UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

Construcción eficiente de datasets de noticias para proyectos de NLP

Marcos Galletero Romero
Tutor: Pablo Ruipérez García

Trabajo Fin de Máster para el
Máster En Ingeniería Y Ciencia De Datos

Escuela Técnica Superior de Ingeniería Informática

2 de octubre de 2021

Dedicado a todos los que han contribuído a que haya podido llegar hasta aquí.

A mis padres.
A mis compañeros en el IFISC,
que me enseñaron cuando no sabía absolutamente nada.
A todos los profesionales de excepcional talento
con los que me he cruzado en mi vida laboral; espero algún día ser como vosotros.

Pero sobre todo, a mis amigos Julián Cendrero e Ivar Zapata.

Índice general

Capítulo 1

Abstract

Las fuentes de datos para estudios de NLP no académicas suelen estar protegidas detrás de un acceso privativo. Los web crawlers (programas que de forma iterativa descargan los archivos HTML de un determinado dominio) permiten que investigadores en NLP tengan acceso a la gran fuente de información que es la red.

Sin embargo, estos no ofrecen una solución completa, pues no pueden acceder a versiones anteriores de webs modificadas o borradas, además de que algunos son difíciles de mantener. Common Crawl ofrece solución al primero de estos problemas, manteniendo un back-up de un gran grupo de dominios a lo largo del tiempo. La resolución del segundo problema es el objeto de este proyecto, que pretende hacer el acceso a los datos de Common Crawl fácil y rápido mediante un diseño cloud.

Keywords: scrapping, Common Crawl, NLP, text mining, cloud architectures, parallel computing, data democratization

Capítulo 2

Prefacio

Sobre la motivación original del proyecto:
Una circunstancia común a todas las startups en ciencia de datos.

El motivo del desarrollo de toda la tecnología que voy a comentar en la memoria de este trabajo fin de máster tiene su origen en una circunstancia que probablemente muchos de los estudiantes de este máster acaben encontrando. A principios de 2019 pasé a formar parte de una muy pequeña empresa tecnológica en ese momento muy rentable, pero cuyo aporte de valor residía en tecnologías que poco a poco estaban perdiendo margen de beneficio al estar aumentando el número de empresas que ofrecen tales servicios. Está muy bien posicionada dentro del sector financiero y decide apostar por una tecnología con buena proyección para los próximos años: la inteligencia artificial y la ciencia de datos. No van desencaminados. A mediados de 2021 se han anunciado los fondos NextGenerationEU que, junto a proyectos de erradicación de la pobreza y sostenibilidad ecológica, ofrecen financiación a proyectos de innovación en inteligencia articial.

Sin embargo, empezar desde cero, siendo una empresa pequeña, sin acceso privilegiado a grandes bases de datos de información, con limitada capacidad para adquirir datos y siendo un grupo excepcionalmente pequeño, todo ello hace que cualquier avance sea muy

complicado, dando la sensación de que proyectos de este estilo sólo pueden funcionar en empresas con altísima capacidad de inversión o fuentes de datos propias.

Aun así, se decide seguir hacia adelante y sacar un primer proyecto que nos coloque en el mercado, aunque sea en un pequeño nicho. Los criterios para decidir la fuente de datos de ese proyecto por aquel entonces indefinido son sencillos: que haya muchos datos, de buena calidad y gratis. Esos criterios los cumplen bastante pocas fuentes de datos, y las empresas y startups que he visto que se han enfrentado a esta situación de incertidumbre siempre miran al mismo sitio: los *web crawlers*.

Por dar un poco más de contexto comentaré la situación de las otras dos startups en las que he estado involucrado, en la primera como amigo de los que la empezaron y la segunda como parte principal de la definición del proyecto.

La primera de ellas consistía en un portal de ayuda a empresas de retail o servicios que estuvieran interesadas en anunciar sus productos vía colaboraciones con *influencers*, en concreto de la aplicación Instagram. Esta red social cuenta con una API para integrar sus servicios de mensajería y publicación de posts desde otras aplicaciones, pero no cuenta con una API de datos al estilo de Twitter. La respuesta: usar un web-crawler aprovechando el portal web de Instagram.

El segundo de los ejemplos se corresponde a un proyecto de análisis sobre el sector inmobiliario, que tenía como objetivo ayudar a empresas y particulares a encontrar el mejor local o vivienda de una forma inteligente, con estadísticas de los resultados y visualizaciones sobre mapas más avanzadas de lo que ofrecen los portales inmobiliarios hoy en día. La fuente de datos usada fue parecida a la primera, un *web crawler* que actúe sobre el portal inmobiliario más usado en España, *Idealista*.

Volviendo a la situación que generó la idea para este proyecto, surgió cuando apareció en escena una necesidad todavía no satisfecha dentro del mundo de las finanzas. Esta necesidad se origina en un mandato del Banco Central Europeo que obliga a los bancos nacionales a realizar provisiones en base a la exposición reputacional de ese banco. Esta decisión surge en el contexto del posible daño que pudiera sufrir un banco al verse sus altos cargos envueltos en corrupción o ser salpicado de escándalos como la venta de activos de alto riesgo como inversiones seguras, cosa que ya ha ocurrido en el pasado. El problema radica en que no hay nadie que diga cuánto dinero hay que provisionar y, tratándose

de bancos, la cifra importa pues es dinero que no se dedica a invertir y está totalmente parado.

Se nos ocurrió una muy buena idea para este problema, apoyada en nuestro conocimiento sobre ciencia de datos y NLP: podíamos usar las noticias publicadas en prensa para estimar cuál es el daño que producen según qué temas en la cotización de una empresa dada.

A pesar de que los tres proyectos están desarrollados en sectores distintos, al ser desarrollado por empresas pequeñas, en este caso también se recurre a la misma solución; un *web crawler* sobre los periódicos nacionales y sobre páginas de cotizaciones abiertas al público (Yahoo Finance).

Este proyecto, tras verse implantado en un banco nacional, aspiraría a poder usar como fuente de datos no ya la cotización de la empresa (disponible públicamente), sino, por ejemplo, el número de impagos de hipotecas, el número de préstamos concedidos, la retirada de efectivo en cajeros, la compra en tiendas mediante datáfonos del banco... Todos ellos, muy importante, datos de acceso restringido gracias a los cuáles es mucho más fácil diferenciarse del resto de empresas y aportar valor.

Tras comenzar el desarrollo de la solución, mientras se buscaban qué otros datos estaban abiertos al público y que pudieran ayudarnos a cerrar el proyecto, se encontró la iniciativa *Open Data* de *AWS*, dentro de la cual estaba incluído el proyecto *Common Crawl*. Tras ver la calidad del proyecto, se decidió usar como fuente principal de datos en el proyecto.

Esta introducción pretende poner en valor el lugar que ocupan los *web crawlers* en un entorno en el que el acceso a datos supone un alto valor estratégico pero reservado a unas pocas empresas.

Capítulo 3

Introducción: Interés del NLP

Desde hace años, los avances en hardware, la generalización del uso de dispositivos móviles conectados a internet y la digitalización de documentación y procesos burocráticos han disparado la cantidad de información generada y guardada en bases de datos en general privativas. De todo el conjunto, se calcula que antes del 2025 el 80 % será información no estructurada. [5] Debido a que es el medio más habitual para transmitir información, el mayor contribuyente a ese conjunto de datos sin estructura es el texto, bien sea en formato plano o alguna otra versión (escáneres de documentación o código HTML entre otros) muy por delante del vídeo o las imágenes. Ahora bien, no todo ese texto es buen candidato para extraer información. [1] hace un muy buen resumen de los principales repositorios de texto con potencia de minado:

1. *Repositorios de documentación*: La producción de publicaciones científicas y libros a día de hoy se hace en formato electrónico, con testimoniales ediciones en formato físico. Este fenómeno ha dado lugar a repositorios de publicaciones con alta densidad de contenido de valor. Campos como la minería de datos para biomedicina explotan ampliamente este tipo de recursos.

2. *Noticias digitales*: El consumo de noticias también ha pasado a ser digital, con particular interés en la inmediatez de la misma. Esto ha provocado el surgimiento

de no pocos flujos de noticias con amplias posibilidades de análisis como Google News o Twitter.

3. *Internet y aplicaciones web*: Los documentos que forman la web están dotados del potente formato de hipertexto que permite relacionar unos documentos con otros. Vienen a su vez acompañados de otros archivos, principalmente imágenes. Estas dos características otorgan mucho valor de cara a un minado, como bien supieron aprovechar en su momento los principales buscadores. Junto a ellas está el hecho de que internet crea plataformas de contenido como foros o chats en los que se representa información con otros diversos formatos de metadatos, creando otra gran fuente de contenido de potencial valor. De todas estas plataformas destacan las diversas redes sociales que hoy día usamos.

Como ejemplos de herramientos de minado de texto o de *procesamiento del lenguaje natural* (NLP) se podrían incluir:

- *Buscadores web* Las distintas empresas que han decidido construir buscadores web han realizado un exhaustivo trabajo de *crawling* (recopilación iterativa de páginas web), indexado de la información y jerarquización de resultados según contexto. En algunos casos como el de Google, el dominio del análisis del lenguaje es tan alto que su recomendación no solo se limita a una búsqueda por palabras (como sería el caso de DuckDuckGo), sino que son capaces de recomendar webs que contengan palabras del *contexto* de tu búsqueda. Muy probablemente esta funcionalidad esté siendo satisfecha accediendo al *embedding* de la búsqueda hecha [6].

- *Categorización de documentos y filtros de spam* La clasificación automática de textos es una tarea de amplio uso en los servidores mail. A pesar de que se ha ido reduciendo con el paso de los años, la proporción de mail considerado *spam* sigue siendo de prácticamente la mitad del tráfico [2].

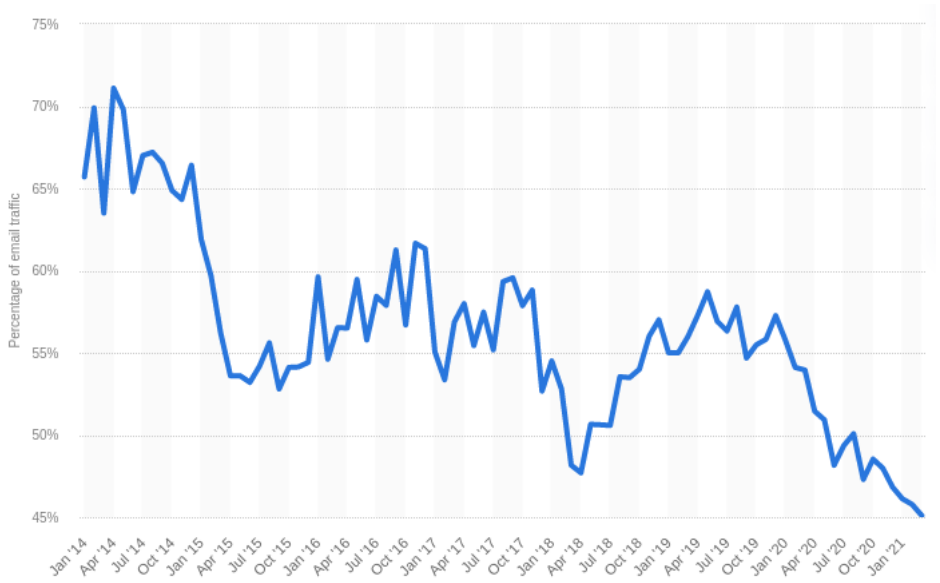


FIGURA 3.1: Porcentaje de mails considerados spam respecto del global

Siendo esta la situación, la clasificación de correo como *spam* o legítimo, fue una de las primeras tareas del NLP. A día de hoy, la clasificación automática de textos ocurre en muchísimas aplicaciones, desde redirección automática de mails a clasificación automática por categorías de teletipos de noticias.

- *Sistemas de recomendación* Analizando el contenido de un conjunto de documentos consumidos por un usuario se le puede recomendar publicaciones de un contenido parecido.

- *Minado de opinión y análisis de sentimientos* Como bien se ha visto en la asignatura de *Redes sociales*, interacciones en plataformas como *Twitter* dan lugar a grupos de alta homofilia [4]. El contenido de los *tweets* generados en tales comunidades es de mucha utilidad para el minado de opinión y puede ser utilizado para segmentación de clientes en marketing o la toma de decisiones informadas en un comité directivo.

