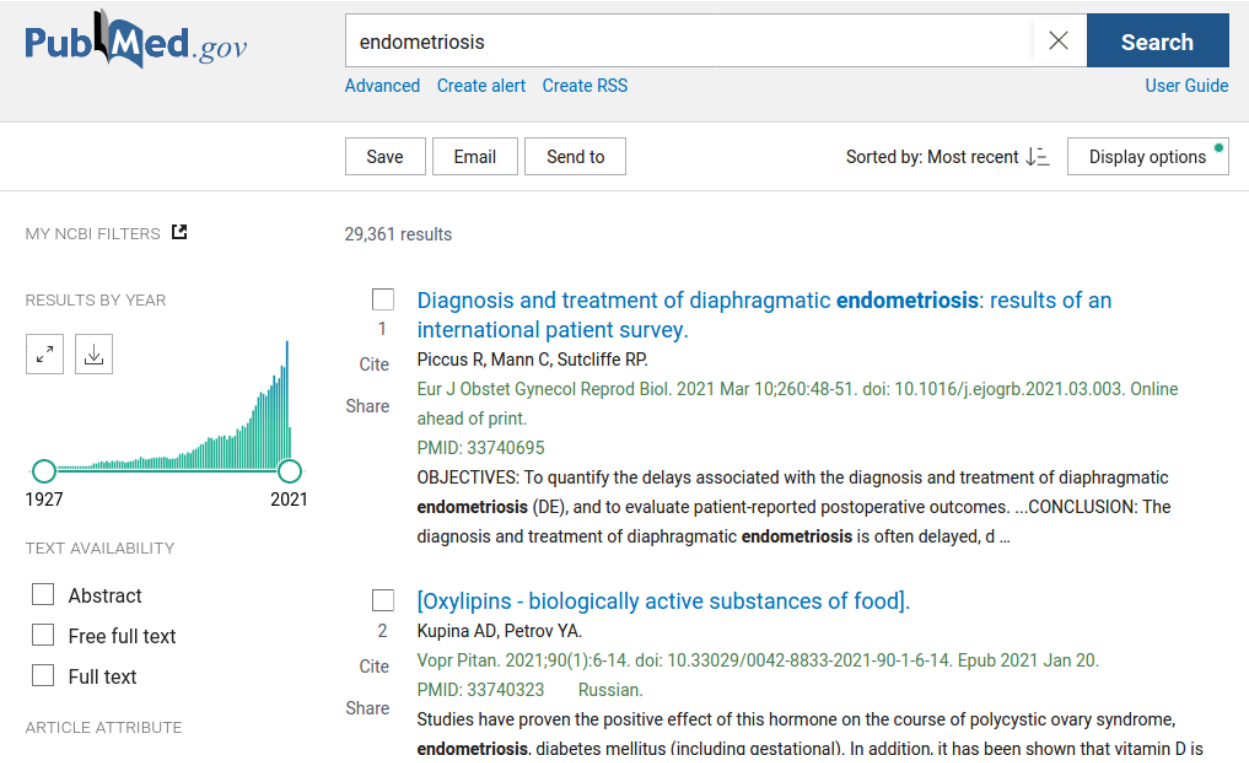

Figura 2: Página de resultados para el término 'endometriosis'. En la parte central se muestran las citas recuperadas por el algoritmo. En el margen izquierdo se nos ofrecen filtros interactivos para refinar la búsqueda.

A través del enlace 'Advanced', que se encuentra en la zona superior izquierda, podemos acceder a un constructor de búsquedas que nos facilita hacer búsquedas avanzadas sin necesidad de conocer todas las etiquetas disponibles. En esa misma página podemos consultar nuestro historial de búsquedas recientes y, en éste, cómo el constructor de búsquedas ha traducido nuestra búsqueda simple ('endometriosis') utilizando las tablas de traducción:

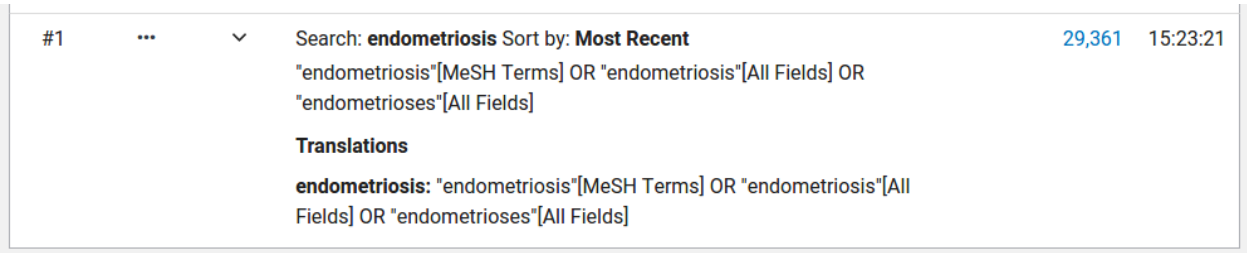

Figura 3: Detalle del historial de búsqueda. De izquierda a derecha muestra la siguiente información: número de la búsqueda en orden cronológico, los términos de búsqueda entrados por el usuario y los términos a los que el algoritmo de mapeo automático los ha traducido, el número de resultados y finalmente la hora a la que se solicitó la búsqueda.

