Proyecto WEBIR 2015

Grupo 15

Agustín Azzinnari	4.521.581-6
Lucía González	4.513.489-4
Darío Wolman	4.383.407-8
Germán Wolman	4.383.412-5

Introducción

Google Ngram Viewer [1] es una aplicación web que permite realizar búsquedas de frases y ver el número de ocurrencias de las mismas a lo largo del tiempo. Esta búsqueda se realiza sobre el conjunto de libros digitalizados por Google (en el marco del proyecto Google Books [2]), por lo que cuenta con una inmensa cantidad de datos (con algunos libros llegando a datar del año 1500).

La idea del proyecto consistió en realizar una aplicación similar, pero basándose en artículos de noticias de Uruguay. Esto permite estudiar la evolución del uso de ciertos términos a lo largo de los meses y años, pudiendo incluso relacionar con sucesos ocurridos en el mundo (por ejemplo, observar la aparición del término “primavera árabe” en la jerga periodística). Como novedad sobre el Ngram Viewer, se agregó la capacidad de asociar picos en las menciones de ciertos términos a los eventos que los generaron.

El proyecto constó de varias etapas, entre las cuales se encuentran el scraping de diversos sitios de noticias uruguayos, el almacenamiento y procesamiento de dichos datos, y la elaboración de una herramienta de visualización de los datos obtenidos, pasos que se detallarán en las secciones siguientes.

La herramienta de visualización puede ser accedida en http://webir.ydns.eu. Allí se puede ingresar uno o varios términos de búsqueda (separados por comas), ante lo cual se desplegará el análisis provisto por la aplicación. Más adelante se detalla su uso y se muestran ejemplos de consultas.

Diseño de la Solución

En esta sección se hará una descripción general de la arquitectura de la solución, identificando los módulos claves de la aplicación con sus responsabilidades. En la sección que sigue, se pasará a detallar cómo se implementó cada uno de ellos y qué problemas surgieron en el camino.

La herramienta consta de cuatro módulos principales, cada uno construyendo sobre el resultado del anterior para llegar a la solución final. Empezando por el más básico, estos son:

Módulo de Scraping

Este módulo es el encargado de realizar el scraping de las fuentes de datos que alimentan la aplicación. Como se mencionó en la introducción, se utilizaron sitios de noticias uruguayos, de los cuáles se obtuvieron el histórico de artículos desde el inicio de cada uno de ellos. Más específicamente, los sitios scrapeados fueron: El País [3], El Observador [4], Montevideo COMM [5], La República [6], El Espectador [7], La Red 21 [8], y 180 [9].

Para cada uno de estos sitios se realizó un scraper (la implementación varía de sitio en sitio y se detalla en la próxima sección) que descarga el HTML asociado a cada artículo y realiza un análisis del DOM [10] para obtener el texto limpio (esto es, sin ningún tipo de *markup*) de las noticias.

El módulo no sólo scrapea el histórico de los sitios de noticias, sino que corre continuamente manteniendo un registro de los artículos ya scrapeados y buscando nuevos artículos publicados. De esta forma, nos aseguramos que nuestro almacén de noticias siempre tiene datos actualizados.

Módulo de Almacenamiento

Este módulo se encarga de almacenar la información recabada por los scrapers. Consiste en un motor de búsqueda que se encarga de indexar adecuadamente los artículos de noticias para su posterior búsqueda. Para ello necesita procesar los documentos acorde a las necesidades de nuestra aplicación (punto que se desarrollará más adelante) utilizando muchas de las técnicas vistas en el teórico del curso, como la utilización de índices invertidos y la realización de consultas basándose en esta información.

Módulo de Búsqueda

Este módulo se encarga de realizar las consultas necesarias sobre el almacén de documentos donde viven las noticias. Dada la naturaleza de nuestra aplicación, algunas de las consultas son más complejas, por lo que requieren una lógica más específica. Esto es, el módulo se encargará de traducir la búsqueda del usuario en una consulta que pueda entender el motor de búsqueda, con el fin de desplegar los datos que se buscan.

La segunda tarea del módulo es, en base a los resultados de la consulta (que consta de la frecuencia de cada uno de los términos a lo largo de un período de tiempo), encontrar los picos de menciones (esto es, períodos en los cuales el término apareció más de lo normal) y las noticias asociadas a los mismos (con el fin de explicar dicho fenómeno).

Módulo de Visualización

Este módulo consiste en una aplicación web que permite realizar consultas sobre nuestros datos. Cuando el usuario ingresa un término y el módulo de búsqueda devuelve los resultados, la aplicación desplegará una gráfica mostrando la evolución del término, junto a una lista de noticias asociadas a cada uno de los picos encontrados.

El principal objetivo de este módulo es presentar de una forma más amigable el resultado de todos los módulos anteriores, y permitir así utilizar la herramienta sin necesitar conocimientos técnicos.

Implementación

En esta sección se pasará a detallar la implementación de cada uno de los módulos anteriormente descritos, junto a los problemas surgidos y las decisiones tomadas durante la elaboración de la aplicación.

Módulo de Scraping

Para el scraping de artículos de noticias se utilizaron dos técnicas distintas.

Scrapy

La primera fue utilizando la herramienta Scrapy [11] para la plataforma Python [12], con la cual se extrajeron todas las noticias del diario El País. Scrapy permite definir de manera muy simple módulos denominados *spiders*, los cuales se encargarán de recorrer la página acorde a una lógica definida por el usuario y obtener la información deseada de las mismas. En el caso de El País, se realizaron dos spiders: una para obtener los datos históricos, y otra obtener noticias actuales.