Hay, sin embargo, ciertas aplicaciones del NLP que son complicadas de llevar a cabo por dificultades de acceso a datos. Hay un caso en concreto al que me estoy refiriendo, que es el del análisis de datos borrados en internet. La pérdida de información en internet es

algo que va más allá de casos puntuales y de mucha difusión como el borrado de *tweets* de la cuenta de *Donald Trump* o el fin de *Yahoo Answers*. Cada día, por cuestiones de diversa índole, se borran infinidad de webs y en algunos casos dominios enteros. Periódicos online como *El País* pueden permitirse backups de su contenido, accesible por todos (https://elpais.com/archivo/), pero miles de otros dominios digitales se pierden sin dejar rastro, perdiéndose la posibilidad de un análisis en retrospectiva.

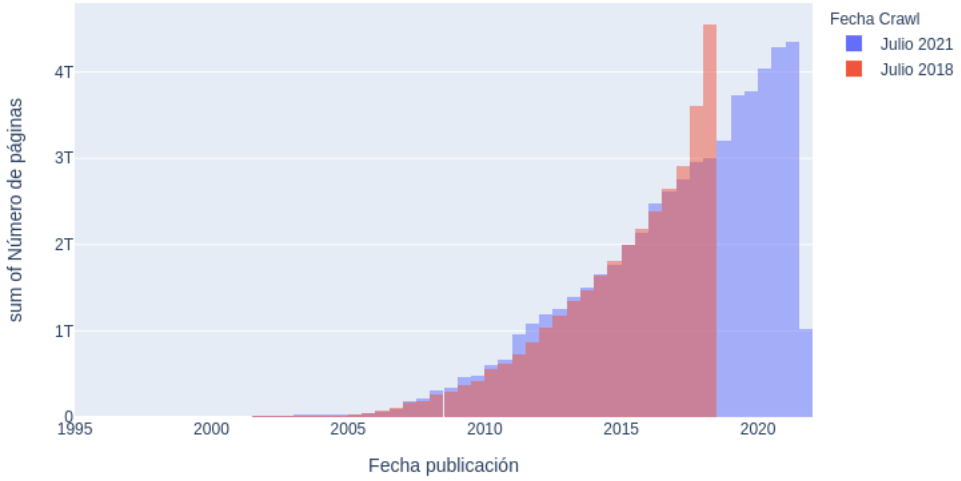


FIGURA 3.2: Agregado de webs disponibles en Common Crawl para dos fechas distintas de crawling en función de su fecha de publicación, en trillones americanos (10^{12}). Hay muchas webs que estaban disponibles a principio y finales de 2018 que a día de hoy han sido borradas o bien modificadas. Datos obtenidos a partir de la herramienta desarrollada.

3.1. Principales técnicas en NLP

Las aplicaciones descritas anteriormente dependen de una representación concreta del texto que se está analizando. El orden en el que aparecen las palabras dentro del documento aporta significado semántico que no puede inferirse a partir de una representación de frecuencia de palabras. Sin embargo, muchas aplicaciones pueden ser satisfechas con este tipo de representación.

Las representaciones de texto que se usan principalmente en la minería de textos son las siguientes:

- Texto como bolsa de palabras

  Es la representación más habitual y sencilla. Consiste en la representación de un corpus de documentos como una matriz en la que las filas son cada uno de los documentos y las columnas se refieren a la frecuencia de cada una de las palabras del vocabulario en los documentos. El vocabulario es un conjunto de tokens decididos por el científico de datos, que toma como punto de partida el conjunto único de palabras del corpus y de él se retiran *stop words* (palabras sin significado). A este vocabulario pueden añadírsele *n-gramas*, que son agrupaciones de palabras con entidad propia, como sería, por ejemplo, "ministerio-interior".

  Esta representación tan sencilla suele ser suficiente en muchas tareas de clasificación y sirve como base para la definición de otras representaciones.

- Term Frequency - Inverse Document Frequency

  Variante del modelo de bolsa de palabras en el que se tiene como objetivo presentar como más relevantes aquellas palabras que son especialmente representativas de un documento. La representación sigue siendo la misma, una matriz con dimensión de número de documentos y número de palabras, pero cambia la forma de calcular el valor a representar, que pasa a ser

$$TFIDF = \frac{termfrequency}{documentfrequency} \tag{3.1}$$

  donde la frecuencia del término es el anterior valor que se usaba en el modelo de bolsa de palabras y el *document frequency* es el número de documentos que contienen el token en cuestión.

- Cálculo de tópicos

  Esta representación satisface la intuición de que un documento puede explicarse

a partir de sus proyecciones sobre varios vectores definidos en el espacio vectorial del vocabulario, de forma parecida a como lo haría un análisis de componentes principales. Estos vectores contienen mucha información contextual que permiten equipararlos al concepto de *temática* y son conocidos como *tópicos*. Un ejemplo de tópico sería el siguiente: [cabo, formentor, mallorca], diferente de [cabo, comandante, guardia-civil]. Como se puede ver en el ejemplo, esta representación permite superar una de las limitaciones de la bolsa de palabras que, al considerar la representación de un documento como un conjunto de palabras clave, pone como iguales en la *feature* 'cabo' a documentos que se refieren a temáticas totalmente distintas.

Es útil comentar ejemplos de proyectos en los que se ha utilizado satisfactoriamente esta representación. El primero de ellos consiste en un sistema de redirección automática de correos en función del contenido del correo. Así, si el correo trata sobre la temática 'altas de línea' se redirige a la cuenta de la persona encargada de tal tarea, que sería una cuenta de correo distinta a la que se encarga del tópico (temática) 'reclamaciones'.

Otro ejemplo en el que también ha sido útil usar tópicos es el proyecto que dió lugar al contenido de este trabajo (comentado en el Prefacio). Si contamos con un corpus de noticias, podemos representar cada uno de esos documentos por las temáticas que mejor definen al conjunto y, junto a la fecha de cada uno de esos documentos, se puede obtener una serie temporal de relevancia temporal de un tema de conversación en medios. Así, se pueden conseguir series temporales como la presencia en medios de la temática 'independencia de Cataluña', o de temas relacionados con criptomonedas, por ejemplo.

Estos tópicos se calculan mediante algoritmos como Latent Dirichlet Allocation, Biterm Topic Modelling o Topics Over Time.

- Representación vectorial - Word2Vec

  Esta representación es a la que se suele llamar *word embedding*. Esta tecnología consiste en una red neuronal cuyos pesos han sido entrenados para aprender asociaciones entre palabras a partir de un gran corpus de texto. Este sistema permite

representar una palabra mediante un vector cuyos pesos respetan una serie de propiedades con respecto a la similaridad del coseno. Así, se cumple el comportamiento descrito en la figura 3.3, donde traslaciones paralelas en el espacio de embedding son correspondidas por transformaciones semánticas equivalentes. Un documento en esta representación correspondería a una matriz de vectores de embedding.

Un ejemplo de uso de los word embedding se encontraría en los algoritmos de búsqueda. Dentro del proveedor de servicios de Azure, existen servicios como Azure Cognitive Search que permiten crear buscadores en torno a una base de datos de documentos (documentos en formato pdf, docx o similares) y personalizar las búsquedas asociadas, como hace, por ejemplo, el buscador web Google al introducir un pequeño texto en su caja de búsqueda. Una forma de ofrecer búsquedas similares consiste en entrenar word2vec en el contenido de tu base de datos y ofrecer aquellas palabras que tengan mayor similaridad del coseno en el espacio de embedding de aquellas que forman el vocabulario de tu corpus de documentos.

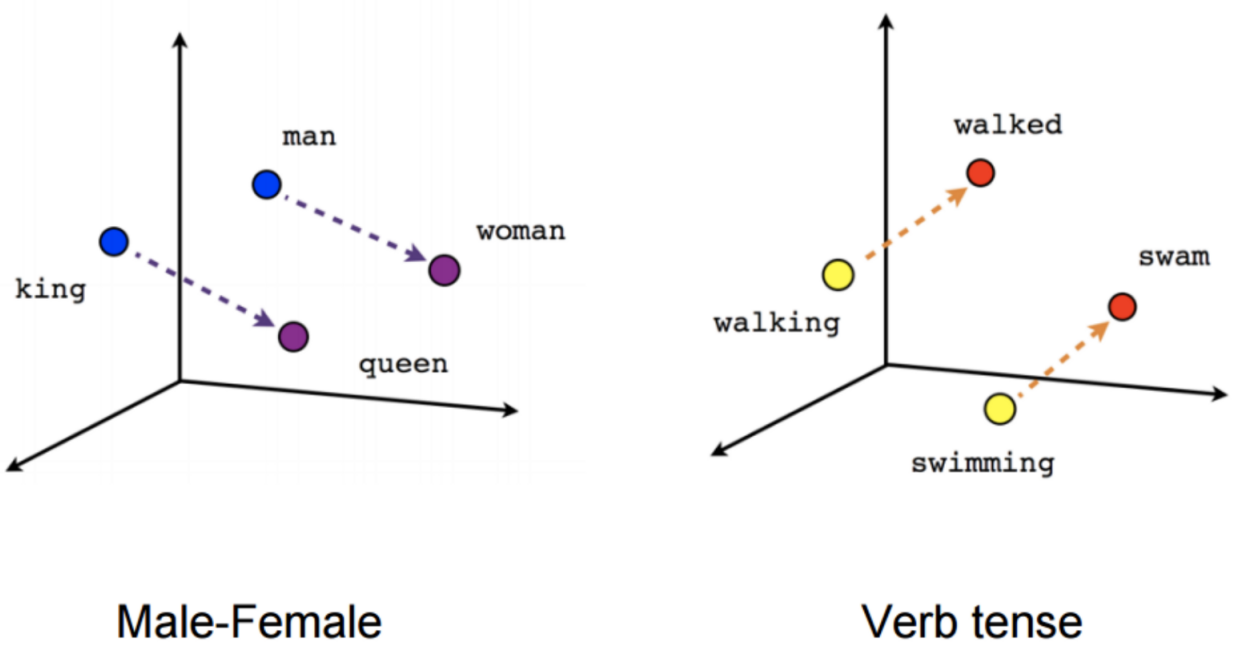


FIGURA 3.3: Símil tridimensional del comportamiento de los vectores de embedding ante traslaciones paralelas.

- Representación vectorial de secuencias

  BERT es una red neuronal entrenada sobre el corpus de Wikipedia mediante una metodología de máscaras. Esta metodología consiste en la entrega como input de secuencias (frases) con algún hueco o la entrega de párrafos con huecos en secuencias completas, y se entrena para conseguir un estimador que sea capaz de encontrar palabras y secuencias, respectivamente, que tengan sentido en ese hueco. La tarea de predecir palabras se conoce como *language modelling* y la de predecir secuencias se conoce como *next sentence prediction*. Al final se acaba con una red que contiene una representación del contexto de todas las palabras y frases con las que ha sido entrenado. Para nuestro caso en concreto, la representación vectorial de las secuencias de un documento se obtendría pasándole como entrada a la red cada una de las secuencias de nuestro documentos y obteniendo los pesos de una capa intermedia de la red. Es una representación que tiene en cuenta el sentido semántico de las frases y es usado en aquellas tareas de inteligencia artificial que necesiten entender el texto más allá de como un conjunto de palabras, como puede ser el caso de proyectos como Chatbots inteligentes, que incluyan *question answering* (QA).

Capítulo 4

Acceso a datos

Hablar sobre el valor de los datos o de cómo la información es el nuevo petróleo en la época en la que estamos es hablar de lo obvio. A día de hoy, quien no está aprovechando su acceso privilegiado a datos, lo está vendiendo. *BBVA*, hasta hace unos años banca tradicional, ha diversificado su actividad para convertirse en proveedor de datos, ofreciendo el timestamp, importe, producto comprado y geolocalización asociada a todas las compras hechas en sus datáfonos.

En relación a proveedores de texto, el ecosistema está dominado por empresas tradicionalmente asociadas a la producción de noticias de diverso ámbito, uniéndoseles también las nuevas plataformas de redes sociales. Internet, sin embargo, cuenta con casi infinito contenido accesible por todos, brindando la oportunidad a muchos proyectos Open Source de ser proveedores también.

4.1. Soluciones comerciales

- *Elsevier* Es la mayor editorial científica del mundo. Organiza su contenido por *journals* de distintas temáticas. Sus principales servicios son *Science Direct* y *Scopus*, que ofrecen acceso restringido a publicaciones científicas y a una completa base de

datos de referencias bibliográficas y *abstracts* permite explorar el contenido de las referencias de un artículo en concreto.

- *Factiva* Herramienta de inteligencia de negocio, agregadora de publicaciones de medios restringidos y públicos que ofrece alertas, indexación y otros servicios sobre este contenido. Otorga acceso a más de 32.000 fuentes de datos, incluyendo periódicos, revistas, transcripciones de radio y televisión. Es particularmente valioso el hecho de que ofrecen mucha riqueza en cuanto a metadatos de la noticia, que automáticamente mejoran la calidad de tus análisis, al permitirte ser mucho más específico con los documentos que estás usando.