Asimismo si usamos los filtros para, por ejemplo, limitar la búsqueda a los artículos publicados durante los últimos diez años, también podemos consultar la estructura de dicha búsqueda:

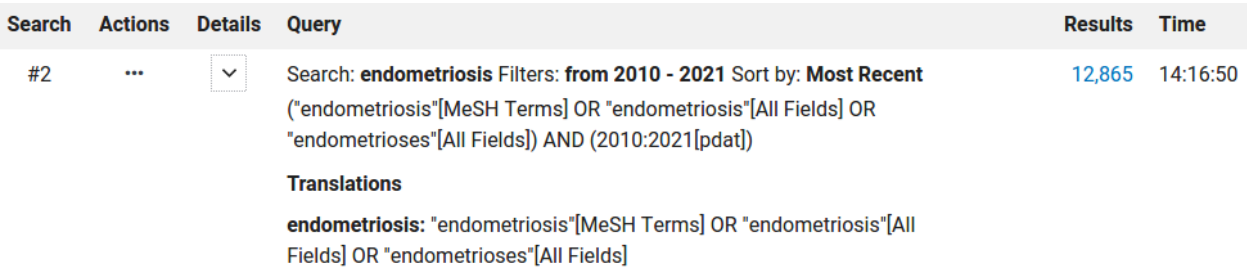

Figura 4: Detalle del historial de búsqueda. Misma búsqueda del ejemplo anterior, filtrada para obtener como resultado citas de entre los años 2010 y 2021

2.1.3 Tarea 3. Extracción de abstracts con R a partir de palabra clave

Esta tarea es previa y necesaria a la generación del corpus primario, que consistirá en todos los sumarios recuperados de la base de datos PubMed utilizando como término de búsqueda el término "endometriosis", sin acotar por fecha de publicación.

Como se señala anteriormente, la función `batch_pubmed_download()` es la adecuada para enviar la búsqueda a la base de datos, recuperar registros y guardarlos en formato texto repartidos en varios ficheros:

```
batch_pubmed_download("endometriosis",
                      dest_dir = "data/",
                      dest_file_prefix = "total_endometriosis",
                      format = "abstract",
                      batch_size = 5000
                      )
```

Previamente, al hacer la búsqueda de prueba en la página de PubMed ya había averiguado que se recuperan casi 30.000 registros. Lo que significa que el tiempo de descarga es relativamente largo y, al haber especificado que se generaría un archivo por cada 5.000 registros, obtenemos como resultado 6 archivos de texto conteniendo todos los registros:

```
[1] "PubMed data batch 1 / 6 downloaded..."
[1] "PubMed data batch 2 / 6 downloaded..."
[1] "PubMed data batch 3 / 6 downloaded..."
[1] "PubMed data batch 4 / 6 downloaded..."
[1] "PubMed data batch 5 / 6 downloaded..."
[1] "PubMed data batch 6 / 6 downloaded..."
[1] "total_endometriosis01.txt" "total_endometriosis02.txt" "total_endometriosis03.txt" "total_endometriosis04.txt"
[5] "total_endometriosis05.txt" "total_endometriosis06.txt"
```

Consolidé los registros en un único archivo creando un nuevo fichero de texto, y copiando en él todos los registros que están repartidos entre los ficheros anteriores:

```
# List of files to be added together
files_list <- list.files(path = "data/",
                         pattern = "total",
                         full.names = TRUE) # include path
# Create new file
out_file <- file(description = "intermediateData/todos.txt",
                 open = "w")
# Read each downloaded file and write into final file
for (i in files_list){
  x <- readLines(i)
  writeLines(x, out_file)
}

close(out_file)
```

El resultado es un fichero llamado `todos.txt` que contiene todos los registros resultado de la búsqueda, en formato TXT. Este paso de consolidación formará parte del script final y se aplicará a cada búsqueda.

2.1.4 Tarea 4. Preprocesado de los sumarios

Esta tarea se compone de dos partes: generación del corpus primario, y desagregación de los sumarios en las palabras que los componen para calcular la frecuencia de aparición de cada palabra.

A partir de la información contenida en el fichero con los resultados de búsqueda consolidados, generé un objeto de clase 'Abstracts' que puede ser manipulado por otras funciones del paquete *pubmed.mineR*:

```
# Generate the object
abstracts <- readabs("intermediateData/todos.txt")
```

Si examinamos la estructura del objeto:

```
## Formal class 'Abstracts' [package "pubmed.mineR"] with 3 slots
##   ..@ Journal : chr [1:29370] "1. Eur J Obstet Gynecol Reprod Bi"| __truncated__ ...
##   ..@ Abstract: chr [1:29370] "10.1016/j.ejogrb.2021.02.022. [Ep"| __truncated__ ...
##   ..@ PMID    : num [1:29370] 33756338 ...
```

Vemos que consta de 3 *slots*, dos de ellos almacenando datos de tipo cadena de caracteres (referencia del artículo y su sumario, respectivamente), y un último *slot* de tipo numérico almacenando el código PMID. Es a la información contenida en este objeto a la que aplicaremos los métodos de minería de textos.

Para desglosar los sumarios en las palabras que los componen, y registrar la frecuencia de cada palabra, usé la función `word_atomizations()` del paquete *pubmed.mineR*. Ésta recopila las palabras del texto y calcula la frecuencia de cada una sin tener en cuenta espacios, signos de puntuación ni palabras muy comunes en inglés.

```
words <- word_atomizations(abstracts)
```

Tabla 2: Las diez palabras más frecuentes en el corpus primario.

Palabras	Frecuencia
endometriosis	66 668
patients	29 232
women	27 229
study	16 110
treatment	15 121
ovarian	13 838
endometrial	12 306
results	12 246
pain	12 019
group	11 417

Entre las palabras más frecuentes encontramos, naturalmente, la propia palabra clave que hemos usado en la búsqueda (*endometriosis*), términos relacionados con investigación o tratamiento (*patients, study, treatment, results, group*), con la biología de este trastorno (*women, ovarian, endometrial*) y el síntoma más común (*pain*). El listado completo de palabras, con sus respectivas frecuencias, se puede descargar como fichero de texto desde este enlace.