Para recolectar los datos históricos se utilizó el archivo histórico de la página web del diario (accedido a través de http://www.elpais.com.uy/ediciones-anteriores/). Aquí se puede elegir una fecha desde el inicio de la edición digital y ver la portada de ese día. Partiendo de este índice, se utilizaron dos URLs adicionales:

- http://historico.elpais.com.uy/06/04/13/todoslostitulos.asp
- http://historico.elpais.com.uy/06/04/13/indexultimomomento.asp

Estas URLs despliegan una lista de todos los artículos publicados en el día, únicamente para fechas anteriores a setiembre de 2013, fecha del último rediseño de El País. Teniendo esto en cuenta, se le instruyó a la spider que para cada fecha desde 2002 hasta 2013, recorriera esas dos páginas y obtuviera una lista de noticias para scrapear.

Ahora, dado que a lo largo de los años El País ha tenido muchos cambios de diseño, lo que implicaría implementar una lógica de posprocesamiento específica a cada uno (esto es, XPaths [13] para obtener los datos limpios), se optó por utilizar la versión impresa, que ha resultado incambiada a lo largo del tiempo. Por lo tanto, con el ID de cada uno de los artículos (derivados de la URL, por ejemplo “707112” en la URL http://historico.elpais.com.uy/130405/pnacio-707112/nacional/mujica-insulto-a-los-kirchner-y-quebro-relacion-con-argentina), se construyó una URL para acceder a la versión impresa (en el caso del artículo anterior,

http://historico.elpais.com.uy/Paginas/ImprimirNota2.asp?i=707112), lo que permitió utilizar un único conjunto de XPaths para obtener el texto del artículo limpio.

Por otro lado, se construyó un spider distinto para los artículos posteriores a 2013, pues estos no tienen un ID con el cuál se pueda acceder a la versión impresa. Para estos, simplemente se accedió al listado de artículos disponibles en el archivo histórico para cada día y se recolectaron los datos. Esto implicó, por supuesto, un nuevo conjunto de XPaths para obtener el texto limpio.

Para este último spider, además, se dejó corriendo un cronjob [14] en el servidor donde corre la aplicación para que todos los días scrapee los artículos faltantes de la semana anterior. De esta forma, siempre tenemos un archivo actualizado de las noticias del diario.

Scrapeo Manual

Investigando el resto de las fuentes, notamos que todas las páginas identificaban a sus artículos por un número ID único y secuencial. Decidimos hacer uso de este punto y, en lugar de usar Scrapy, construimos un script en Python que recorre secuencialmente cada uno de estos IDs y descarga el artículo asociado, utilizando un conjunto de reglas (XPaths y selectores CSS) específicas para cada uno de los sitios.

El script cuenta de tres áreas:

- Una base de datos que se encarga de registrar las fuentes de datos disponibles, y cuáles IDs existen y fueron descargada de cada una de ellas. Para ello se utilizó el RDBMS PostgreSQL [15] y la librería Python SQLAlchemy [16] como interfaz.

- Un conjunto de módulos que encapsulan el conocimiento de cada fuente, las cuales pueden saber, a través de la página principal del portal, cuál es el último ID de las noticias (esto es, revisan cuál es la última noticia y se fijan cuál es el ID asociado). Luego proceden a crear una entrada en la base de datos por cada ID recolectado.

  También cuentan con la lógica para obtener el artículo de noticias limpio a partir del ID del mismo (primero descargarán el artículo y luego procesarán el HTML).

- Por último, un motor de scraping que se encarga de dos tareas: primero, revisar cuáles son los artículos faltantes para cada uno de los sitios de noticias (haciendo uso de los módulos del punto anterior), y luego de ir por cada una de las entradas creadas en la base de datos, y hacer que el módulo correspondiente scrapee el artículo. El resultado de esto es luego almacenado por el módulo de almacenamiento.

El script corre continuamente en el servidor donde corre la aplicación. Cuando se agotan las entradas para scrapear, espera 15 minutos y vuelve a empezar, descargando las noticias

nuevas publicadas desde el último chequeo. De esta forma, se consigue tener una base de noticias siempre actualizada.

A continuación presentamos la cantidad de artículos y palabras que se obtuvieron de cada una de las fuentes de noticias, al día en que se escribe este informe:

Fuente	URL	Artículos	Palabras
La Red 21	lr21.com.uy	542.232	228.056.073
El País	elpais.com.uy	527.407	214.892.429
El Espectador	espectador.com	285.497	102.015.460
El Observador	observador.com.uy	239.471	86.467.449
Montevideo COMM	montevideo.com.uy	225.754	65.584.296
La República	republica.com.uy	105.019	49.243.033
180	180.com.uy	49.101	21.66.516
	Total	1.974.481	767.625.833

Módulo de Almacenamiento

Para el almacenamiento de los artículos se utilizó Elasticsearch [17]. Esta herramienta es un motor de búsqueda basado en la librería Apache Lucene [18] (similar a otros motores como Solr), la cual provee funcionalidades para armar índices invertidos y realizar consultas sobre los mismos. Elasticsearch tiene la ventaja, sin embargo, de que es muy simple de configurar (a diferencia de Solr, no requiere crear un schema para los documentos de antemano), y provee una interfaz HTTP a través de la cuál se puede interactuar. Esto último permite que sea muy fácil de utilizar a partir de cualquier lenguaje de programación, o incluso desde el navegador mediante algunas herramientas adicionales [19].