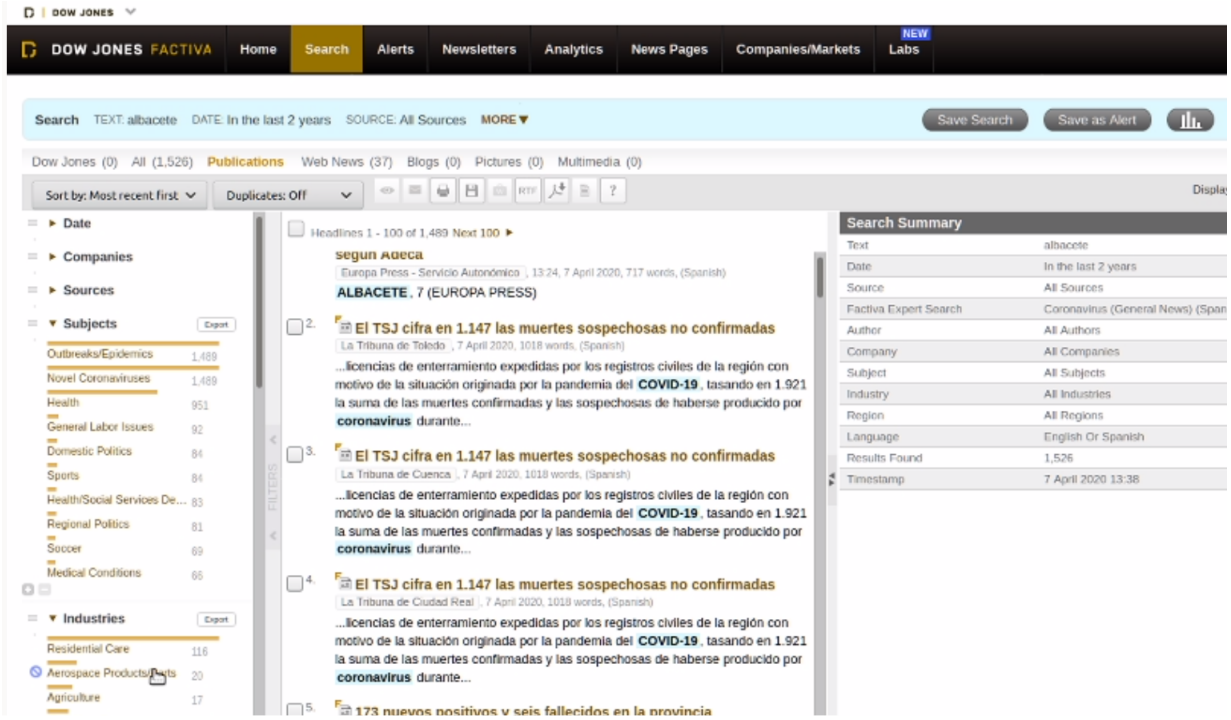


FIGURA 4.1: Portal de la herramienta de datos de Factiva, que da acceso a información documental de muy alta calidad, con gran granularidad en el aporte de metadatos. Contiene información sobre qué industrias, temas, regiones y empresas se mencionan en la noticia. Cuenta con series temporales para estudiar el ritmo de publicación de una determinada query de documentos.

- *Bloomberg* Compañía que ofrece servicios de información, principalmente financiera. Dan acceso a su terminal de análisis de mercado e inversión, en la que viene incluída otro agregador de noticias parecido al de Factiva.

4.2. Soluciones Open Source

- *Arxiv* Repositorio abierto de publicaciones científicas que tiene como objetivo democratizar el acceso a los últimos resultados de la investigación global como contrapunto a Elsevier. Puede criticársele el hecho de que *cualquiera* puede publicar, independientemente de la calidad de su investigación.

- *Web Crawlers*

  - *Scrapy* Framework open source para la creación de crawlers web en Python. Flexible en su configuración, permitiendo conseguir un crawler funcional en poco tiempo.

  - *Nutch* Es el crawler que utiliza Common Crawl (descrito más abajo) y cuenta con potentes integraciones con indexadores y parseadores de documentos.

  - *Heritrix* Usado por una de los proyectos de *web archiving* más famosos, el *Wayback Machine* de *Internet Archive*. No cuenta con una comunidad tan grande como Scrapy, ni con unas características técnicas tan potentes como Nutch.

- *Common Crawl* Realiza una tarea de crawl, indexación y guardado de crawls de internet parecida a la que hace Google. El resultado de estos crawls es puesto a disposicíón de todos en abierto. Dedico el capítulo siguiente a hablar sobre este proyecto y qué datos podemos esperar encontrar en él.

Capítulo 5

Common Crawl

5.1. Información general y acceso a sus repositorios

Es una organización sin ánimo de lucro que realiza amplios crawls de toda la web cada mes y los ofrece en abierto como enormes archivos comprimidos alojados en buckets de AWS S3. Sus crawls respetan las políticas de robots.txt, descritas en la figura 5.1. Common Crawl se enmarca dentro de un proyecto mayor llamado AWS Open Data, que contiene gran cantidad de datasets de temática variada (https://registry.opendata.aws/). Podemos encontrar datasets de imágenes satelitales de la tierra actualizadas cada 5 días, datasets de medicina o datasets de histórico climático, entre otros.

En torno al proyecto de Common Crawl hay otros que lo complementan, como el *Common Crawl Index*. Este proyecto tiene como objetivo indexar los billones de dominios que se guardan en cada crawl cada mes. Los índices de los archivos pueden verse en la figura 5.2 y el resultado de realizar una query a cada uno de ellos, esencial para el desarrollo de este proyecto, se puede ver en la figura 5.3, en la que aparecen los campos *offset* y *length* para una página web en concreto. Con estas dos variables, ya podemos descargar una página web capturada. Sencillamente debemos pedir una cantidad de bytes igual a *length* a partir del *offset* en bytes indicado para la web que nos interesa, dentro de cada uno de los archivos comprimidos que se publican mensualmente.

El crawler de Common Crawl está hecho usando el framework *Nutch* de Apache. Este crawler tiene las mismas posibilidades que Scrapy (mencionado anteriormente) pero cuenta con mayor extensibilidad, lo que lo hace más interesante desde el punto de vista de puesta en producción. Esta extensibilidad consiste principalmente en su comportamiento modular con respecto a tecnologías como *Apache Solr* y *Elastic Search* (usados en este caso para indexar las webs descargadas) y potentes parseadores como *Apache Tika*, que son capaces de extraer texto de archivos Word, PDFs o presentaciones PowerPoint. Esto hace que uno tenga que dedicar mucho menos tiempo a pensar cómo almacenar la información descargada y cómo extraer el contenido en texto de archivos PDF.

Si se usa Scrapy, esos pasos hay que decidirlos y para nada son una tarea trivial. De querer copiar la infraestructura de Common Crawl, el problema sería el lenguaje en el que se define su funcionamiento (Java), que al menos en 2021, para un científico de datos es un lenguaje con el que no estará tan familiarizado como Python, R o JavaScript.

```
# 80legs
User-agent: 008
Disallow: /

# 80legs' new crawler
User-agent: voltron
Disallow: /

User-Agent: bender
Disallow: /my_shiny_metal_ass

User-Agent: Gort
Disallow: /earth

User-agent: MJ12bot
Disallow: /

User-agent: PiplBot
Disallow: /

User-Agent: *
Disallow: /*.json
Disallow: /*.json-compact
Disallow: /*.json-html
Disallow: /*.xml
Disallow: /*.rss
Disallow: /*.i
Disallow: /*.embed
Disallow: /*/comments/*?*sort=
Disallow: /r/*/submit$
Disallow: /r/*/submit/$
Disallow: /message/compose*
Disallow: /api
Disallow: /post
Disallow: /submit
Disallow: /goto
Disallow: /*after=
Disallow: /*before=
Disallow: /domain/*t=
Disallow: /login
Disallow: /remove_email/t2_*
Disallow: /r/*/user/
Disallow: /gold?
Disallow: /static/button/button1.js
Disallow: /static/button/button1.html
Disallow: /static/button/button2.html
Disallow: /static/button/button3.html
Disallow: /buttonlite.js
Disallow: /timings/perf
Disallow: /counters/client-screenview
Allow: /
Allow: /sitemaps/*.xml
Allow: /posts/*
```

FIGURA 5.1: Archivo robots.txt de la web *reddit.com*. Este archivo es una guía de cooperación entre el crawler y el sito web, es decir, no obliga a no recabar los datos, sino que informa de su deseo de que los dominios indicados no sean scrapeados por el crawler indicado mediante su user-agent. Este archivo no solo se usa para evitar la descarga de webs por parte de crawlers como los tratados en este trabajo, también valen para mejorar el posicionamiento de una página web en los navegadores al evitar que los crawlers de, por ejemplo, Google, indexen el contenido de subdominios que el gestor de la web sabe que no son de interés de cara a un navegador web. Si se quisiera, por ejemplo, que una web no apareciera en Google, tendríamos que restringir el acceso a toda la web al crawler con User-Agent: googlebot.

Common Crawl Index To Main Archives 2013-2021

Index files of the monthly crawl archives are used on index.commoncrawl.org, see the announcement of the Common Crawl index.

The following monthly crawl archives provide index files on the prefix `s3://commoncrawl/cc-index/collections/CC-MAIN-XXXX-YY/`:

ID	Announcement	Billion Pages	Index File Listing
CC-MAIN-2021-31	July/August 2021	3.15	CC-MAIN-2021-31/cc-index.paths.gz
CC-MAIN-2021-25	June 2021	2.45	CC-MAIN-2021-25/cc-index.paths.gz
CC-MAIN-2021-21	May 2021	2.60	CC-MAIN-2021-21/cc-index.paths.gz
CC-MAIN-2021-17	April 2021	3.10	CC-MAIN-2021-17/cc-index.paths.gz
CC-MAIN-2021-10	February/March 2021	2.70	CC-MAIN-2021-10/cc-index.paths.gz
CC-MAIN-2021-04	January 2021	3.40	CC-MAIN-2021-04/cc-index.paths.gz
CC-MAIN-2020-50	November/December 2020	2.64	CC-MAIN-2020-50/cc-index.paths.gz
CC-MAIN-2020-45	October 2020	2.71	CC-MAIN-2020-45/cc-index.paths.gz
CC-MAIN-2020-40	September 2020	3.45	CC-MAIN-2020-40/cc-index.paths.gz

FIGURA 5.2: Índices de cada archivo Common Crawl

sempleo/elpepueco/20070110elpepueco_7/Tes", "mime": "text/html", "mime-detected": "text/html", "status": "301", "digest": "OV625HPYNIBNEJDRCGAX6PBIIAVTIL4P", "length": "1045", "offset": "8119565", "filename": "crawl-data/CC-MAIN-2021-31/segments/1627046154175.76/crawldiagnostics/CC-MAIN-20210801092716-20210801122716-00160.warc.gz", "redirect": "/economia/2007/01/10/actualidad/1168417979_850215.html"}
{"urlkey": "com,elpais)/articulo/economia/solo/parados/tienen/muchas/posibilidades/salir/desempleo/elpepueco/20070110elpepueco_7/tes", "timestamp": "20210801111449", "url": "https://www.elpais.com/articulo/economia/Solo/parados/tienen/muchas/posibilidades/salir/desempleo/elpepueco/20070110elpepueco_7/Tes", "mime": "text/html", "mime-detected": "text/html", "status": "301", "digest": "ICTVRJ7UAOJBDJ3655DDCTIQIVB5T637", "length": "805", "offset": "21426591", "filename": "crawl-data/CC-MAIN-2021-31/segments/1627046154175.76/crawldiagnostics/CC-MAIN-20210801092716-20210801122716-00473.warc.gz", "redirect": "https://elpais.com/articulo/economia/Solo/parados/tienen/muchas/posibilidades/salir/desempleo/elpepueco/20070110elpepueco_7/Tes"}
{"urlkey": "com,elpais)/articulo/economia/soy/europesimista/aplica/medicina/equivocada/elpepueco/20120129elpepieco_7/tes", "timestamp": "20210806031146", "url": "http://www.elpais.com/articulo/economia/Soy/europesimista/aplica/medicina/equivocada/elpepueco/20120129elpepieco_7/Tes", "mime": "unk", "mime-detected": "application/octet-

FIGURA 5.3: Ejemplo del resultado de una query a un índice de Common Crawl

La forma de conseguir estos índices es a través de una API REST que, a partir de una petición *GET* con la query te ofrece el resultado que ya hemos comentado. El código usado desde el backend del proyecto para conseguir ese índice es el que se puede ver en la figura 5.4

```python
import urllib.parse
import logging
import time
# from botocore.vendored import requests
import requests

logger = logging.getLogger()
logger.setLevel(logging.INFO)
logging.basicConfig(format='%(asctime)s: %(message)s',
                    level=logging.INFO)


def fetch_result_page(
        api_url,
        url,
        page,
        timeout=30,
        max_retries=20,
        delay_retries=0.5):
    """
    query the api, getting a stream with the specified results page
    """

    query = {'url': url,
             'page': page,
             'output': 'json'}

    query = urllib.parse.urlencode(query)

    retry = 0
    while True:
        r = requests.get(api_url + '?' + query, timeout=timeout)
        if r.status_code == 404:
            logger.info('No Results for for this query')
            return None
        elif r.status_code != 200:
            logger.info('Status code {}, Retrying'.format(r.status_code))
            retry += 1
            if retry == max_retries:
                logger.info('Max retries exceeded for url {}, page {}'.format(url, page))
                return None
            time.sleep(delay_retries)
            continue

        break

    return r.text
```

FIGURA 5.4: Código usado en el backend del proyecto para descargar los índices asociados a cada web localizada.