2.1.5 Tarea 5. Extracción de genes

Una de las maneras de representar la información contenida en un texto es mediante la extracción de **entidades con nombre** como son organizaciones, personas o lugares. En el caso de este proyecto, las entidades de interés son los **genes**.

Partiendo de la hipótesis de que los genes que aparecen en los sumarios de artículos acerca de la endometriosis son importantes para este trastorno, extraje un listado de los mismos. Más adelante, este listado servirá para recuperar información a partir de los términos de ontología génica que estos genes tienen en común.

Para realizar la extracción usé la función `gene_atomization()` del paquete *pubmed.minerR* de R. Esta función reconoce, y recupera de los sumarios, los símbolos aprobados por el HGNC[6] para representar genes concretos. Esta función devuelve el símbolo del gen, su nombre largo y su frecuencia en el corpus. En la tabla siguiente vemos una muestra con los primeros diez genes más frecuentes. El listado completo puede descargarse desde este enlace.

```
genes <- gene_atomization(abstracts)
```

Tabla 3: Los diez genes más frecuentes en el corpus primario.

Símbolo	Nombre largo	Frecuencia
AMH	anti-Mullerian hormone	721
CPP	ceruloplasmin pseudogene	445
ARID1A	AT-rich interaction domain 1A	273
PIP	prolactin induced protein	210
PTEN	phosphatase and tensin homolog	209
HOXA10	homeobox A10	194
EGF	epidermal growth factor	158
MIF	macrophage migration inhibitory factor	156

[6]HUGO Gene Nomenclature Commitee (Comité de Nomenclatura de Genes de la HUGO), https://www.genenames.org/

Símbolo	Nombre largo	Frecuencia
GSTM1	glutathione S-transferase mu 1	154
NGF	nerve growth factor	154

2.1.6 Tarea 6. Filtrado estadístico de los genes

En el momento de entregar este informe, la implementación del filtrado no ha sido completada. La razón es que subestimé el tiempo necesario para hacerlo. Como resultado, esta tarea me ha consumido mucho más tiempo del necesario, incluso consumiendo tiempo que debería haber sido dedicado a otras tareas. En este apartado explico la tarea, cuánto de la misma he podido completar y qué falta por hacer.

El razonamiento que justifica la extracción y análisis de los genes contenidos en el corpus, es que dichos genes están relacionados con la *endometriosis* (nuestro término de búsqueda) con **más probabilidad** que los genes contenidos en cualquier muestra de sumarios recogidos al azar de la misma base de datos (PubMed). Dicho de otra forma, trabajamos bajo la hipótesis de que los genes presentes en el corpus están **sobrerrepresentados** en el propio corpus con respecto a su frecuencia en el total de la base de datos.

Podemos utilizar esta hipótesis para, a partir de la lista de todos los genes presentes en el corpus, excluir aquellos que no estén sobrerrepresentados y que, por tanto, es menos probable que estén relacionados con las palabras clave de búsqueda (*endometriosis* en este caso).

Para cada uno de los genes que hemos recuperado del corpus se realizará un contraste entre las siguientes hipótesis:

- Hipótesis nula: la frecuencia del gen en el corpus es igual a la frecuencia del gen en la base de datos.
- Hipótesis alternativa: la frecuencia del gen en el corpus es mayor a la frecuencia del gen en la base de datos.

$$H_0 : \frac{k}{n} = \frac{K}{N}$$

$$H_A : \frac{k}{n} > \frac{K}{N}$$

k: número de veces que el gen aparece en el corpus.
n: número de citas en el corpus (tamaño del corpus).
K: número de veces que el gen aparece en la base de datos.
N: número total de citas en la base de datos.

Este tipo de contrastes de hipótesis puede hacerse utilizando la **distribución hipergeométrica**, que se usa para modelar el muestreo *sin* reposición a partir de una población finita(Liu and Zhao 2016). Su función de probabilidad es:

$$Pr(X = k) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}$$

En este caso la muestra son las citas incluidas en el corpus, y la población de origen son todas las citas incluidas en PubMed. Para este fin podemos usar la función `phyper()` de R {stats}, que calcula la probabilidad de, por azar, obtener q éxitos (citas conteniendo el gen) o menos en una muestra de tamaño k, cuando la población de origen está compuesta por m éxitos (total de citas conteniendo el gen) y n fracasos (total de citas que *no* contienen el gen).

Para contrastar la hipótesis de que un gen esté sobrerrepresentado en la muestra necesitamos la probabilidad de obtener ***q* éxitos o más**, que calcularemos mediante esta fórmula:

```
1 - phyper(i, m, n, k)
```

Siendo i el número de éxitos menos uno $(q-1)$.

Debido a que se realizarán múltiples contrastes de hipótesis (uno por cada gen), para controlar el error de tipo I se aplicará la corrección de Bonferroni al límite de significatividad estadística: $\alpha_A = \alpha/g$. Donde α es el límite de significatividad estadística para un único contraste (p.ej. 0.05) y g representa el número de genes a contrastar. Si, por ejemplo, el número de genes en el corpus fuera de 700, el límite de significatividad estadística tras la corrección sería de 0.05/700 = 7e-5.