Elasticsearch trabaja con documentos JSON [20] (similares a los diccionarios de JavaScript), lo cual lo hace muy conveniente para guardar información semiestructurada como son los artículos de noticias. Para cada documento, se almacenan los siguientes campos: título, contenido (todo el texto asociado, incluido el título), fecha de publicación, URL, fuente (de qué diario sale), y el ID que tenía en el sitio fuente (para prevenir duplicados).

Para la construcción del índice invertido, Elasticsearch permite definir cómo debe ser procesado el documento a través del denominado *mapping* [21]. Esto es un diccionario JSON que especifica cómo tratar a cada uno de los campos del documento. El mapping utilizado para nuestra aplicación fue el siguiente:

```
{
  "articles": {
    "properties": {
      "title": {"type": "string", "analyzer": "folding"},
      "content": {
        "type": "string",
        "analyzer": "folding",
        "fields": {
          "stemmed": {
            "type": "string",
            "analyzer": "snow_folding"
          },
          "word_count": {
            "type": "token_count",
            "store": "yes",
            "analyzer": "folding"
          },
      }
    },
    "pub_date": {"type": "date", "format": "dateOptionalTime"},
    "url": {"type": "string", "index": "not_analyzed"},
```

```
        "source": {"type": "string", "index": "not_analyzed"},
        "source_id": {"type": "string", "index": "not_analyzed"},
      }
    }
```

Este mapping se traduce en lo siguiente:

- Armar un índice invertido para el título de los artículos, pero primero pasando el texto a minúsculas y sacándole los tildes.

- Hacer lo mismo para el contenido del artículo, pero también armar un segundo índice invertido que además de pasar a minúsculas y sacarle los tildes, realice stemming de las palabras (para pasar todas las formas de los verbos a la misma representación, remover plurales, etc.).

- Almacenar la URL, la fecha de publicación, la fuente y el ID de la fuente sin procesar de ninguna forma, ni crearles índices invertidos.

- Para la fecha de publicación, almacenar como una fecha, para que se puedan realizar búsquedas temporales.

- Para el contenido de los artículos también calcular la cantidad de palabras de cada uno.

La forma de procesar el texto de los artículos fue una decisión del diseño de la aplicación que obviamente determinará su posterior funcionamiento. El remover los tildes hace que no se puedan realizar consultas del estilo “Papa vs. Papá”, pues para el motor de búsqueda ambos términos serán lo mismo. Sin embargo, decidimos hacerlo para evitar sesgar consultas comparativas cuando uno de los términos tiene tildes, pues indudablemente en los artículos aparecerá a veces con tilde y a veces sin, debido a errores ortográficos. Estos errores también podrían ser cometidos por el usuario de la herramienta, por lo que nos pareció una decisión adecuada.

Siguiendo el mismo razonamiento, decidimos almacenar una copia adicional del contenido realizándole primero stemming, y dejarle al usuario la libertad de elegir que alternativa desea. Esto lo hicimos porque puede depender de la consulta cuál es la mejor opción: si se están buscando nombres, es preferible no realizar stemming (stemming sobre un nombre propio no tiene sentido, pero el algoritmo no puede saberlo), mientras que si se buscan frases, puede ser conveniente aplicarlo (por ejemplo, buscando “refugiados sirios” queremos que cuenten instancias de “refugiado sirio”).

El tamaño final del índice terminó siendo de 6.5GB. Este número es significativamente más alto que lo que sería simplemente el peso del texto plano, pues hay una gran cantidad de metadatos que deben almacenarse (incluidos dos índices invertidos para el mismo campo).

El Elasticsearch corre en el mismo servidor que se realiza el scraping, y los artículos son inyectados por el módulo de scraping descrito en la sección anterior.

Módulo de Búsqueda

La principal tarea de este módulo es procesar las consultas realizadas por el usuario, recopilando los datos necesarios a partir de los datos almacenados en el motor de búsqueda. Esta funcionalidad se provee a través de una API HTTP RESTful, la cual cuenta con un único *endpoint*:

```
POST /search
```

```
{
   “query”: [“Tabaré Vázquez”, “José Mujica”],
   “filter”: {
       “source”: [“elpais.com.uy”, “lr21.com.uy”] | “elpais.com.uy”,
       “date_from”: “2015-01-01”,
       “date_to”: “2015-10-01”
   },
   processing”: {
       “smoothing”: 0,
       “stemming”: true|false
   }
}
```

```
{
   “pointsets”: [
       {
           “query”: “Tabaré Vázquez”,
           “points”: [
               // Puede ser un punto por año, por mes, o por día,
               // dependiendo el rango de búsqueda.
               {“key”: “2015-01-01”, “value”: 0.0003},
               {“key”: “2015-02-01”, “value”: 0.0002},
               {“key”: “2015-03-01”, “value”: 0.0001},
               {“key”: “2015-04-01”, “value”: 0.0005},
               ...
```

```
            ],
            “highlights”: [
                {
                    “title”: “...”,
                    “pub_date”: “...”,
                    “url”: “...”,
                    “source”: “...”
                },
                ...
            ]
        },
        {
            “query”: “José Mujica”,
            “points”: [
                ...
            ],
            “highlights”: [
                ...
            ]
        }
    ]
}
```

Como se puede ver arriba, en el llamado se especifican las siguientes opciones:

- Una lista de términos a consultar, que pueden ser una única palabra o una frase corta.

