UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

título del TFG

Presentado por Nombre del alumno
en Universidad de Burgos — 6 de junio de 2017
Tutor: nombre tutor

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Nombre del alumno, con DNI dni, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 6 de junio de 2017

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. nombre tutor	D. nombre co-tutor

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice de figuras

Índice de tablas

Introducción

Descripción del contenido del trabajo y del estrucutra de la memoria y del resto de materiales entregados.

Objetivos del proyecto

2.1. Objetivos basados en los requisitos funcionales

Los siguientes objetivos se corresponden con las funcionalidades que debería satisfacer la aplicación:

- Almacenar noticias de distintas páginas web con información esencial: autor, fecha, titular, texto... etc.
- Leer noticias que se adjunten en un formato PDF en un directorio concreto, cuyo contenido tenga un formato prefijado para facilitar la lectura.
- Poder asignar una clase a estas noticias respecto a un dataset, ya sea de forma manual, o por medio de una predicción realizada con minería de datos.
- La posibilidad de incluir un dataset entero de noticias mediante un archivo XML, con etiquetas ya añadidas, por si el usuario quiere añadir un corpus de entrenamiento directamente.
- Mostrar una explicación de la predicción que se realice de las noticias, de modo que el usuario sea capaz de observar cuáles han sido los criterios para sugerir una clase u otra.
- Añadir nuevos datasets de forma manual, y clasificar las noticias existentes para esos nuevos datasets.

2.2. Objetivos técnicos

Los siguientes objetivos están relacionados con los aspectos técnicos que la aplicación debe cumplir:

- La aplicación debe poder funcionar, al menos en un despliegue local, en el ordenador del usuario.
- La aplicación en general debe tener un rendimiento óptimo, aunque en ciertos puntos claves los tiempos de carga puedan sufrir aumentos.
- Al ser una aplicación web, se espera que funcione correctamente en los navegadores web más populares, como pueden ser Mozilla Firefox o Google Chrome.
- Reducir todo lo posible el proceso de configuración y ejecución de la aplicación al exportarla a nuevos equipos.

2.3. Objetivos personales

Los siguientes puntos resumen los objetivos del alumno respecto de este proyecto:

- Poder trabajar en un proyecto con mayor envergadura de lo habitual siguiendo una metodología de trabajo ágil.
- Adquirir conocimientos relacionados con el machine learning y minería de datos sobre texto. Especialmente, poder trabajar con ejemplos reales de análisis de textos (etiquetado de noticias).
- Aprender a manejar bases de datos NoSQL, como MongoDB.
- Mejorar las capacidades y experiencia de programación con Python, y utilizar librerías de terceros muy populares en la minería de datos.
- Aprender a crear aplicaciones web usando Python como lenguaje backend, en vez de otras opciones más convencionales como C# o PHP.
- Experimentar con herramientas como Jupyter, que tienen una creciente popularidad, ya que son entornos alternativos de programación interesantes y prácticos.

Conceptos teóricos

A continuación se explicarán los conceptos de carácter teórico más relevantes del proyecto, de modo que se puedan comprender con más exactitud aquellas partes con una mayor complejidad.

3.1. Minería de Datos

En los siguientes apartados se van a desarrollar todos los conceptos del proyecto relacionados con la minería de datos, esenciales en la parte de predicción del perfil ideológico de las noticias.

Bag of Words

Bag of Words es una técnica de agrupación de palabras relacionada con el Procesamiento de Lenguaje Natural. Consiste en, a partir de un texto de muestra, almacenar la frecuencia de aparición de cada palabra distinta que esté contenida en el texto, independientemente del orden de aparición, y sin tener en cuenta el contexto en el que aparezca.

En términos de estructuras de datos, un bag of words se suele almacenar con un formato de diccionario, donde la clave está formada por cada palabra distinta, y el valor es la frecuencia de aparición de esa palabra.

Vamos a ilustrar lo explicado con un ejemplo. A continuación, mostramos unas frases de ejemplo:

- "No se puede fumar en establecimientos públicos"
- "Se puede fumar en residencias privadas"
- .En general, no se debería fumar"

Palabras	Frecuencia
No	2
Se	3
Puede	2
Fumar	3
En	3
Establecimientos	1
Públicos	1
Residencias	1
Privadas	1
General	1
Debería	1

Tabla 3.1: Frecuencias de palabras en las frases de ejemplo

Teniendo estas frases, si almacenamos la frecuencia de las palabras distintas para cada una de ellas, y las combinamos, tendríamos una tabla como la siguiente:

El potencial de esta técnica para análisis de texto es muy interesante. Por ejemplo, y por relacionarlo con este proyecto, se podría asociar una etiqueta a cada una de las frases propuestas, como .A favor de los fumadoresƶ .En contra de los fumadores"; para que una vez realizado el bag of words se pueda, entre otras funcionalidades, deducir las probabilidades que tiene cada una de las palabras del vocabulario obtenido de pertenecer a una de las dos clases.

De forma indirecta, lo previamente mencionado se puede usar para predecir cuál sería la ideología de una frase nueva, en función de las palabras de la misma que coinciden con el vocabulario ya almacenado por los clasificadores.

Vectorizador de texto

Herramientas que convierten un documento de texto en matrices de tokens, que usualmente son palabras [**?**]. Son muy útiles para aplicar la técnica del bag of words a cualquier texto sin tener que recurrir a un conteo manual.

El tamaño de la matriz suele corresponderse con el tamaño del vocabulario obtenido a partir de la extracción de palabras del texto, aunque en la mayoría de herramientas el propio programador puede elegir un límite fijo para el tamaño.