Para obtener las variables referentes al total de la población (m y n), descargué los resultados pre-calculados de minería de texto correspondientes a la base completa de PubMed desde el servidor ftp de PubTator[7] (con fecha del 26/03/2021). El archivo gene2pubtatorcentral.gz es un documento TSV que contiene información de todos los genes nombrados en los sumarios de PubMed, recuperada mediante GNormPlus. Éste archivo contiene más de 53 millones de filas divididas en 5 columnas que muestran la siguiente información:

1. PMID: Identificador único en PubMed del sumario del que se ha recuperado el gen.

2. Type: Tipo de entidad. En este caso la entidad es siempre "Gene".

3. Concept ID: Identificador único de la entidad (en este caso el identificador Gene ID de la base de datos NCBI).

4. Mentions: El símbolo HGNC o el nombre del gen que aparece en el sumario.

5. Resources: Recurso del que se ha extraido la información (e.g. GNormPlus).

```
9479000 Gene    12514           gene2pubmed
6814000 Gene    1351    VIII    GNormPlus
3220000 Gene    5328            MESH
17438000        Gene    432248  Xbra    GNormPlus|MESH
17438000        Gene    378587  eFGF    GNormPlus
17438000        Gene    380196  CHD4    GNormPlus|MESH
17438000        Gene    108701318       Sip1    GNormPlus|MESH
17438000        Gene    386595          generifs_basic
8395000 Gene    7391    USF     GNormPlus
8395000 Gene    4654    MyoD-R|MyoD     GNormPlus
```

Figura 5: Muestra de varios registros del documento gene2pubtatorcentral.gz.

A partir de este archivo he obtenido una tabla de contingencia que contiene la frecuencia de cada gen (representado por su Gene ID). Los datos de esta tabla son los que se usarán para calcular la relación entre la frecuencia del gen en la base de datos (K/N) y la frecuencia del gen en los resultados de la búsqueda (k/n).

Tabla 4: Frecuencia en la base de datos de seis genes elegidos al azar

GeneID	Frecuencia
20493484	1
4314	23219
23765474	1

[7]PubTator Central (PTC, https://www.ncbi.nlm.nih.gov/research/pubtator/) es un servicio web para la exploración y recuperación de anotaciones de bioconceptos en artículos biomédicos. Utiliza sistemas de minería de textos para recopilar información acerca de genes y proteinas, variantes genéticas, enfermedades, productos químicos, especies y líneas celulares de las bases de datos PubMed, PMC y AMC. Mantiene un repositorio FTP de anotaciones agregadas descargables en ficheros con formato TSV. Las anotaciones se actualizan con perioricidad mensual.

GeneID	Frecuencia
807900	2
929154	1
436880	1

La obtención de esta tabla ha representado cierta dificultad debido a que cada fila puede contener más de un identificador de gen (Gene ID), y por tanto hay que limpiar y reorganizar los datos antes de generar la tabla de contingencia. Además, operar con más de cincuenta millones de filas en mi ordenador portátil ha sido un pequeño reto. El código que he usado para llevarlo a cabo se puede consultar en el apéndice C.

Llegado a este punto tuve que dejar la tarea de lado, ya que me había ocupado mucho más tiempo del previsto en el plan de trabajo. Para completar esta tarea e implementarla todavía es necesario hacer lo siguiente:

1. Debido a que la frecuencia de genes en la base de datos está referida al Gene ID y no al símbolo HGNC (debido a que el Gene ID es único, pero existen varios símbolos para cada gen), en lugar de utilizar la función `gene_atomization()` será necesario utilizar la función `pubtator_function()` para recuperar tanto el Gene ID (para usarlo en el test estadístico) como el símbolo HGNC para mostrar los resultados al usuario.

2. A partir de los resultados de la función `pubtator_function()` generaremos dos tablas: una contendrá PMID y GeneID, y será la que usaremos para el test. La otra tabla contendrá el GeneID y el símbolo HGNC correspondiente, y la usaremos para anotar los resultados del test.

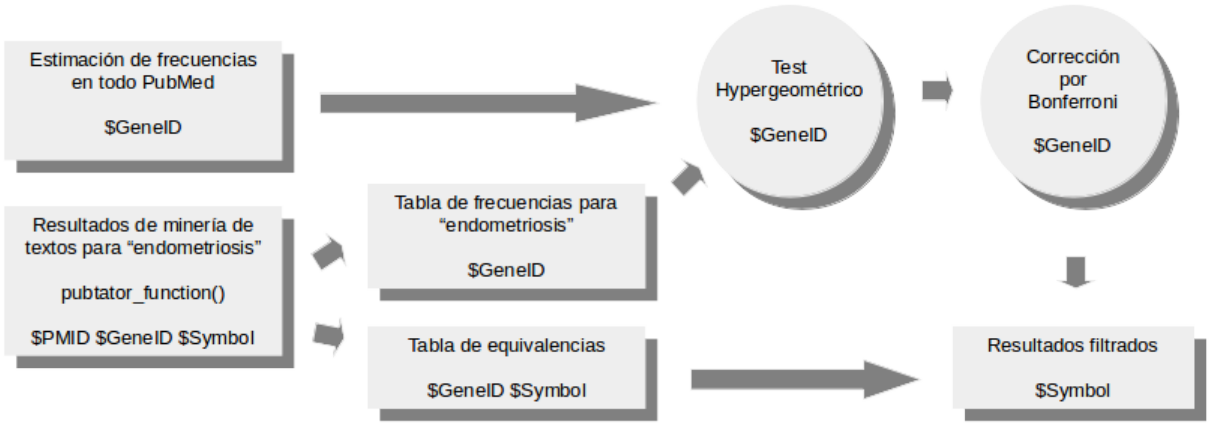

Figura 6: Diagrama del filtrado de los resultados de la minería de texto mediante el test hipergeométrico.