- Una serie de filtros que restringen el dominio de búsqueda: límites para la fecha de publicación (`date_to` y `date_from`), y un filtro por sitio de noticias (`source`).

- Información de procesamiento, que permite activar o no el stemming (detallado en la sección anterior) y la opción de aplicar *smoothing*.

El *smoothing* hace que se promedie el valor obtenido en un período dado con los N períodos a ambos costados (donde N es el valor de smoothing). De esta forma, el resultado será una media móvil del verdadero valor, haciendo que el gráfico del módulo de visualización sea un poco más estable.

El llamado devuelve, para cada uno de los términos, dos conjuntos de datos:

- El porcentaje de artículos en los que aparece el término a lo largo del período especificado. Para esto se divide dicho período en intervalos, los cuales pueden ser meses, semanas o días (qué subdivisión se utiliza se determina en base al largo del período, pero se busca tener al menos una centena de puntos), y se calcula la ocurrencia del término sobre el total de artículos en el período.

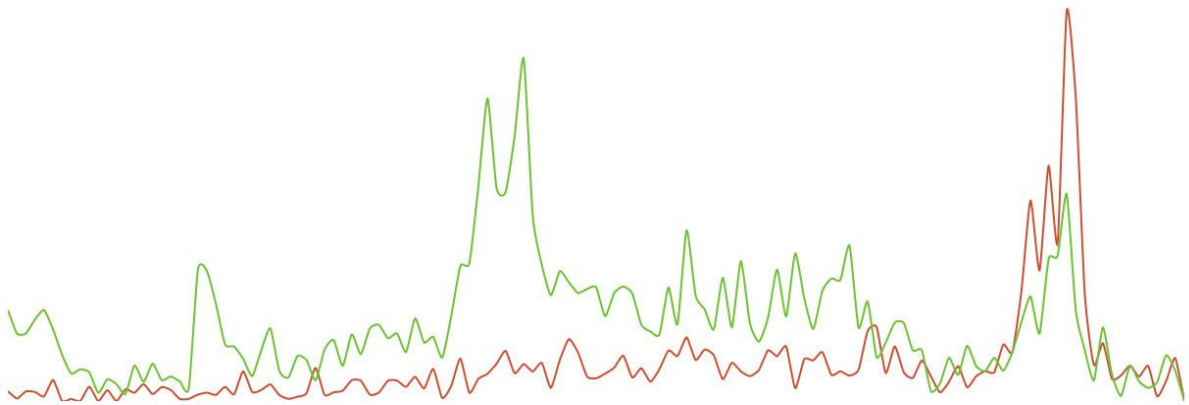
  Cabe destacar que esta no es la forma en la que lo calcula el Ngram Viewer, pues éste se fija la proporción de ngramas (no de libros) en los que aparece. Ahora, dado que los artículos de noticias son más cortos y suelen abarcar un único tema, nos pareció más adecuado utilizar este método.

  Los valores son calculados casi enteramente por Elasticsearch, mediante el empleo de aggregations [22], teniendo únicamente que elegir sobre qué campos se realiza la búsqueda (si sobre el contenido stemmizado o no), en qué rango de fechas, etc.

- Por otro lado se devuelve una serie de eventos importantes (*highlights*), una serie de fechas donde hay un pico en las menciones del término, junto a una noticia relevante a dicho pico. Esto permite darle un poco de contexto al comportamiento que muestran los datos devueltos. El procedimiento para encontrarlos se detalla a continuación.

Highlights

Durante la implementación inicial de la aplicación, notamos que muchos de los términos que buscábamos tenían picos de menciones en el período de la búsqueda. Por ejemplo, cuando se realiza la consulta “Lacalle Pou, Bordaberry” entre 2002 y 2015, se observa el siguiente comportamiento (con el término “Bordaberry” en verde y “Lacalle Pou” en rojo):



Se puede deducir fácilmente que los dos grandes picos corresponden a las campañas electorales de las elecciones presidenciales de 2009 y 2014. Sin embargo, no es tan fácil saber a qué se debe el primer pico de Bordaberry cerca de 2006. Con el fin de poder dar

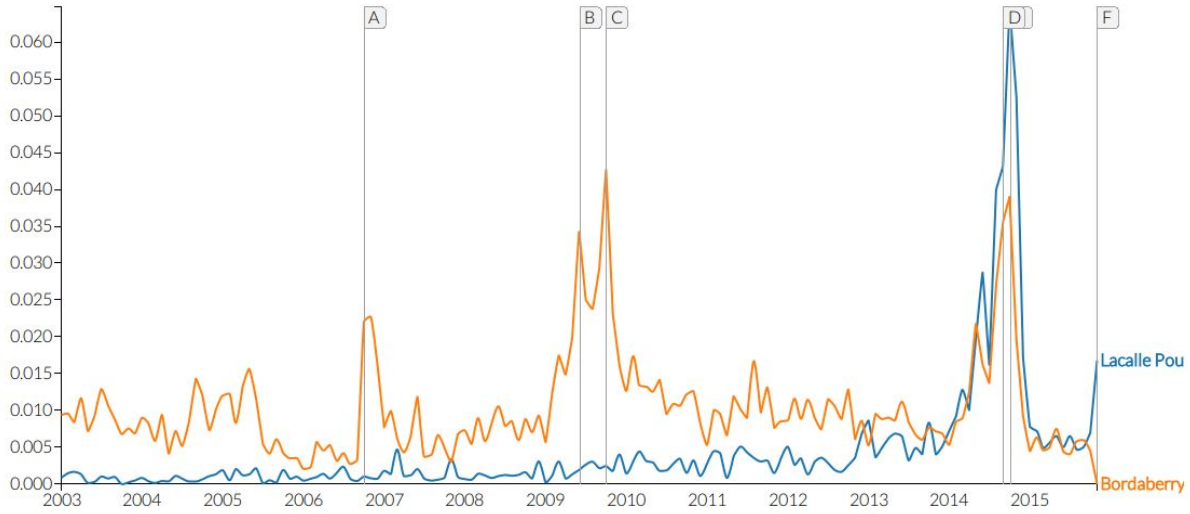
contexto a los resultados, decidimos realizar un procedimiento para identificar automáticamente estos picos.