5.2. Estadísticas de datos descargados

El tamaño descomprimido de cada archivo mensual en el repositorio de Common Crawl es del orden de 270 TB. Como el crawler de Common Crawl salta de unas páginas a otras utilizando los hipervínculos a partir de una serie de localizaciones *semilla*, es complicado

saber con qué conjunto de webs final acabará el archivo. Por esto, es interesante comentar estadísticas de los archivos de Common Crawl.

Una estadística interesante es la distribución de idiomas en las webs encontradas, siendo el inglés el lenguaje de internet, con un 44 % de las webs obtenidas.

crawl	CC-MAIN-2021-21	CC-MAIN-2021-25	CC-MAIN-2021-31
Idioma	%	%	%
Inglés	44.8424	45.1857	45.5605
Ruso	7.2658	6.7665	7.2898
Alemán	5.6906	5.5856	5.9029
Chino	4.6508	5.0691	4.1251
Japonés	3.5850	4.3859	4.9273
Francés	4.4747	4.3725	4.5090
Español	4.3315	4.3107	4.4105
Desconocido	2.8840	2.3539	1.7425
Italiano	2.4277	2.4112	2.4354
Portugués	2.1373	2.1200	2.2068
Neérlandes	1.8082	1.7758	1.9176
Polaco	1.6050	1.5885	1.6471
Checo	1.1124	1.0004	1.1233
Turco	1.0099	1.0407	1.0210
Indio	0.8311	0.8450	0.8500
Vietnamita	0.8036	1.5225	0.9478
Sueco	0.7398	0.7236	0.7292

CUADRO 5.1: Proporción de idiomas en las webs recolectadas por Common Crawl para los últimos tres meses.

Completando la estadística anterior sobre idiomas, se pueden obtener cuáles son los dominios con mayor representación.

Dominio	Páginas	% de páginas
blogspot.com	19184361	0.606124
wordpress.com	12788305	0.404042
livejournal.com	8508551	0.268825
wikipedia.org	6187601	0.195495
photoshelter.com	2879154	0.090966
europa.eu	1846329	0.058334
fandom.com	1803875	0.056993

Dominio	Páginas	% de páginas
eleconomista.es	160534	0.005072
csic.es	160376	0.005067
upv.es	149969	0.004738
gva.es	128915	0.004073
uv.es	124363	0.003929
ugr.es	121593	0.003842
museoreinasofia.es	119632	0.003780

CUADRO 5.2: Tabla con los dominios más representados dentro de los archivos de Common Crawl, tanto en el global como para dominios terminados en *.es*

Los datos de Common Crawl pueden ser analizados desde el punto de vista del análisis de redes, siendo los nodos cada una de las páginas web y las aristas dirigidas los hipervínculos que ligan unos nodos con otros. La herramienta para realizar este análisis puede encontrarse en el repositorio de código de [3]. Una muestra de los resultados que se pueden obtener de tales análisis son los de la figura 5.5 que muestra en escala logarítmica la distribución del número de hipervínculos salientes y entrantes de cada una de las webs analizadas.

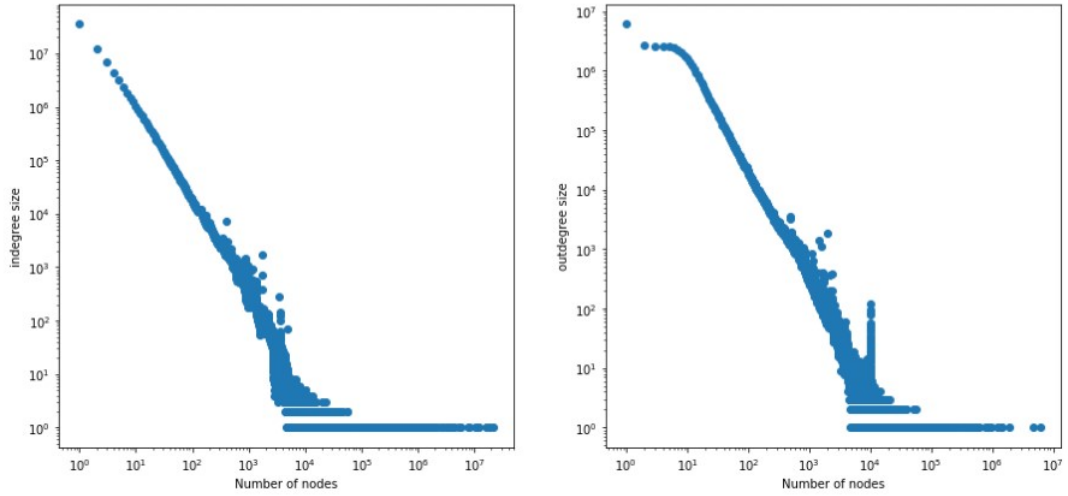


FIGURA 5.5: Distribución del grado de entrada y de salida para el grafo formado por las webs encontradas en el archivo de Common Crawl Mayo 2020. El número de webs decae exponencialmente con el grado de entrada y salida.

Capítulo 6

Arquitectura de la solución

La arquitectura consta de los elementos tradicionales dentro de un proyecto web, pero con un par de decisiones de diseño que abordan el problema de la lenta descarga. Hablaré sobre esto al final del capítulo, después de introducir el resto de componentes. Puede verse un diagrama de todos los componentes en la figura 6.1.

El diseño del frontend se ha hecho con *Vuetify*, una librería de componentes de *Vue*, que es a su vez un *web-framework* de JavaScript. Un *web-framework* es un lenguaje o librería que te permite abstraer el proceso de creación de código *JavaScript* y maquetación *HTML* y *CSS*, en lugar de tener que crearlo directamente. La introducción de esta librería ahorra mucho trabajo de diseño, pues en una aplicación que no cuente con él, habría que diseñarlo todo; desde botones y maquetación hasta detalles más complicados como el *look and feel* de la web. Este último punto lo resuelve *Vuetify* adoptando el estilo de *Material Design*, que a pesar de que aporta poca novedad dado el hecho de que es utilizado por miles de webs y apps para móvil, permite sin embargo poco esfuerzo conseguir una aplicación presentable y funcional.

En cuanto a las funcionalidades hechas en el frontend, se reducen a la mínima extensión posible que hace al proyecto cumplir su objetivo, pues la novedad del mismo no está en sus bondades de cara al usuario, sino en la forma en la que se ejecutan las tareas del backend. Esto quiere decir que la aplicación cuenta con una pequeña gestión de usuarios,

con pantalla de login, un apartado de tres etapas para la selección de dominios, descarga de sus índices y descarga de documentos, respectivamente; un segundo apartado para la visualización de las descargas que todavía no han acabado y un último apartado dedicado a la visualización del dataset construido (realizado gracias al componente de Vue de Ag Grid (https://www.ag-grid.com/vue-data-grid/components/).

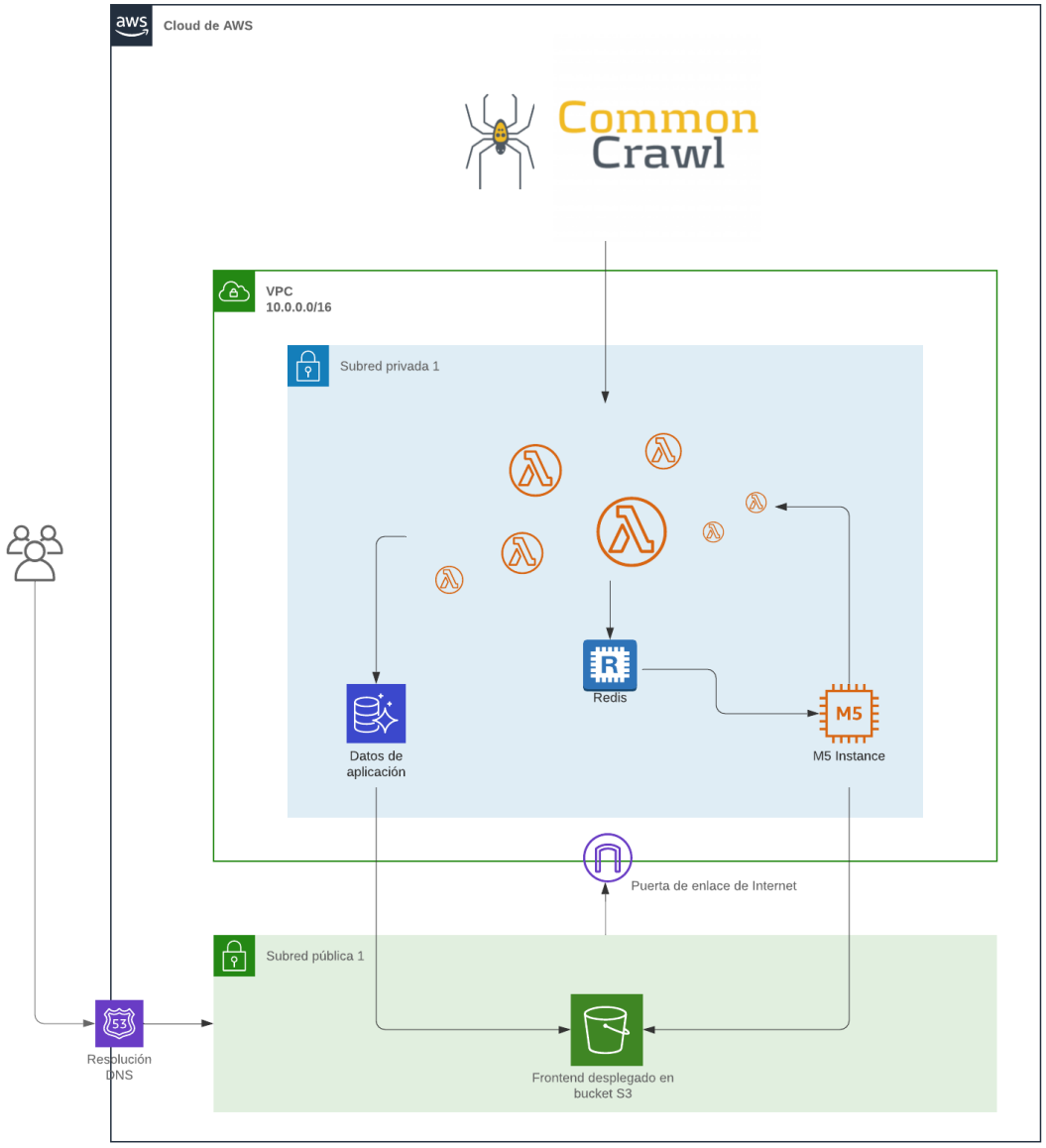


FIGURA 6.1: Arquitectura cloud completa del proyecto

En cuanto a despliegue del mismo se refiere, *AWS S3* ofrece una buena forma de alojar

páginas web en un bucket público. Además, gracias a la amplia comunidad que hay en el mundo del desarrollo web, hay plugins de *npm* (manejador de paquetes de JavaScript) que hacen de este despliegue algo muy sencillo (https://www.npmjs.com/package/vue-cli-plugin-s3-deploy).

Hay dos desarrollos hechos en el frontend que sí son algo diferencial y que tienen mérito remarcar. El primero se corresponde a la introducción de GraphQL, un servicio que permite interactuar con la base de datos directamente desde el frontend, ahorrando el desarrollo de endpoints de interacción con base de datos en el backend. Así, por ejemplo, no existe una tarea de borrado de dataset en el backend, sino que toda esa lógica de borrado se pasa al frontend, haciendo la tarea de programación algo menos compleja.

El otro desarrollo a poner en valor es el árbol de subdominios implementado en la etapa tres del primer apartado del frontal. Este árbol permite elegir qué subdominios te interesan de un determinado domino descargado. Se calcula dinámicamente para cada dominio en particular.