2.1.7 Tarea 7. Aprendizaje de desarrollo de aplicaciones con Shiny

El alcance de esta tarea es un tanto vago, ya que por sí misma no está asociada a un objetivo concreto sino a un proceso, el de aprendizaje. A efectos de entrega de este informe, he interpretado esta tarea como "aprender lo suficiente como para publicar una aplicación Shiny en la que estén implementadas las funcionalidades de minería de textos, análisis y visualización que estoy desarrollando para este proyecto".

Esa interpretación es meramente para especificar un objetivo que pueda alcanzar de forma objetiva ya que, aunque lo he conseguido, seguiré aprendiendo y mejorando en esta habilidad mientras continúe implementando funcionalidades en la aplicación resultado de este proyecto, y mientras genere nuevas aplicaciones en el futuro.

2.1.7.1 Qué es Shiny

Shiny es un paquete para el lenguaje de programación **R** que permite diseñar, **páginas web interactivas usando el lenguaje de programación R**. Las funciones incluidas en el paquete, traducen las órdenes en R al código HTML, CSS y JavaScript que proporcionarán estructura, estilo y funcionalidad a la aplicación.

Estas aplicaciones - programas - web pueden formar parte de una página web propiamente dicha, o estar incluidas documentos escritos en R Markdown. Es posible, aunque no necesario, complementar estas aplicaciones con lenguaje HTML, páginas de estilo CSS y con scripts escritos en JavaScript al igual que ocurre en las páginas web tradicionales (RStudio 2021, Wickham (2021)).

Uno de sus puntos fuertes es la inclusión de *widgets* preprogramados, pequeñas aplicaciones para la entrada de datos y la comunicación de resultados, que pueden ser añadidas a la aplicación web mayor con sólo un mínimo de código.

2.1.7.2 Estructura del código

Todo programa Shiny tiene dos componentes principales: la **UI** (*user interface*, interfaz de usuario), que define el aspecto del programa y cómo el usuario interactúa con él; y la **función de servidor**, que establece cómo funciona el programa. Shiny utiliza un paradigma conocido como **programación reactiva**, significando que los resultados que se muestran al usuario cambian se actualizan automáticamente cuando cambian los inputs. Esto se logra con un tercer componente de los programas Shiny, las **expresiones reactivas**.

Esquema básico de una aplicación Shiny:

```
library(shiny)

ui <- fluidPage(
  # Interfaz de usuario
)

server <- function(input, output, session) {
  # Función de servidor
}

shinyApp(ui, server)
  # Construye e inicia la aplicación
```

La UI incluye principalmente **inputs** y **outputs**. Los *inputs* especifican controles con los que el usuario puede introducir información que será usada por las funciones incluidas en la aplicación para la búsqueda de información, el análisis y la visualización. Todas las funciones de input deben incluir el argumento `inputId`, generando un identificador que se usa para conectar la interfaz de usuario con las funciones de servidor. Además muchas funciones de input tienen un segundo parámetro, `label`, que especifica el texto que se mostrará en el control presente en la interfaz.

Un ejemplo de función input que especifica la existencia en la interfaz de usuario de un campo para la introducción de texto:

```
# User interface
ui <- fluidPage(
  # Enter keywords
  textInput(inputId = "keywords",
            label = "Palabras clave",
            value = "endometriosis",
            placeholder = "endometriosis")
)
```

En el ejemplo de la función `textInput()` los argumentos tienen las siguientes funciones: *inputId* genera un identificador (input$keywords) que será usado en las funciones de servidor para recuperar las palabras introducidas por el usuario; *label* genera una etiqueta, visible para el usuario en la interfaz, que identifica

el control; *value* es el valor que adopta este input por defecto, sin que el usuario haya introducido nada; y *placeholder* es la palabra que aparece en el campo de entrada de texto como un ejemplo para mostrar al usuario el tipo de información que puede introducir.

Tabla 5: Funciones de input y su utilidad

Función input	**Tipo de input**
`actionButton()`	Botón para pulsar
`actionLink()`	Enlace para pulsar
`checkboxGroupInput()`	Elección múltiple (casillas)
`checkboxInput()`	Casilla de verificación
`dateInput()`	Introducción de una fecha
`dateRangeInput()`	Introducción de un rango de fechas
`fileInput()`	Introducción de un archivo (upload)
`numericInput()`	Introducción de un valor numérico
`passwordInput()`	Introducción de texto (ocultando caracteres)
`radioButtons()`	Botones de selección
`selectInput()`	Lista de opciones
`sliderInput()`	Control deslizante
`submitButton()`	Poner en práctica
`textInput()`	Introducción de texto

Figura 7: Ejemplo de interfaz de usuario con tres controles de input: texto, rango de fechas y botón

Los **outputs** especifican dónde aparecerá la información devuelta por las funciones del servidor, y en qué formato. Al igual que los *inputs*, cada *output* recibe un identificador único para que las funciones de servidor puedan enviar sus resultados a un *output* concreto.

Ejemplo de código para output de texto. Su identificador único es `n_archivos`, lo que significa que mostrará información en formato texto de una función servidor que envía sus resultados a `output$n_archivos`:

```r
# User interface
ui <- fluidPage(
  # Display number of retrieved results
  textOutput("n_archivos")
)

# App behaviour
server <- function(input, output, session){
 # Muestra la cantidad de citas recuperadas
  output$n_archivos <- renderText({
    paste0("Nº de citas recuperadas: ",
    length(pubmed_results()@PMID))
    })
}
```

Cada función *output* está emparejada con una función *render*. Este último tipo de funciones hacen dos cosas: i) Preparan un **contexto reactivo** que rastrea automáticamente los valores de los *inputs* que usa el *output* con el que está emparejada la función *render*, y ii) convierte los resultados del código R en el código HTML que servirá para mostrar dichos resultados en la página web.