El procedimiento consta de dos etapas:

- Primero, claro está, la identificación de los picos de menciones. Para ello se utilizó una sugerencia vista en un artículo [23] en Wolfram Community, la cual se basa en encontrar puntos que se separan de la media móvil para una ventana de largo N más de una cierta proporción de la desviación estándar.

  Esto es: para cada punto se calcula la media en una ventana de largo `N` (utilizamos un valor de `N=3`) y nos fijamos si el valor en el punto difiere más de `T` veces la desviación estándar de la media previamente calculada (utilizamos un valor de `T=1.3`). De esta forma encontramos puntos donde hay muchas más menciones de lo normal, pero evitando detectar demasiados picos para los términos que tienen demasiada varianza.

  En el gráfico anterior, los picos detectados serían:



- Una vez encontrado el pico, se intenta buscar una noticia representativa de los eventos que llevaron al aumento de menciones, para lo cuál se utilizaron técnicas de procesamiento de lenguaje natural.

  Más específicamente, primero se identificó cuál fue el día con más menciones y se obtuvieron todos los títulos de las noticias publicadas en esa fecha donde aparece el término. Luego se vectorizaron los títulos usando una representación dispersa con la ocurrencia de las 25 palabras más populares (excluyendo stopwords), y se calculó el centroide del conjunto de vectores. Por último, se devolvió el artículo más cercano al centroide utilizando la distancia coseno.

A partir de este método, pudimos saber que el pico A en el gráfico anterior correspondía al procesamiento de Juan María Bordaberry [27].

El servidor web que implementa esta API corre en el mismo servidor que los dos anteriores módulos.

Módulo de Visualización

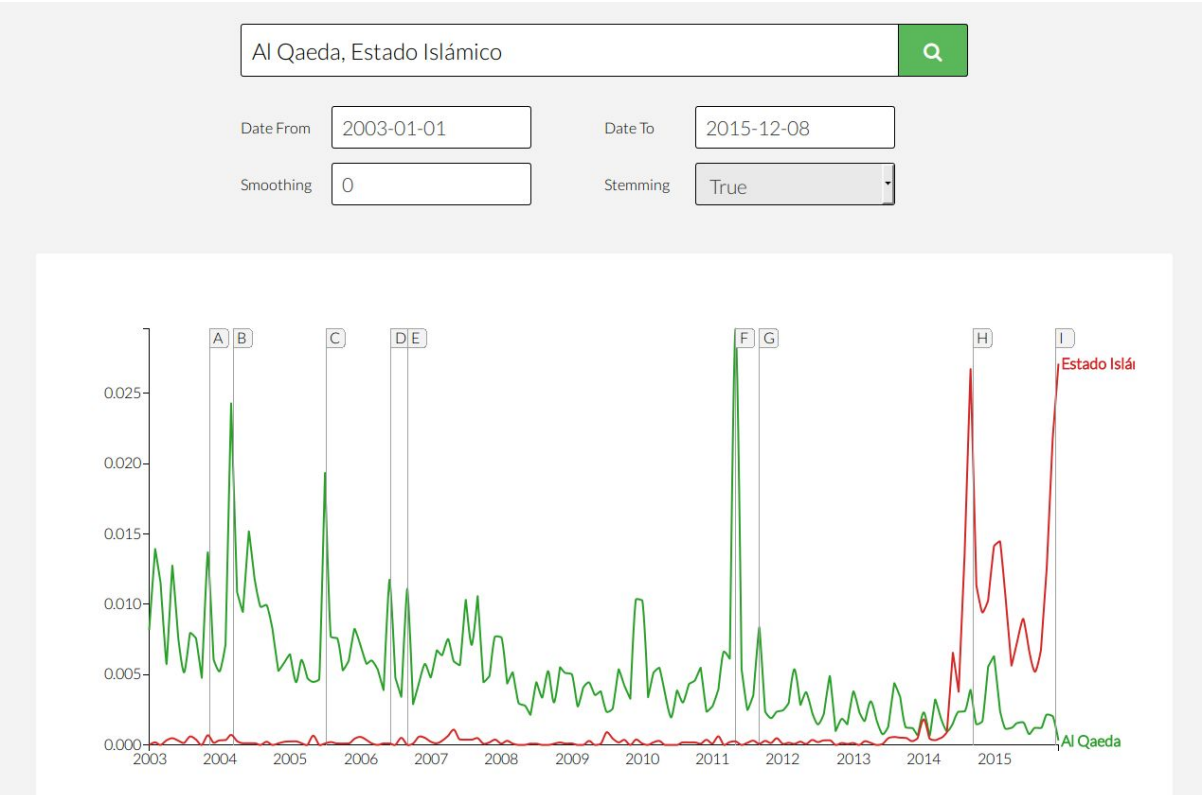
Para poder visualizar los resultados de nuestro análisis, se construyó una aplicación web que, utilizando la API HTTP descrita en el punto anterior, permite al usuario realizar consultas de manera muy simple. La aplicación fue escrita utilizando tecnologías web (HTML, Javascript y CSS), con el apoyo de la librería JQuery [24] para la comunicación con la API y la librería D3.js [25] para la generación de una gráfica a partir de los datos obtenidos.