Las tareas asociadas a cada evento desencadenado por el usuario en la interfaz de usuario son satisfechas por un servidor REST que funciona desde una instancia EC2 en nuestro tenant de AWS. Este ha sido hecho con *Flask*, otro *framework*, esta vez de Python, para el desarrollo de servidores web. Explicado de forma intuitiva, este servidor contiene un archivo Python principal en el que se mapean rutas web a funcionalidades particulares (ver figura 6.2). Así, hay un endpoint para crear usuarios y cambiar contraseñas en la tabla de usuarios, otro para devolver la información del usuario logueado, otro para devolver la información del contenido de cada rama dentro del árbol de subdominios... Con respecto a este endpoint para devolver la información de subdominios, su respuesta cuenta con una memoria en caché mediante la extensión de Flask *Flask Caching* (https://flask-caching.readthedocs.io/en/latest/index.html), introducida para que, si se abre y cierra una rama del árbol, no se tenga que esperar tanto como la primera vez que se han conseguido los datos desde el servidor REST.

```
api = Api(app)

endpoints = [
    {"resource": User, "url": '/user', 'protected': False},
    {"resource": UserInfo, "url": '/user-info', 'protected': True},
    {"resource": CorpusDatasets, "url": '/corpus/datasets', 'protected': True},
    {"resource": CorpusDocumentSearch, "url": '/corpus/document-search', 'protected': True},
    {"resource": IndexDatasets, "url": '/commoncrawl/index-datasets', 'protected': True},
    {"resource": IndexPreview, "url": '/commoncrawl/index-preview', 'protected': True},
    {"resource": IndexDownloadQueue, "url": '/commoncrawl/index-download-queue', 'protected': True},
    {"resource": IndexDownloadAWSLambda, "url": '/commoncrawl/index-download-aws-lambda', 'protected': True},
    {"resource": CCIndexRecords, "url": '/commoncrawl/cc-index-records', 'protected': True},
    {"resource": DocumentDownloadQueue, "url": '/commoncrawl/document-download-queue', 'protected': True},
    {"resource": DocumentDownloadAWSLambda, "url": '/commoncrawl/document-download-aws-lambda', 'protected': True},
    {"resource": ShowProgress, "url": '/text-mining-queue/show-progress/<string:job_id>', 'protected': True},
    {"resource": ShowProgressAWSLambda, "url": '/text-mining-queue/show-progress-aws-lambda', 'protected': True},
    {"resource": GetSearch, "url": '/commoncrawl/get-search', 'protected': True},
    {"resource": GraphQL, "url": "/graphql", 'protected': False},
]
```

FIGURA 6.2: Endpoints de las distintas funcionalidades asociadas al servidor REST

Otro de los elementos centrales de la aplicación es la base de datos que respalda las actividades de descarga. En principio, el tamaño del dataset diseñado por el usuario puede ser tan grande como lo necesite. Siendo este el caso, no se puede confiar en una base de datos montada sobre la misma instancia que sostiene el servidor REST, pues se podría agotar el almacenamiento, y hay que recurrir a una RDS (sistema de bases de datos de AWS). La base de datos es relacional de tipo PostgreSQL y contiene los datos de usuario, las indexaciones de las webs dentro de los archivos de Common Crawl y las propias webs descargadas, a la espera de ser comprimidas para la posterior descarga desde el frontend por el usuario. La creación de la base de datos se ha hecho mediante un ORM de SQLAlchemy (librería para tal fin en el lenguaje Python). Este módulo crea una representación de las tablas en forma de clase (véase figura 6.3) que permite por un lado mantener una versión en código de lo que hay en la base de datos y, por otro, permite no realizar la interacción con la base de datos desde Python vía SQL, sino mediante instancias de clases de cada una de las tablas definidas, algo más conveniente.

En la base de datos también se ha definido un campo con soporte para Full Text Search. Este campo responde a un trigger que en cada inserción de documentos, transforma el contenido de los campos de texto *clean_text* y *title* a un array de strings. Este tipo de columna ofrece funcionalidades de búsqueda que son usadas en la pantalla de inspección

de datasets, lo que permite al usuario buscar una palabra que le interese y encontrar qué documentos la contienen.

```
class Document(Base):
    __tablename__ = 'document'

    id = Column(BigInteger, primary_key=True, autoincrement=True, server_default=FetchedValue())
    dataset_id = Column(ForeignKey(f'{Base.metadata.schema}.dataset.id'), index=True)
    title = Column(Text,
                   comment='As provided by the obtention procedure (short text of tweet, ...) or extracted from the web
    author = Column(Text,
                    comment='As provided by the obtention procedure (user writter of tweet, ...) or extracted from the w
    url = Column(Text, comment='url to the original content')
    html_content = Column(Text, comment='Download page in HTML format or link to download (CommonCrawl)')
    clean_text = Column(Text,
                        comment='Text of the document, as extracted from the htmlcontent or provided by the obtention pr
    This is the "real" content to be analysed by NLP tools')
    publish_date = Column(DateTime, comment='As provided by the obtention procedure or extracted from the web page')
    obtention_date = Column(DateTime)
    fts = Column(TSVECTOR)
    # entities = Column(MutableDict.as_mutable(JSONB))
    entities = Column(JSONB)
    lang = Column(Text)
    lang_probability = Column(Float(53))

    dataset = relationship('Dataset', primaryjoin='Document.dataset_id == Dataset.id', backref='documents')

    __table_args__ = (
        Index('document_entities_index', entities, postgresql_using='gin'),
        Index('document_fts_index', fts, postgresql_using='gin'),
        {'schema' : Base.metadata.schema,
         'comment': 'Holds web pages, tweets, ... (in general, text documents)'})
```

FIGURA 6.3: Definición de la tabla de documentos en el ORM de SQLAlchemy.

La novedad que se presenta en el diseño Cloud de esta aplicación es el original uso del servicio *AWS Lambda*. Este servicio normalmente se usa para el desarrollo de aplicaciones *serverless*. Este tipo de diseños cloud están muy de moda estos días, pues permiten prescindir de ese servidor en instancia EC2 que se usa en este diseño. Para entender mejor su caso de uso, me refiero a la imagen 6.2 que contiene la relación de endpoints del servidor usado. En una arquitectura *serverless*, cada una de esas funcionalidades se vería sustituida por funciones *Lambda* que, explicado sencillamente, son ordenadores de escasísimos recursos cuyo tiempo máximo de ejecución es 15 minutos.

La razón por la que se usan en este proyecto es para optimizar tiempos. En la descarga de recursos de Common Crawl hay principalmente dos factores determinantes que actúan como cuello de botella: el número de procesos paralelos que realizan distintas partes de la query y el ancho de banda entre el origen y el ordenador que realiza la descarga.

Si uno quiere obtener el mejor desempeño posible tiene que atajar ambos problemas a la vez.

Sabemos que los archivos generados por Common Crawl se almacenan en la red us-east-1 de AWS. El mejor ancho de banda al que vamos a poder optar se consigue levantando las bases de datos e instancias en esta misma red. Ahora bien, el problema no acaba aquí. Para aumentar el número de procesos de descarga no basta con que la tarea de descarga la realice la instancia EC2 en una ejecución en paralelo. El ancho de banda máximo para un ordenador dado es exactamente el mismo independientemente del número de procesos que esté ejecutando. Si se quiere dar el mayor ancho de banda posible a cada proceso hay que ejecutar cada uno de ellos en distintas instancias. *Lambda*, a pesar de no estar diseñado para esto, parece un muy buen candidato para resolver el problema de una descarga masiva y rápida en paralelo, pues la descarga de archivos no demanda muchos recursos de procesamiento y, a pesar de lo barato que es *Lambda* en comparación con *EC2*, el ancho de banda del que dispone es el mismo.

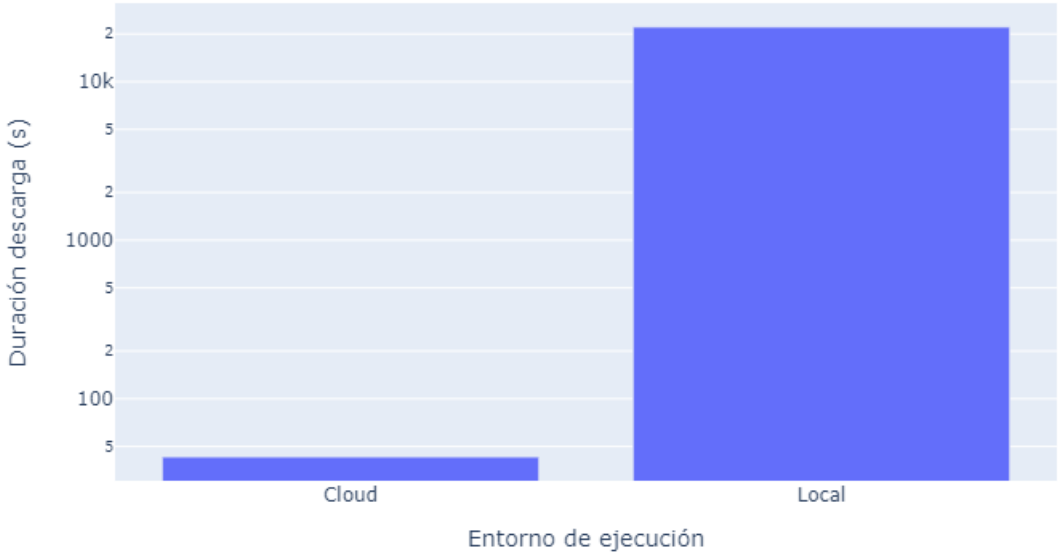


FIGURA 6.4: Duración de la descarga de los dominios www.elpais.com, www.expansion.com y www.libremercado.com para el crawl de Julio de 2021 para una ejecución con Lambdas en paralelo y una ejecución local. Se mejora de un tiempo de descarga cercano a las 6 horas a un tiempo de descarga de 43 segundos.

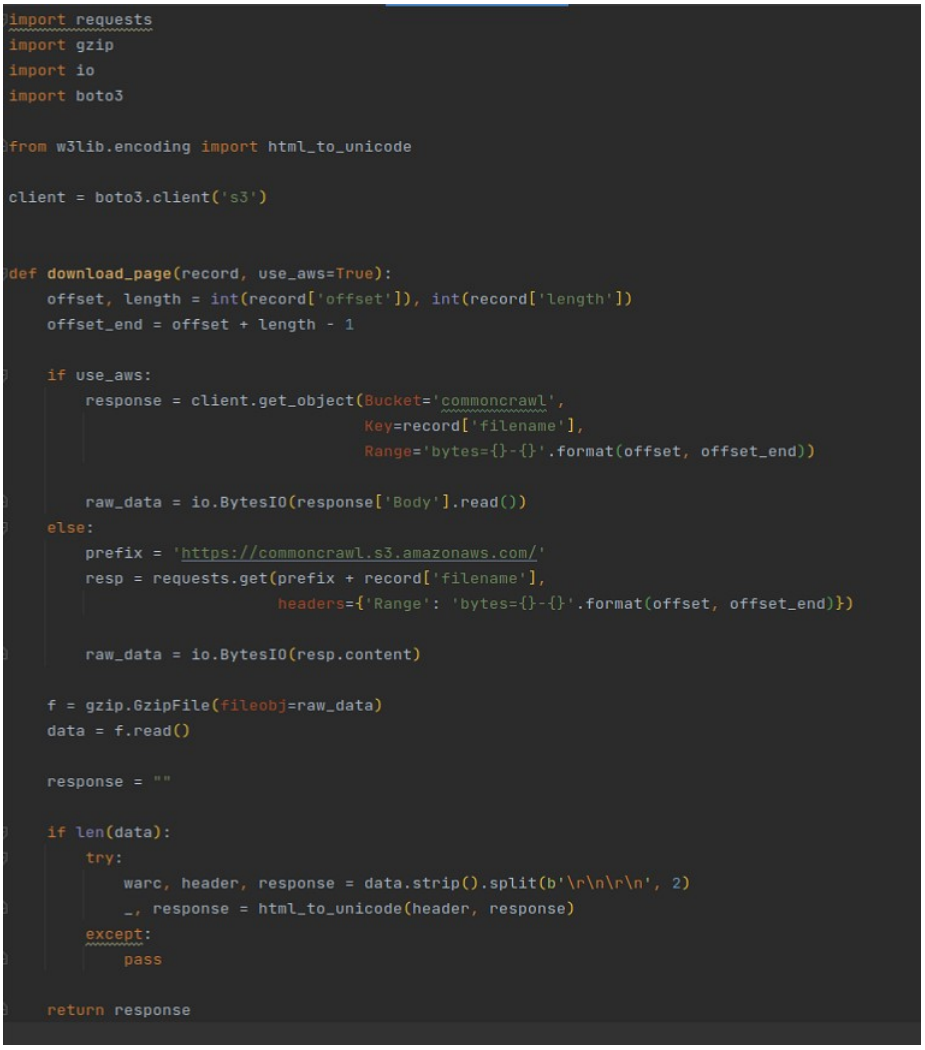
Una vez el usuario ha decidido qué query quiere hacer, los recursos a buscar se reparten en trozos de 500 webs por ejecución de *Lambda*. Esta cantidad se ha elegido en base a pruebas de tiempo de descarga para obtener una cifra que permita aprovechar al máximo el tiempo de ejecución de *Lambda* sin entrar en *TimeOut*.