Tabla 6: Relación entre funciones *render* y sus correspondientes funciones *output*.

Funciones *render*	**Funciones *output***	**Formato**
`DT::renderDataTable()`	`dataTableOutput()`	Tablas
`renderImage()`	`imageOutput()`	Imágenes
`renderPlot()`	`plotOutput()`	Gráficas
`renderPrint()`	`verbatimTextOutput()`	Texto
`renderTable()`	`tableOutput()`	Tablas
`renderText()`	`textOutput()`	Texto
`renderUI()`	`uiOutput()` & `htmlOutput()`	HTML

Hay tres tipos principales de *outputs*, que se corresponden con los tres formatos más comunes que se incluyen en cualquier informe: texto, tablas y gráficas.

2.1.7.3 Reactividad

El funcionamiento del código shiny se basa en la **programación reactiva**, que consiste en especificar relaciones de dependencia entre *inputs* y *outputs* de forma que, cuando cambia el valor de un *input* (ej. cuando introducimos un nuevo rango de fechas en nuestra búsqueda) todos los *outputs* relacionados con ese valor se actualizan automáticamente.

Esas relaciones de dependencia se establecen a través de las **expresiones reactivas**, como `renderText()` o `reactive()`, que leen los *inputs* cuando estos cambian y generan los nuevos valores de los *outputs* correspondientes. Se parecen un poco a las funciones en R base, en el sentido de que permiten evitar la duplicación de código. La idea principal es que no es necesario especificar en el código cuándo actualizar los *outputs*, sino que estos se actualizan automáticamente cuando resulta necesario.

Al conjunto de relaciones de dependencia entre *inputs* y *outputs* se le conoce como **gráfico reactivo**, y es el que determina el orden en que se ejecuta el código (mientras que en R normalmente el orden de ejecución está determinado por el orden de las líneas de código).

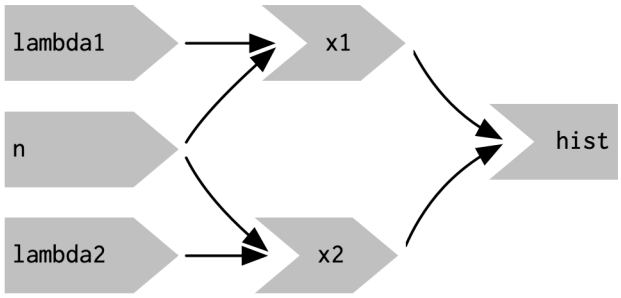

Figura 8: Esquema de gráfico reactivo representando - de derecha a izquierda - tres inputs, dos expresiones reactivas y un output. Extraído de Wickham (2021).

2.1.7.4 Publicación de aplicaciones Shiny en la web

Existen dos métodos principales para publicar en la web aplicaciones diseñadas con Shiny: en un servidor propio en el que se haya instalado el software **Shiny Server**, o en el servicio **Shinyapps.io** mantenido por la empresa RStudio (desarrolladora de Shiny). En éste último caso se puede elegir entre opciones que ofrecen mayor o menor funcionalidad dependiendo del precio (incluyendo una opción gratuita con funcionalidad mínima, pero ideal para un proyecto menor como éste).

El código mínimo necesario que tiene que enviarse al servidor es el correspondiente a la interfaz de usuario (`ui`) y el correspondiente a las funciones de servidor (`server`). Este código puede estar recogido en un único fichero llamado `app.R` o dividido en dos ficheros llamados respectivamente `ui.R` y `server.R`, que tendrán que estar almacenados en directorios separados.

El código correspondiente a funciones diseñadas por el desarrollador puede también estar incluido al principio del fichero `app.R`, o en uno o más ficheros independientes que pueden ser llamados desde el código de la aplicación. En mi caso, como el código de la aplicación para este proyecto todavía es muy simple, lo mantengo todo todavía en un único fichero `app.R`.

2.2 Actividades no previstas y realizadas

Las tareas no previstas en realidad son dos tareas previstas para la fase 2 de la PEC2 pero que ya he realizado parcialmente.

2.2.1 Incorporar los procesos de minería y análisis de texto a la aplicación web

Inicialmente prevista para los días 25 al 29 de abril, la he comenzado mientras aprendía a desarrollar la aplicación Shiny. Tenía sentido hacerlo ya que, aunque ha ralentizado mi ritmo de aprendizaje, ese mismo aprendizaje es más profundo al aplicar los conocimientos a un problema propio en lugar de sólo copiar los ejemplos de manuales y tutoriales.

Los procesos incluidos actualmente en la aplicación Shiny son:

- la recuperación de citas de la página PubMed a partir de palabras clave y un rango de fechas,
- la generación del corpus primario,
- el análisis de la frecuencia de palabras en los sumarios del corpus primario,
- la extracción de genes de los sumarios y el calculo de su frecuencia,
- y la visualización de frecuencias de palabras y genes en forma de tablas y gráficos de barras.

Esta tarea sigue abierta, ya que a medida que implemente nuevos procesos en código R los iré incorporando simultáneamente a la aplicación web.

El código de la aplicación, en su estado actual, está incluido en el apéndice D.