La herramienta le provee al usuario de una barra de búsqueda donde se ingresan los términos a consultar, junto con un formulario para especificar las opciones avanzadas (el rango de fechas a buscar, si aplicar stemming, y el nivel de smoothing a utilizar). Una vez realizada la consulta, se despliega una gráfica mostrando la evolución del uso del término en el período especificado.

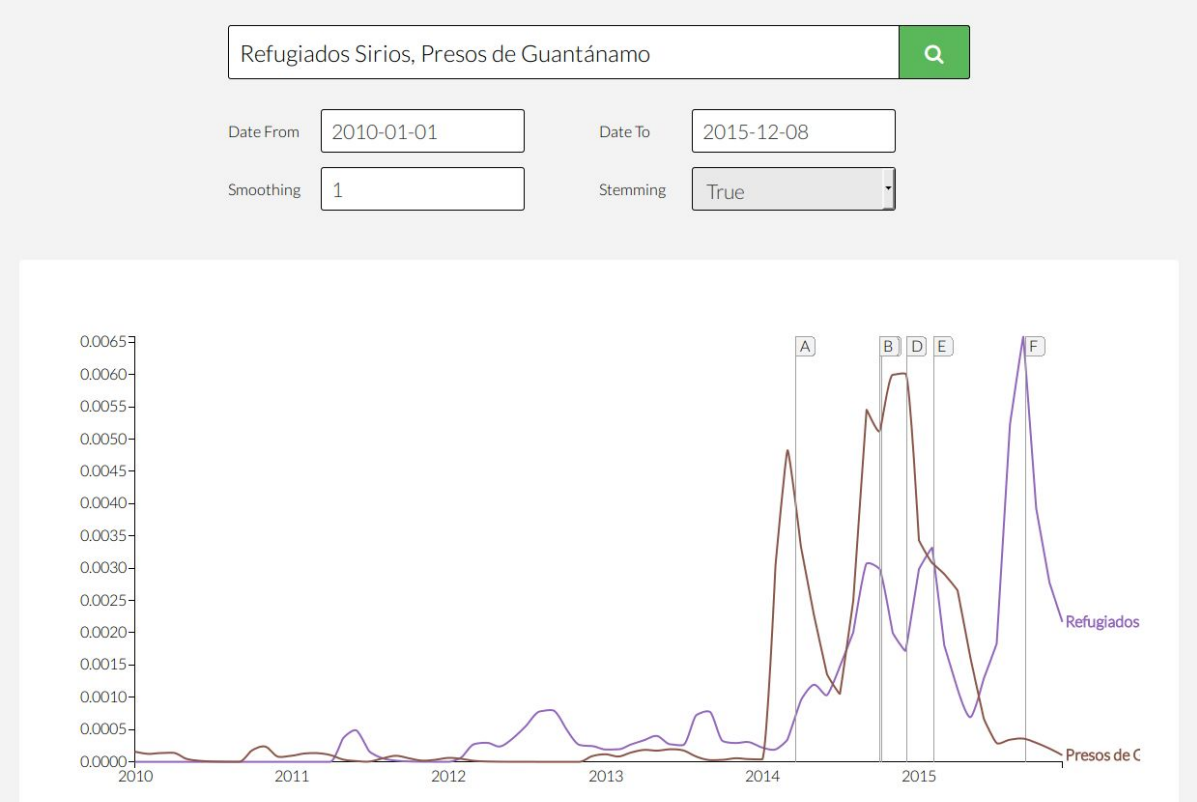
La gráfica cuenta a su vez con marcadores indicando los picos de menciones y una referencia a la noticia asociada al mismo. Bajo la gráfica se listan todas las noticias, junto con links para acceder a las mismas en el sitio de noticias original.

A continuación se muestran algunos ejemplos de búsquedas a través de la interfaz:

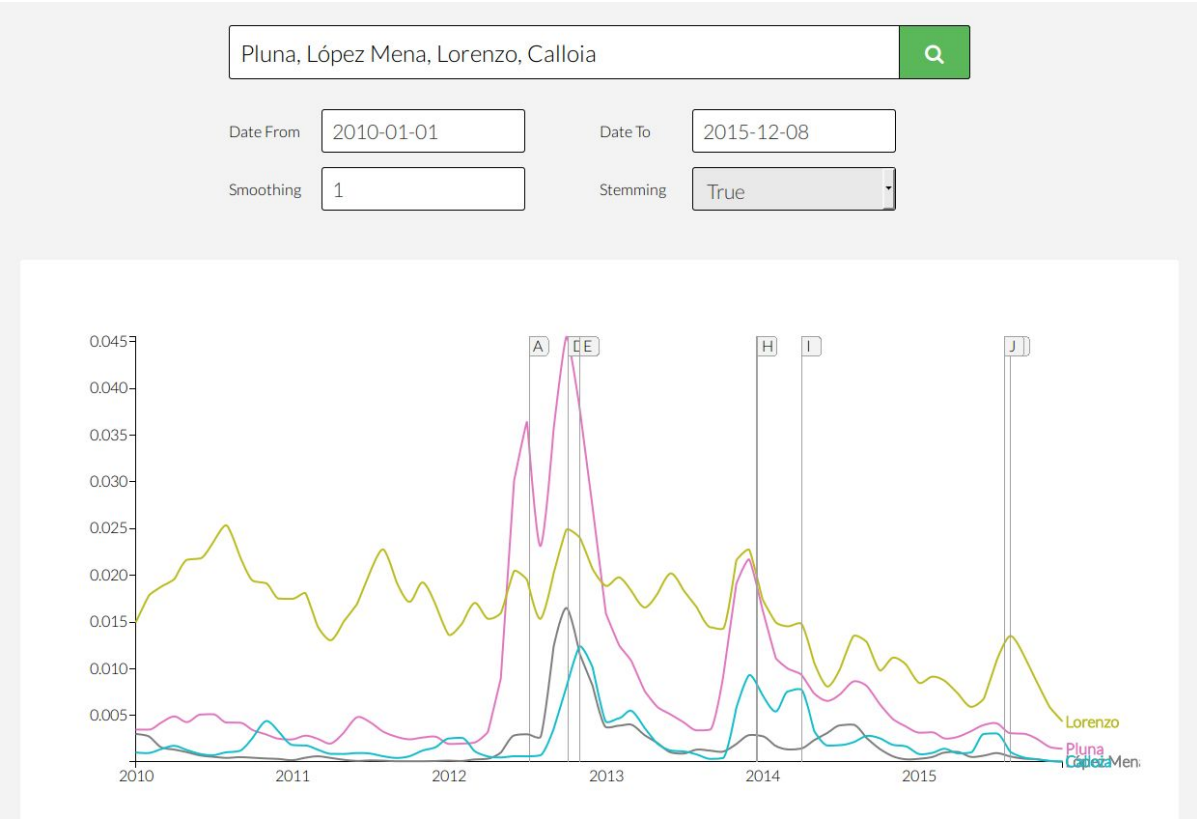
Al Qaeda, Estado Islámico



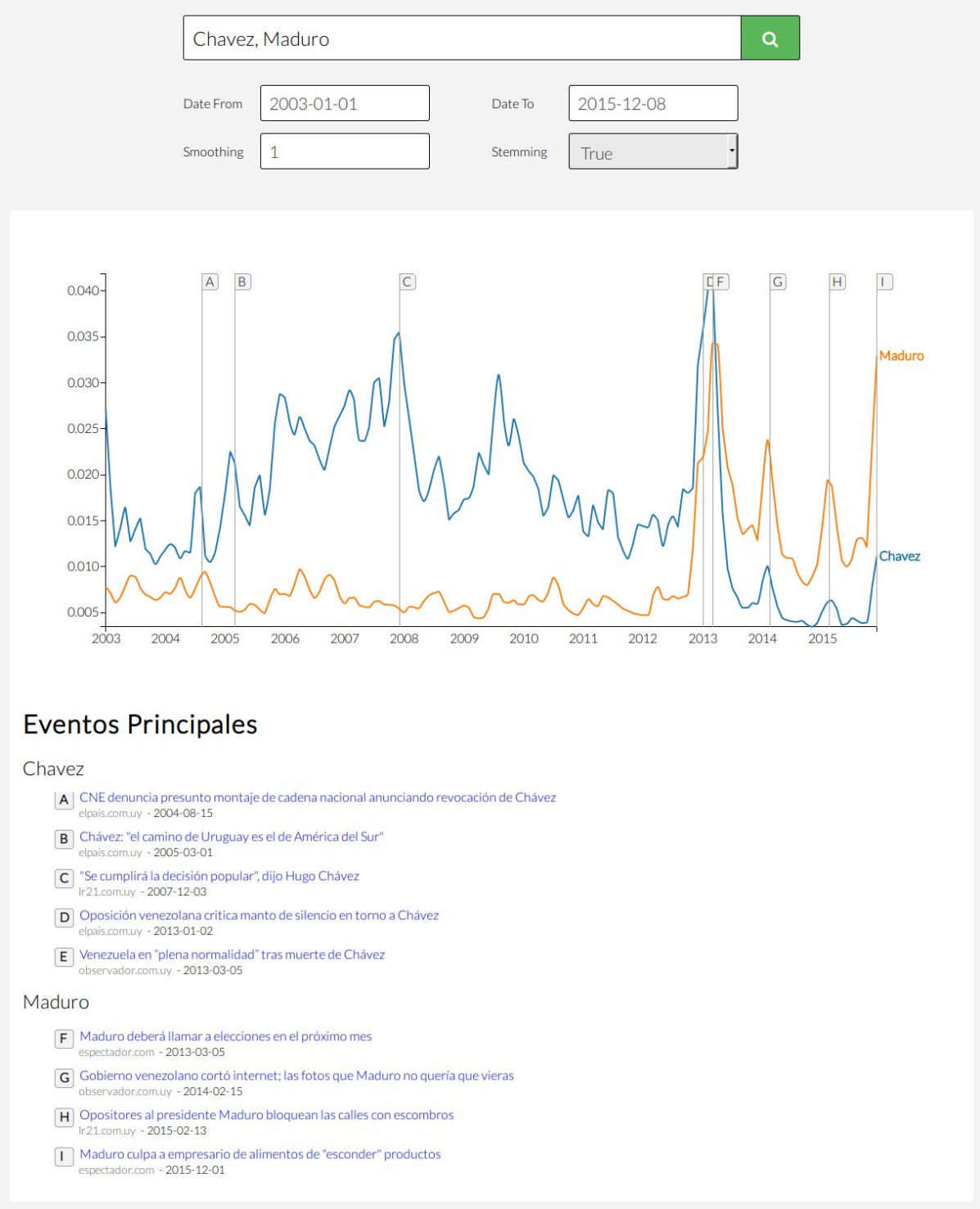
Refugiados Sirios, Presos de Guantánamo



Pluna, López Mena, Lorenzo, Calloia



Chavez, Maduro



La herramienta web se encuentra en el mismo servidor que los módulos anteriores, y puede ser accedida a través de la siguiente URL: http://webir.ydns.eu.

Conclusiones

La cantidad de información que hay en la web tiene mucho potencial, pero por lo general tiene el problema de carecer de estructura, tratándose principalmente de texto libre. A través de este proyecto, sin embargo, vimos como aún así es posible destilar datos estadísticos interesantes, mediante herramientas de Recuperación de Información y Procesamiento de Lenguaje natural.

Esta última tarea viene de la mano del Web Scraping. Como pudimos apreciar durante la elaboración de la tarea, hay muy pocos estándares acerca de la disposición de contenido en una página, y los que hay muy pocos sitios los siguen (como la utilización de microdata de HTML [26]). Esto hace que la recolección de información limpia (sin markup) pueda tornarse muy laboriosa, porque implica generar reglas para cada sitio objetivo. Existen diversas técnicas que mediante heurísticas logran extraer texto automáticamente, como jusText [27], pero corren el peligro de introducir ruido.

Otro punto interesante fue la utilización de motores de búsqueda, que resultaron muy útiles para almacenar documentos de texto. La simplicidad de uso de Elasticsearch, y la cantidad de funcionalidad que ofrece con mínima configuración, nos fue de gran utilidad a la hora de analizar los artículos recolectados. Puede resultar útil incluso hacer uso de estas herramientas para hacer un análisis preliminar de un corpus de datos.

Aún así, es importante prestar atención a cómo procesar los datos antes de guardarlos: decisiones de diseño como la utilización de stemming y la eliminación de tildes tienen que ser decididos de antemano y pueden afectar mucho la performance del sistema (principalmente el uso de memoria y almacenamiento), y dependerán del sistema que se está implementando.

Trabajo Futuro

Identificamos algunos puntos sobre los cuales consideramos que sería interesante seguir trabajando:

- Mejorar el cálculo de los highlights. El cálculo de la noticia más relevante que se hace actualmente es muy básico, y suele obtener noticias que, por más que son relevantes al término, no son relevantes al evento que causó el pico de menciones.

  Por ejemplo, si se realiza la consulta “Chávez”, la noticia asociada al pico generado por su muerte se titula “Venezuela en plena normalidad tras muerte de Chávez” que, por más que es relevante al evento, no fue lo que desató el pico. Idealmente se buscaría una noticia que anuncie la muerte.

- Agregar más opciones de filtrado. Se podrían etiquetar los artículos de noticias según su categoría y restringir la búsqueda a e.g. noticias de deportes. También puede resultar interesante poder agrupar los resultados por sitio de noticias, para poder ver qué diario habla más de qué cosa.

- Agregar más fuentes de noticias, principalmente de otros países, sería una mejora interesante, pues permitiría ver la importancia que se le dio a distintos eventos en distintos países.

- Por último sería muy útil construir un sistema de alertas para notificar cuando un spider deja de funcionar. Actualmente contamos con siete spiders, pero si se agregaran más fuentes de datos, puede ocurrir que por un rediseño de un sitio una spider deje de funcionar y que esto pase desapercibido.

Referencias

[1] - https://books.google.com/ngrams
[2] - http://www.google.com/intl/en/googlebooks/about/index.html
[3] - http://elpais.com.uy
[4] - http://observador.com.uy
[5] - http://montevideo.com.uy
[6] - http://republica.com.uy
[7] - http://espectador.com
[8] - http://lr21.com.uy
[9] - http://180.com.uy
[10] - https://en.wikipedia.org/wiki/Document_Object_Model
[11] - http://scrapy.org
[12] - http://python.org
[13] - http://en.wikipedia.org/wiki/XPath
[14] - http://en.wikipedia.org/wiki/cronjob
[15] - http://postgresql.org
[16] - http://postgresql.org
[17] - http://elastic.co
[18] - http://lucene.apache.org/
[19] - https://chrome.google.com/webstore/detail/sense-beta/lhjgkmllcaadmopgmanpapmpjgmfcfig
[20] - http://en.wikipedia.org/wiki/JSON
[21] - https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html
[22] - http://community.wolfram.com/groups/-/m/t/96823
[23] - http://jquery.com
[24] - http://d3js.org/
[25] - http://www.w3.org/TR/2009/WD-html5-20090825/microdata.html
[26] - https://code.google.com/p/justext/
[27] - http://www.espectador.com/sociedad/79752/defensa-de-bordaberry-pidio-citacion-de-fernandez