Barajé también la posibilidad de que, en caso de que alguna de las *Lambdas* estuviera llegando a la cifra de 14 minutos de ejecución, ella misma invocara otra *Lambda* encargada de terminar el trabajo asignado. Acabé descartando esta opción por estar satisfecho con la baja tasa de error en ejecuciones.

En cuanto a la implementación técnica de las funciones Lambda se refiere, consiste en un paquete de despliegue que contiene el código que se va a ejecutar en forma de función de Python (lo que se conoce como el *lambda_handler*) y todo el entorno de librerías necesarias para que pueda funcionar esa función. Tal y como he comentado antes, las funciones

Lambda suelen utilizarse para tareas que no requieren un gran despliegue técnico por lo que AWS proporciona un entorno de librerías básico. A esto hay que añadirle que no hay forma de decirle a AWS que utilice un determinado entorno virtual (al estilo de un requirements.txt o archivo Pipfile del gestor de entornos Pipenv). Uno pensaría entonces que con añadir las librerías de su entorno local al paquete de despliegue resolvería el problema, pero eso no es suficiente, pues resulta que determinadas librerías, entre ellas las de amplio uso *numpy* y *pandas*, cuentan con archivos compilados en el entorno en el que se ejecutan, teniendo esto como resultado que uno no pueda copiar y pegar las librerías que usa localmente para desarrollar el código en el paquete de despliegue. La solución final consiste en levantar una instancia EC2 que ejecute el mismo sistema operativo que va a correr el entorno Lambda, instalar y compilar las librerías en esa instancia, volcarlas al paquete de despliegue (por supuesto, no compatible con tu entorno local) y, ahora sí, desplegar el código en el servicio AWS Lambda. La puesta en producción de este servicio, sin duda, fue más laboriosa que el hecho de programar la lógica interna. En cuanto al código que se ejecuta en cada Lambda, toda su funcionalidad central puede verse en la figura 6.5, que contiene de forma explícita cómo interactuar con los archivos de Common Crawl (es una petición de archivo desde la API de Python para AWS, *boto3*. El resto de la funcionalidad es accesoria y corresponde a formateo de datos e inserción en base de datos. Como pequeño comentario, la extracción de la fecha de publicación de una página web se ha realizado gracias a la librería *htmldate* (https://htmldate.readthedocs.io/en/latest/).

El último elemento a comentar es la base de datos Redis, que sirve como contador del número de ejecuciones Lambda que han terminado. Al invocar el enjambre de Lambdas, el servidor pone ese contador (una clave:valor) con valor igual al número de Lambdas a invocar. Cada vez que una de ellas termina, se reduce en uno ese contador.



```python
import requests
import gzip
import io
import boto3

from w3lib.encoding import html_to_unicode

client = boto3.client('s3')


def download_page(record, use_aws=True):
    offset, length = int(record['offset']), int(record['length'])
    offset_end = offset + length - 1

    if use_aws:
        response = client.get_object(Bucket='commoncrawl',
                                     Key=record['filename'],
                                     Range='bytes={}-{}'.format(offset, offset_end))

        raw_data = io.BytesIO(response['Body'].read())
    else:
        prefix = 'https://commoncrawl.s3.amazonaws.com/'
        resp = requests.get(prefix + record['filename'],
                            headers={'Range': 'bytes={}-{}'.format(offset, offset_end)})

        raw_data = io.BytesIO(resp.content)

    f = gzip.GzipFile(fileobj=raw_data)
    data = f.read()

    response = ""

    if len(data):
        try:
            warc, header, response = data.strip().split(b'\r\n\r\n', 2)
            _, response = html_to_unicode(header, response)
        except:
            pass

    return response
```

FIGURA 6.5: Interacción explícita con el contenido de los crawls de Common Crawl.

Capítulo 7

Experiencia de usuario

El usuario comienza en una pantalla de Login en la que puede iniciar sesión como en cualquier otra web.

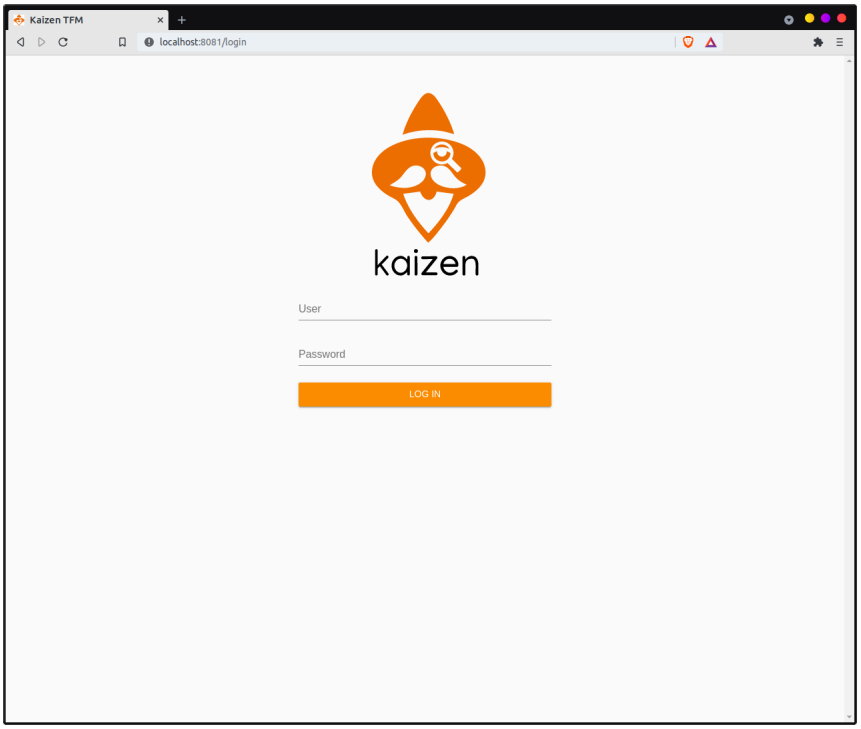


Figura 7.1: Inicio de sesión

El primer elemento que se encontrará el usuario en la barra de navegación será el apartado de creación de datasets. Este apartado, dividido en tres etapas, es el que le permite crear su dataset.

En la primera etapa puede seleccionar qué archivo de Common Crawl quiere utilizar como fuente para su dataset y, en el formulario de debajo, puede incluir cualquier web en el formato aceptado por Common Crawl Index.

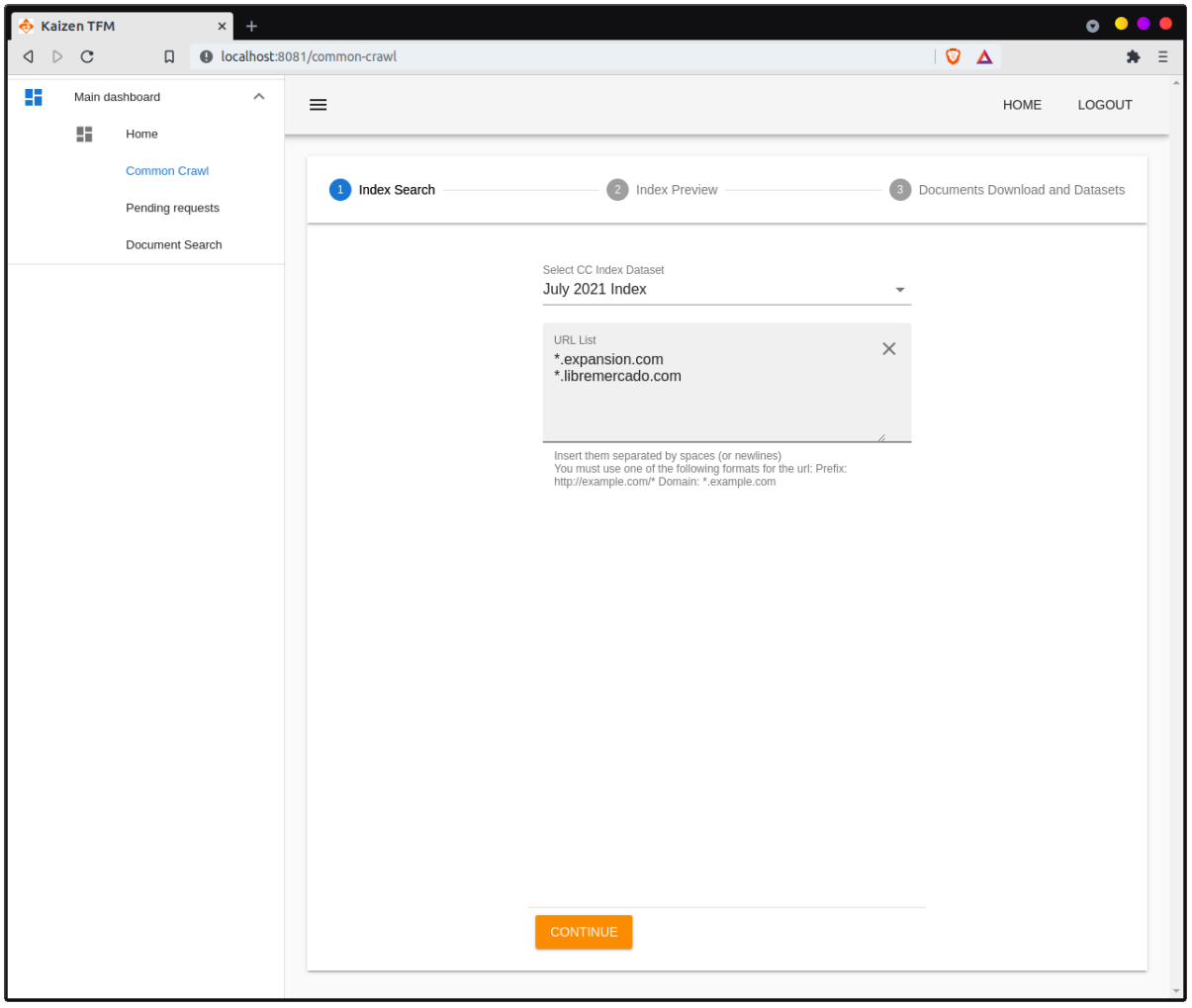


FIGURA 7.2: Selección del archivo Crawl a examinar y la query al Common Crawl Index

La segunda etapa corresponde a la pantalla de confirmación de descarga de la indexación de las webs que son de interés y da una estimación del tamaño de cada dominio.

El número que aparece se refiere al número de páginas de Common Crawl que contienen documentos de ese dominio. Esto quiere decir que si se indica un 22 en la columna 'Num index pages', significa que el backend pedirá los documentos asociados a cada una de las 22 páginas dadas. por ejemplo para el periódico *El País*, una de estas páginas estaría dada por el resultado de esta petición web (puede verse en cualquier navegador): https://index.commoncrawl.org/CC-MAIN-2021-31-index?url=*.elpais.com/&page=0. Para obtener todas las webs almacenadas por Common Crawl para un determinado dominio, habría que descargar todas las webs hasta llegar al page=21.