3 Apéndices

3.1 Apéndice A: Código

El documento original en formato .Rmd, que incluye el código completo en lenguaje R usado para generar este informe, se puede consultar y descargar en el siguiente repositorio de Github: jorgevallejo/endometriosis-text-mining

3.2 Apéndice B: Reproducibilidad

```
sessionInfo() # For better reproducibility
```

```
## R version 3.6.3 (2020-02-29)
## Platform: i686-pc-linux-gnu (32-bit)
## Running under: Ubuntu 16.04.7 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/libblas/libblas.so.3.6.0
## LAPACK: /usr/lib/lapack/liblapack.so.3.6.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_GB.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_GB.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_GB.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_GB.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] pubmed.mineR_1.0.17 knitr_1.25
##
## loaded via a namespace (and not attached):
##  [1] XML_3.99-0.3      digest_0.6.27     bitops_1.0-6
##  [4] magrittr_1.5      evaluate_0.14     highr_0.8
##  [7] rlang_0.4.10      stringi_1.4.3     boot_1.3-25
## [10] rmarkdown_2.6     tools_3.6.3       stringr_1.4.0
## [13] RCurl_1.98-1.1    xfun_0.20         yaml_2.2.0
## [16] compiler_3.6.3    R2HTML_2.3.2      htmltools_0.5.1.1
```

3.3 Apéndice C: Generación de tabla de contingencia de genes para distribución hipergeométrica

```
# Download TSV file from PubTator
download.file(url = "ftp://ftp.ncbi.nlm.nih.gov/pub/lu/PubTatorCentral/gene2pubtatorcentral.gz",
              destfile = "data/gene2pubtatorcentral.gz")

# Read the text file into a data frame
gene2pubtator <- read.table(
  file = "data/gene2pubtatorcentral.gz",
  header = FALSE,
  # We only need PMID and GeneID columns
  # Columns marked as class NULL are not read (saving memory)
  colClasses = c("integer", "NULL", "character", "NULL", "NULL"),
  col.names = c("PMID", "Type", "GeneID", "Name", "Resource"),
  fill = TRUE,
  quote = "",
  sep = "\t"
)

# Process multiple gene ID per file
###
# BEWARE: this code is VERY INEFFICIENT because
# it will scan >50 million rows up to 88 times!
# It would be better to keep two dataframes.
# One with only the rows containing multiple geneIDs
# and other with only clean rows.
# Since I have already the contingency table as an RData file... I am not
# going to change the algorithm yet.
###

round <- 0

# Loop will be active while there were semi-colon in column GeneID
while (TRUE %in% grepl(pattern = ";", gene2pubtator$GeneID)) {
# Retrieve indices
double_geneids_vector <- grep(pattern = ";",
                              x = gene2pubtator$GeneID)

# Subset dataframe
gene2pubtator_dup <- gene2pubtator[double_geneids_vector, ]

# Delete first geneID in original df
gene2pubtator$GeneID <- sub(pattern = "^[0-9]+;",
             replacement = "",
             gene2pubtator$GeneID)

# Keep first gene ID in new df
gene2pubtator_dup$GeneID <- sub(pattern = ";[[:graph:]]+",
                             replacement = "",
                              x = gene2pubtator_dup$GeneID)

# Copy new df to original df
```

```
gene2pubtator <- rbind(gene2pubtator,
                       gene2pubtator_dup)
# Delete objects
rm(list = c("gene2pubtator_dup",
            "double_geneids_vector"))

# Show signs of life
round <- round + 1
print(paste0("Round ", round,"!" ))
}
# END OF WHILE LOOP

# Delete duplicated rows
  gene2pubtator <- unique(gene2pubtator)
  })

# Generate contingency table
# Structure: one-dimensional array
# Each geneID is a column, the name of the column is the geneID
geneID_frequencies <- table(gene2pubtator$GeneID)

# Save the precious, precious data
save(geneID_frequencies,
     file = "data/geneID_frequencies.RData")
```