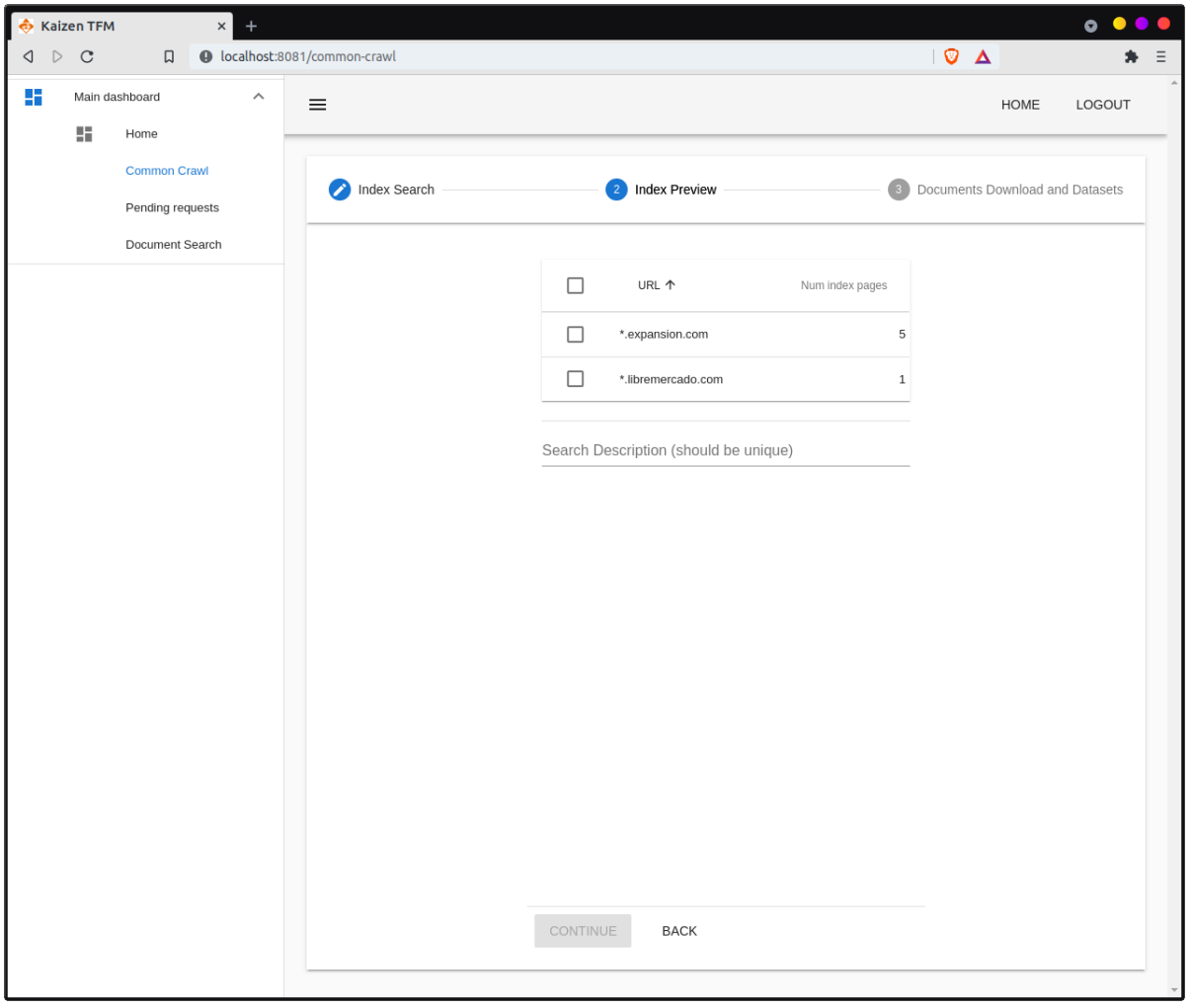


FIGURA 7.3: Confirmación de qué índices se van a guardar en base de datos

La descarga de los índices también se realiza con funciones Lambda y tardan entre 15 y 20 segundos.

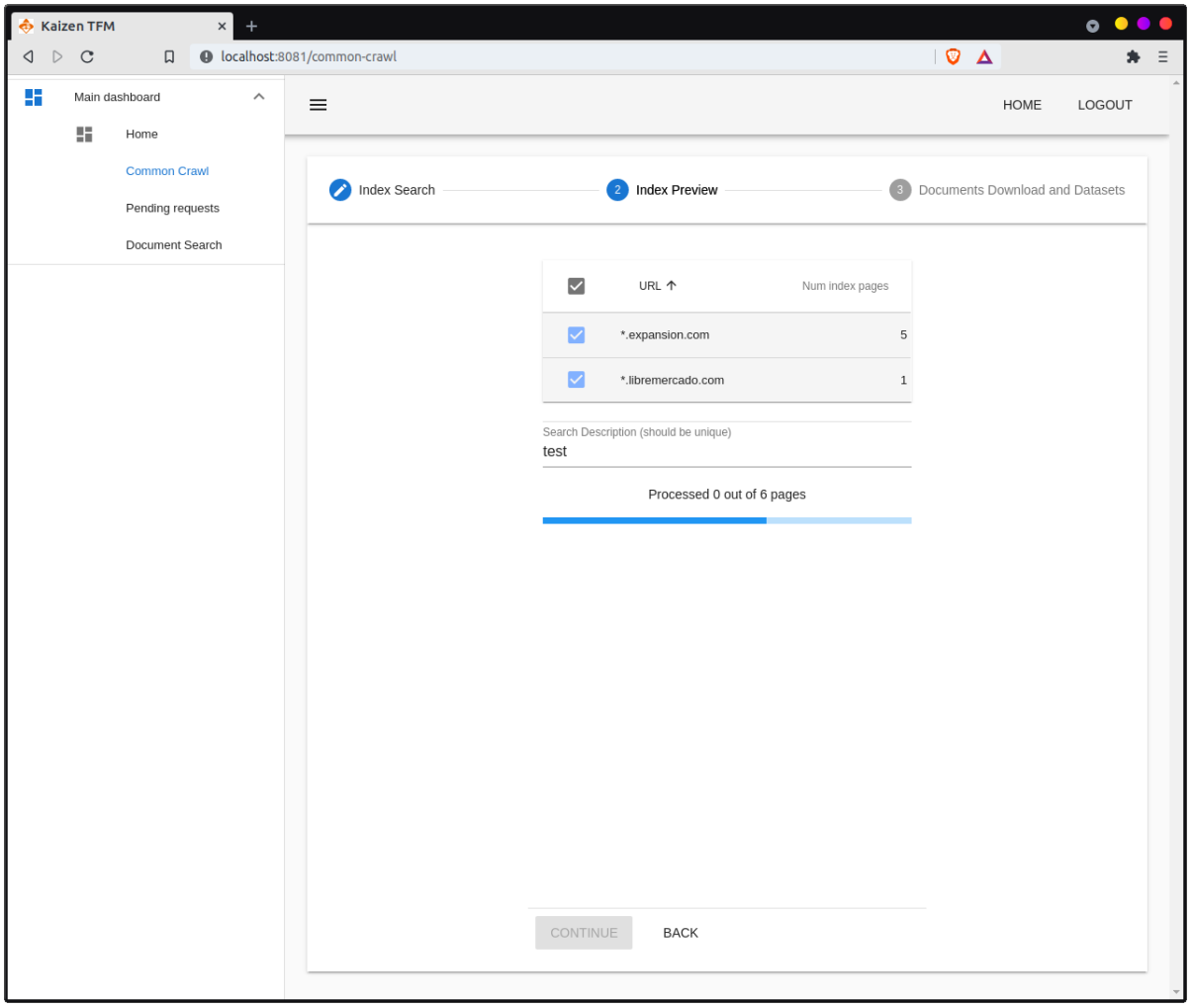


FIGURA 7.4: Ejecución de las Lambdas de descarga de índices

La última etapa de la creación del dataset se refiere al ya comentado árbol de selección de subdominios, calculado de forma dinámica y a propósito para cada dominio que haya elegido el usuario. Aquí uno podría, por ejemplo, descartar las secciones de deportes de la creación de un análisis, o quedarnos con las secciones de economía de cada periódico que hayamos elegido (tendrá distinto formato en cada periódico). Una vez seleccionados, damos un nombre al dataset y lo descargamos.

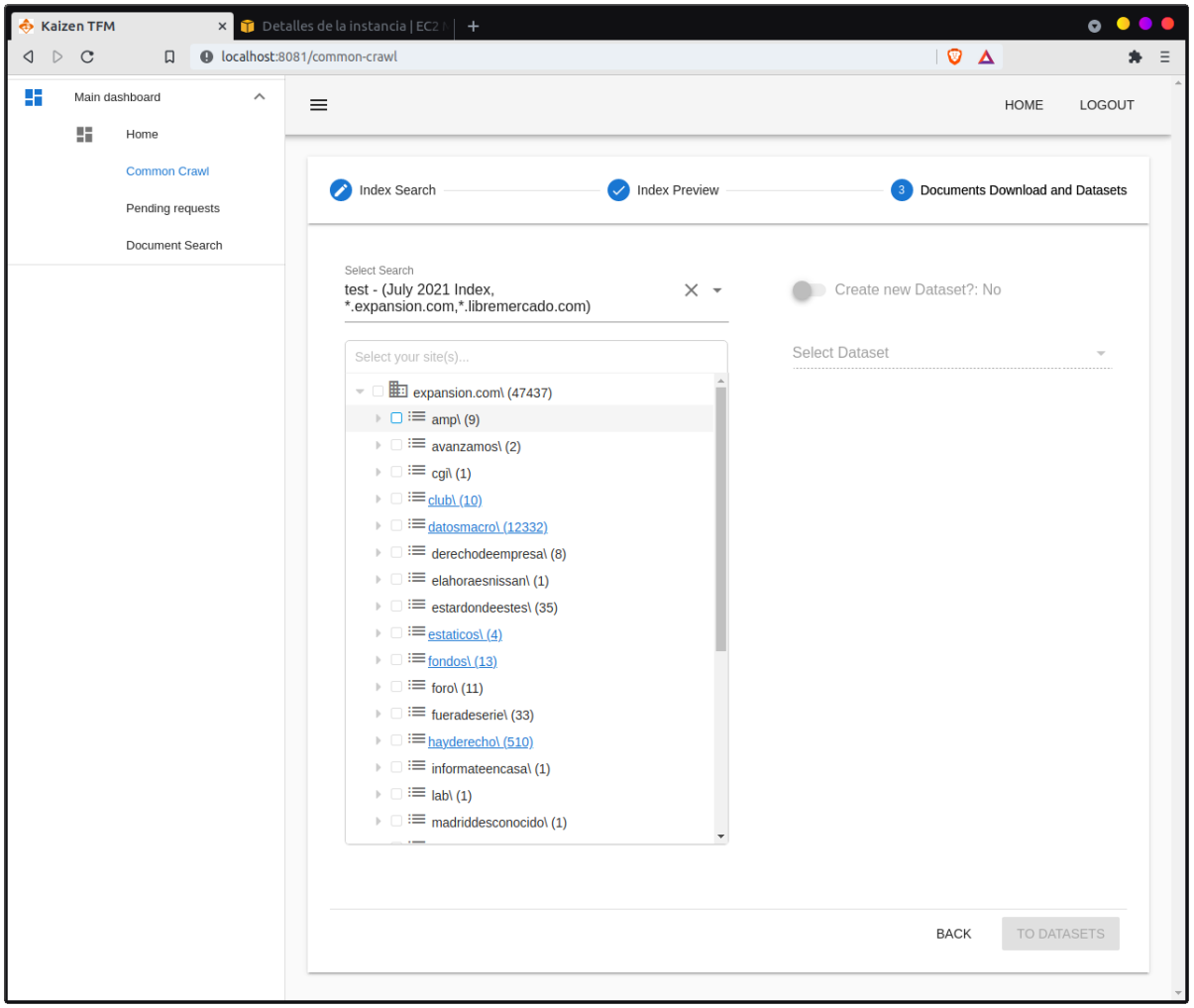


FIGURA 7.5: Selección de qué subdominios se desean

Tras esperar alrededor de un minuto, el dataset ya estará completamente descargado y podremos ver el contenido. Podemos buscar palabras concretas para ver qué noticias las contienen o incluso podemos hacer una query al estilo SQL que satisfaga situaciones como por ejemplo, una búsqueda en las que queremos buscar un término pero excluir otro.

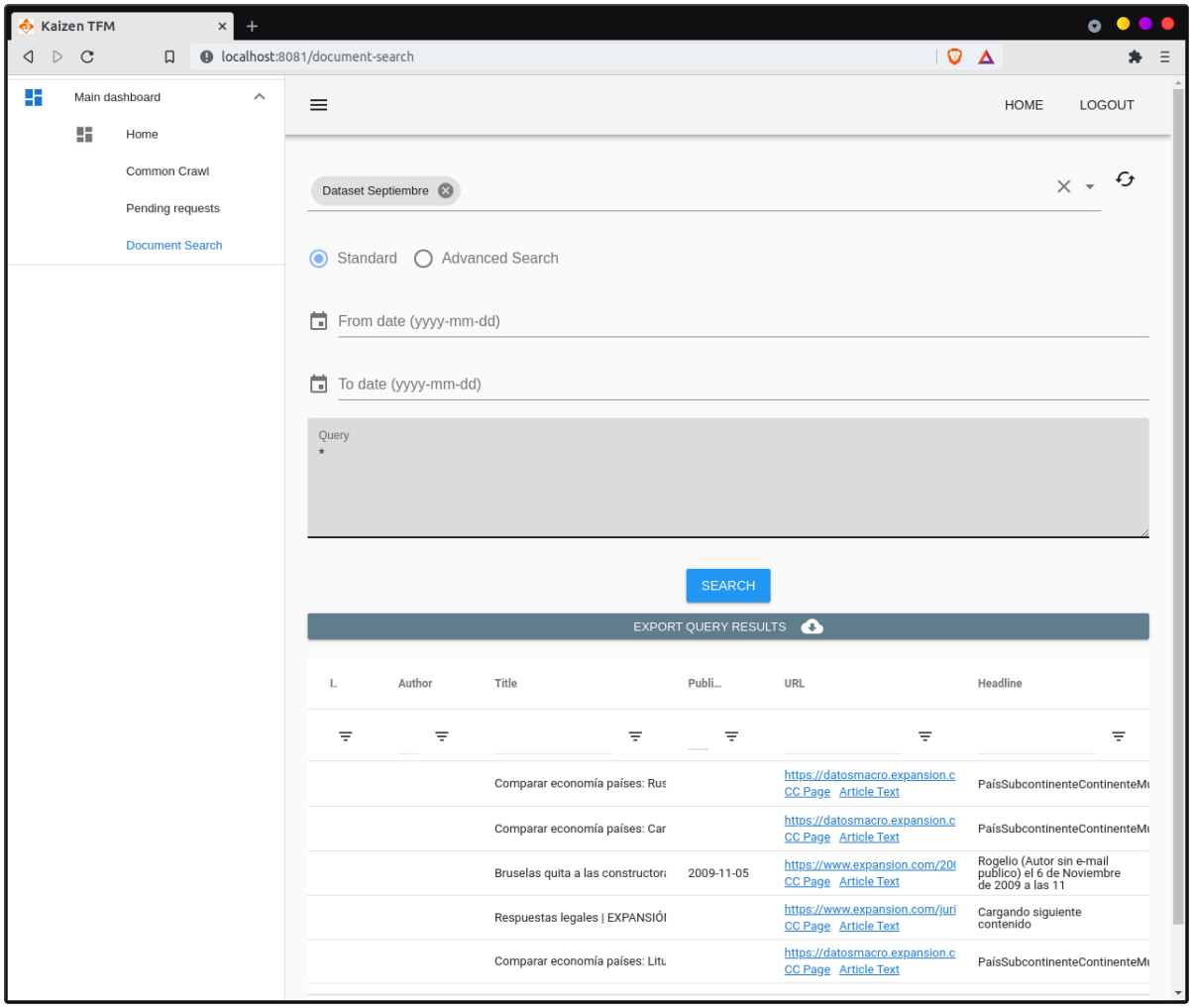


FIGURA 7.6: Tras la selección de los subdominios a descargar por las Lambdas y la ejecución de las mismas, el usuario puede inspeccionar su dataset generado

Una vez el usuario haya decidido si quiere quedarse con la totalidad del dataset generado o con un subconjunto (obtenido a partir de la query), puede exportar los resultados del dataset a un archivo *jsonl*, que consiste en una lista de objetos *json* en el que cada uno de ellos ocupa una fila del archivo.

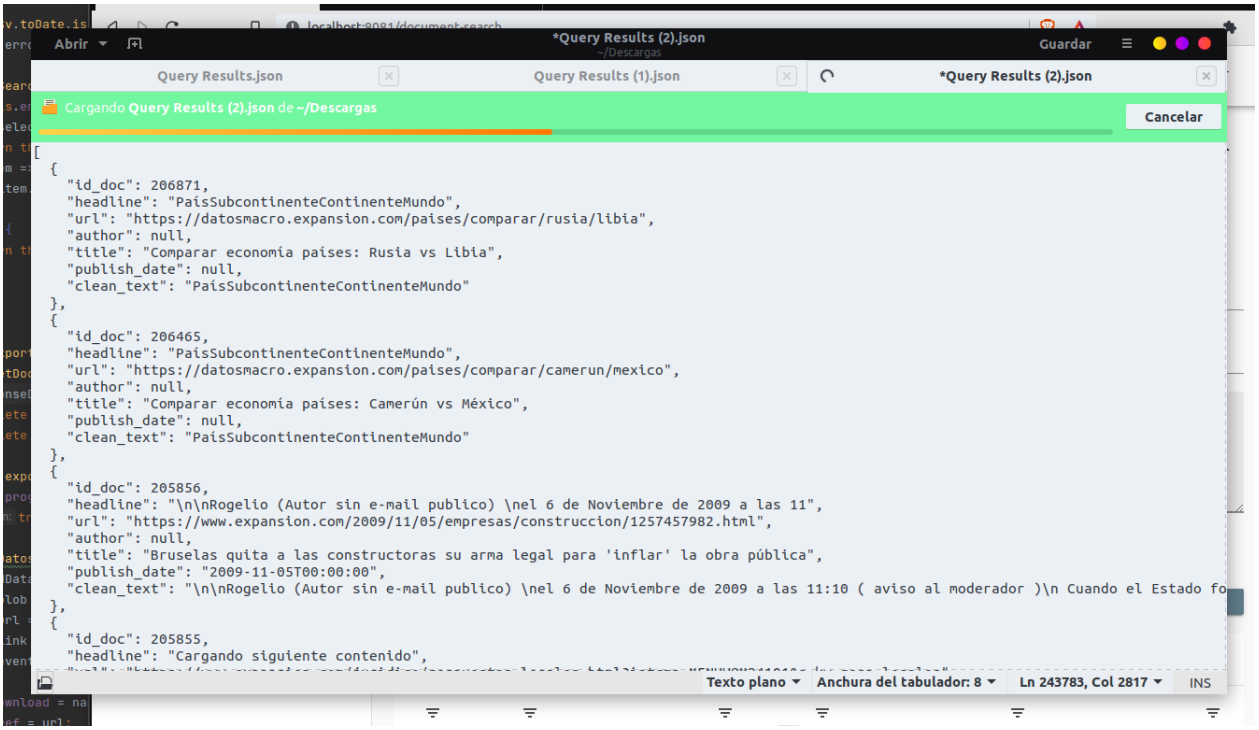


FIGURA 7.7: Archivo jsonl que ya puede ser utilizado para cualquier análisis de NLP

La carga en Python del dataset descargado para inspeccionarlo puede realizarse en un par de líneas y se puede ver en la figura 7.8:

```
import pandas as pd
import json
with open('/home/mgalletero/Descargas/Query Results.json', 'r') as f:
    df = pd.DataFrame(json.load(f))
```

FIGURA 7.8: Código usado para cargar el archivo descargado en Python.

Una vez ya tenemos este archivo, podemos utilizarlo para infinidad de tareas, desde minado de opinión, hasta detección de tendencias.

Capítulo 8

Comentarios finales

Este proyecto es una versión reducida del proyecto original. El diseño inicial contaba con preprocesado adicional antes de la inserción a base de datos. Este consistía en una extracción de entidades del texto de las webs que permitía realizar filtrados por entidad a la hora de crear datasets.

Este cálculo de entidades se realizaba mediante el servicio Comprehend de AWS. La razón de retirarlo fue el elevadísimo precio, lo que implicó un rediseño de la solución. Este episodio de rediseño, junto a una descripción de la distribución de horas a lo largo del proyecto, se puede encontrar en el diagrama 8.1.

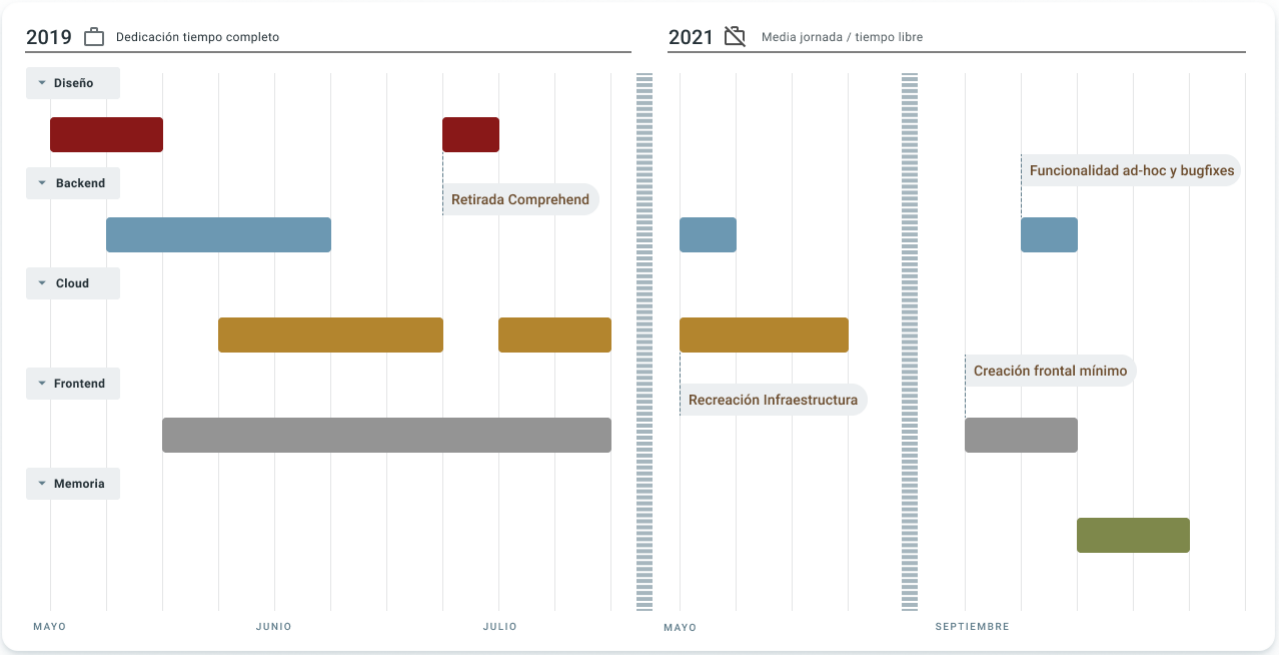


FIGURA 8.1: Diagrama de Gantt con el tiempo dedicado a cada apartado a lo largo del tiempo.

Para arreglar este problema sin afectar a la funcionalidad ya hecha recurrimos a la autogestión de un servicio de extracción de entidades basado en servicios básicos de AWS (EC2), retirándonos de cualquier servicio de alto nivel que ofreciera AWS. Este sistema consistía en una potente instancia de EC2 levantada bajo demanda por el servidor REST que automáticamente ejecutaba en inicio un servidor FastAPI configurado para ejecutar las funciones de análisis NLP ofrecidas por el proyecto de la universidad de Stanford, *CoreNLP* (https://stanfordnlp.github.io/CoreNLP/).

Esta solución fue particularmente satisfactoria, ya que no solo se redujeron los costes de ejecución a una centésima parte del coste de partida, sino que mejoraron tiempos de ejecución y simplicidad del código, pues Comprehend requería un formato de datos distinto al usado en toda la aplicación y, al gestionar el servicio de extracción, se puede aplicar ese mismo formato dentro del "nuevo servicio".

Toda esta arquitectura servía, hasta finales de 2019 como la base de un producto mayor puesto en producción para el análisis de series temporales financieras en función de series

temporales de NLP. A día de hoy ese proyecto sigue vivo y en producción, sólo que con una rescritura completa del frontal web y el uso de un proveedor privativo de documentos (Factiva).

Al quedar todo este proyecto superado por la versión actual del producto y ver que este código ya no se usaba, con todo el trabajo que hay puesto detrás, me pareció buen candidato como proyecto fin de máster, pues incluye varias tecnologías vistas en el máster y cuenta con originalidad.

En cuanto a estimación de costes se refiere, el mayor coste es el de la base de datos (en torno a los 150$), seguido del servidor REST (50$, con provisión de EBS). El gasto asociado a invocaciones Lambda cuando este proyecto estaba en producción no superaba los 15$.

Índice de figuras

Índice de tablas

Bibliografía

[1] Charu C. Aggarwal. Machine Learning for Text. 2018.

[2] Joseph Johnson. Global spam volume as percentage of total e-mail traffic from January 2014 to March 2021, by month. doi: https://www.statista.com/statistics/420391/spam-email-traffic-share/.

[3] Sebastian Nagel. Repositorio cc-webgraph. doi: https://github.com/commoncrawl/cc-webgraph.

[4] Marcos Galletero Romero. Prueba de evaluación continua de asignatura Redes sociales.

[5] John Rydning. Worldwide Global DataSphere and Global StorageSphere Structured and Unstructured Data Forecast, 2021–2025. 2021. doi: https://www.idc.com/getdoc.jsp?containerId=US47998321.

[6] Greg Corrado Tomas Mikolov, Kai Chen and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. 2013. doi: https://arxiv.org/abs/1301.3781.