3.4 Apéndice D: Código de la aplicación Shiny

```
library(shiny)
library(easyPubMed)
library(pubmed.mineR)

### Fixed variables ###

# Starting value for data range
# Ten years (in days) before current date
start_date <- Sys.Date() - (365.25 * 10)

### Custom functions ###

# Function for frequency barplots
freq_barplot <- function(varcat, varnum, main = ""){ # Categorical variable
                                                     # and numerical variable
  # Adjust width of left margin
  # https://stackoverflow.com/questions/10490763/
  # automatic-adjustment-of-margins-in-horizontal-bar-chart
  par(mar=c(5.1,
            max(4.1,max(nchar(as.character(varcat)))/1.5) ,
            4.1,
            2.1)
  )

  # The y object retrieves the coordinates of the categories
  # so they can be used for drawing text
  y <- barplot(varnum ~ varcat,
               horiz = TRUE,
               las = 2,
               space = 0.1,
               main = main,
               ylab = "",
               xlab = "",
               xlim = c(0,max(varnum * 1.1)),
               axes = FALSE
  )
  # Writes the frequency of each gen at the end of the bar
  text(rev(varnum),
       y = y,
       labels = rev(varnum),
       adj = NULL,
       pos = 4,
       cex = 0.9
  )
}

## User interface
ui <- fluidPage(
  titlePanel("Endo-Mining",
             windowTitle = "Endo-Mining: minería de textos aplicada a la endometriosis"),
  fluidRow(
```

```
    column(4,
  # Enter keywords
  textInput("keywords",
            label = "Palabras clave",
            value = "endometriosis",
            placeholder = "endometriosis"),
  # Enter date range
  dateRangeInput("fechas",
                 label = "Rango de fechas",
                 start = start_date,
                 format = "dd-mm-yyyy",
                 startview = "year",
                 weekstart = 1, # Monday
                 language = "es",
                 separator = "hasta"),
  # textOutput("keyw"),
  # Search button
  actionButton("search", "Buscar en PubMed")
    ),
  column(8,
  textOutput("n_archivos"),
  # Cites as a table
  tableOutput("titulos")
  ),
  fluidRow(
  # Table of words
    column(6,
           plotOutput("words_barplot"),
  tableOutput("palabras")
    ),
  column(6,

  plotOutput("genes_barplot"),
  tableOutput("genes_table")
    )
  )
  )
)


## App behaviour
server <- function(input, output, session){
  # Generates text for the query
  query <- reactive(
    paste(c(input$keywords, " AND " , format(input$fechas[1],"%Y/%m/%d"),":",
            format(input$fechas[2],"%Y/%m/%d"),"[dp]"), collapse="")
  )

  # # Displays text of the query
  # output$keyw <- renderText(query())

  # Downloads search results
  pubmed_results <- eventReactive(input$search, {
```

```
    # Progress bar
    withProgress(message = "Descargando sumarios desde PubMed...",
                 detail = "Espere, por favor...",
                 value = 0, {
                   incProgress(7/15)
    resultados_busqueda <- batch_pubmed_download(
      pubmed_query_string = query(),
      dest_file_prefix = "pubmed_",
      format = "abstract",
      batch_size = 5000)

    ## Concatenate files
    # List of files to be added together
    files_list <- list.files(pattern = "pubmed_",
                             full.names = TRUE) # include path
    # Create new file
    out_file <- file(description = "todos_resultados.txt",
                     open = "w")
    # Read each downloaded file and write into final file
    for (i in files_list){
      x <- readLines(i)
      writeLines(x, out_file)
    }

    close(out_file)
    # Generate object of class abstract
    abstracts <- readabs("todos_resultados.txt")

    # Delete unnecesary text files
    files_to_delete <- list.files(pattern = "\\.txt$")
    file.remove(files_to_delete)
incProgress(15/15)
})
    abstracts
  })

  # Muestra la cantidad de citas recuperadas
  output$n_archivos <- renderText({
    paste0("Nº de citas recuperadas: ",
    length(pubmed_results()@PMID))
    })

  # Table of pmid plus title
  output$titulos <- renderTable({
    corpus <- pubmed_results()
    if (length(corpus@PMID) <10) {
      citas <- length(corpus@PMID)
    } else {
      citas <- 10
    }

    tabla_titulos <- data.frame(corpus@PMID[1:citas], corpus@Journal[1:citas])
    colnames(tabla_titulos) <- c("PMID", "Publicaciones")
```

```
  tabla_titulos
})

## Preprocesado del corpus primario
# Word atomization
words <- reactive({
  withProgress(message = "Recuperando palabras...",
               value = 0, {
                 incProgress(1/2)
  words <- word_atomizations(pubmed_results())
  incProgress(2/2)
  words
               })
  })
# Table of words
output$palabras <- renderTable({

  tabla_palabras <- data.frame(words()[1:10,])
  colnames(tabla_palabras) <- c("Palabra", "Frecuencia")
  tabla_palabras
})

# Barplot with frequency of words
output$words_barplot <- renderPlot({
  tabla_frecuencias <- data.frame(words()[1:10,])
  tabla_frecuencias$words2 <- factor(tabla_frecuencias$words,
                                     levels = rev(factor(tabla_frecuencias$words)))
  freq_barplot(varcat = tabla_frecuencias$words2,
               varnum = tabla_frecuencias$Freq,
               main = "Palabras más frecuentes")
})

# Gene atomization
genes <- reactive({
  withProgress(message = 'Recuperando genes...',
               detail = 'Suele tardar un rato...',
               value = 0, {
                 incProgress(1/2)
  genes_data <- gene_atomization(pubmed_results())
  # Codify frequency of genes as numeric
  genes_table <- data.frame(genes_data,
                            stringsAsFactors = FALSE)
  colnames(genes_table) <- c("Symbol", "Nombre", "Frecuencia")
  genes_table$Frecuencia <- as.integer(genes_table$Frecuencia)
  incProgress(2/2)
               })
  genes_table
  })

# Table with frequency of genes
output$genes_table <- renderTable({
  genes()
})
```

```
  # Barplot with frequency of genes
  output$genes_barplot <- renderPlot({
    tabla_frecuencias <- genes()[1:10,]
    tabla_frecuencias$genes2 <- factor(tabla_frecuencias$Symbol,
                                       levels = rev(factor(tabla_frecuencias$Symbol)))
    freq_barplot(varcat = tabla_frecuencias$genes2,
                 varnum = tabla_frecuencias$Frecuencia,
                 main = "Genes más frecuentes")
  })



}



## Execution
shinyApp(ui, server)
```

4 Referencias

Fantini, Damiano. 2019. *EasyPubMed: Search and Retrieve Scientific Publication Records from Pubmed*. https://CRAN.R-project.org/package=easyPubMed.

Liu, Ji-Long, and Miao Zhao. 2016. "A PubMed-Wide Study of Endometriosis." *Genomics* 108 (3-4): 151–57. doi:10.1016/j.ygeno.2016.10.003.

National Library of Medicine. 2021a. "About - PubMed." Accessed March 9. https://pubmed.ncbi.nlm.nih.gov/about/.

———. 2021b. "MEDLINE Overview." Accessed March 9. https://www.nlm.nih.gov/medline/medline_overview.html.

Rani, Jyoti, S.Ramachandran, and Ab. Rauf Shah. 2014. *Pubmed.mineR: An R Package for Text Mining of Pubmed Abstracts*. https://CRAN.R-project.org/package=pubmed.mineR.

RStudio. 2021. "Shiny." Shiny. Accessed March 30. https://shiny.rstudio.com/.

Wickham, Hadley. 2021. *Mastering Shiny*. https://mastering-shiny.org/.