En términos comerciales, los vectorizadores más usados son los de imágenes, los cuales permiten transformar una imagen en formatos más flexibles para poder manejar las mismas o exportarlas con facilidad. No obstante, en

contextos de investigación y análisis de textos, estas herramientas son especialmente prácticas.

Bagging

También conocido como Bootstrap Agreggation, es una técnica de minería de datos cuyo objetivo es el de reducir todo lo posible la varianza en la clasificación de datos. Consiste en obtener una predicción por votación a partir del promedio de los resultados obtenidos aplicando un gran número de clasificadores a distintas partes de los datos de entrenamiento.

De forma breve, podríamos explicar el proceso que realiza el Bagging en los siguientes pasos:

1. A partir del conjunto de entrenamiento, se obtienen m subconjuntos distintos, llamados "bags", que contienen combinaciones aleatorias de instancias del conjunto de entrenamiento.
2. Para cada "bag", se entrena un modelo con un clasificador distinto.
3. Todos los modelos resultantes se usan posteriormente para predecir el conjunto de prueba.
4. Para saber el resultado final, basta con saber cuál fue la clase que obtuvo más predicciones de todas las que se realizaron.

Ésta técnica se usa mucho en los casos de predicciones basadas en árboles, como es el caso de los bosques aleatorios, que explicaremos a continuación. En la siguiente imagen se pueden observar de forma sencilla los pasos previamente mencionados.

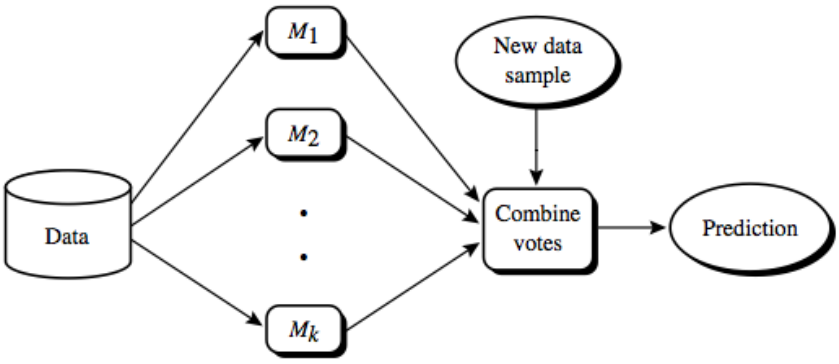

Figura 3.1: Esquema de funcionamiento básico de Bagging

Clasificador de bosque aleatorio

El clasificador de Bosque aleatorio es una técnica derivada del bagging aplicada a árboles de predicción. Como su nombre sugiere, se basa en el uso de la combinación de varios árboles de clasificación.

Cada uno de estos árboles se encarga de entrenar una instancia de los datos de entrenamiento, y la solución devuelta será la clase obtenida por votación de todos los árboles.

El principal objetivo de realizar este proceso es el de, mediante los subconjuntos aleatorios, aumentar todo lo posible la precisión y controlar el sobreajuste de los resultados. Por ese motivo, si tenemos un mismo conjunto de entrenamiento para todas las pruebas, se obtendrá un menor margen de error en aquellas clasificaciones que tengan un mayor número de árboles, mientras que los resultados con un número mínimo de árboles no serán tan fiables.

En la siguiente imagen se muestra un esquema de funcionamiento del bosque aleatorio, en la que se puede observar las similaridades con el proceso de Bagging explicado anteriormente.

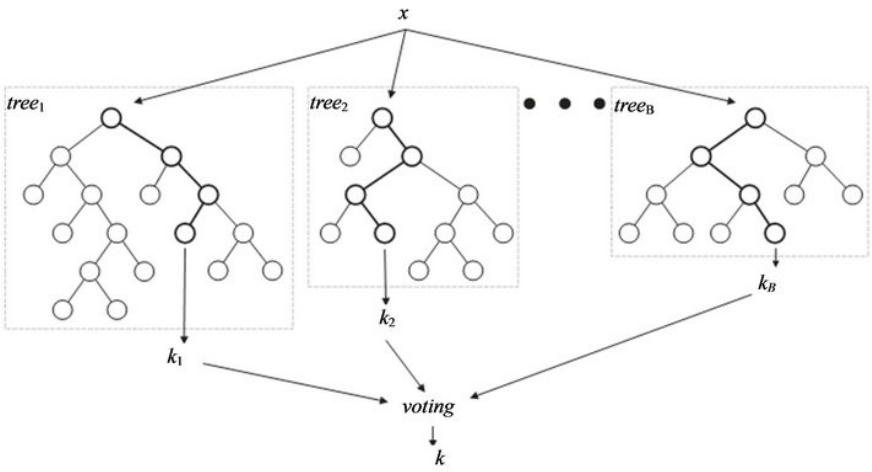

Figura 3.2: Esquema de funcionamiento de bosque aleatorio

3.2. Web Scraping

El Web Scraping es una técnica, principalmente alternativa al uso de APIs de los propietarios de páginas web, para extraer información relevante de las propias páginas web.

De forma más concreta, lo que realiza un web scraper es la detección de información que se encuentra dentro de etiquetas concretas de un documento

de tipo HTML, y su posterior extracción al indicarle las etiquetas en las que buscar. Es una técnica con un gran potencial, pues le evita al usuario la búsqueda manual de esta información, lo cual es una tarea especialmente ardua en los complejos códigos HTML de la mayoría de páginas web.

A continuación mostramos un ejemplo sencillo de Web Scraping escrito en Python usando urllib2 y BeautifulSoup, en el que obtenemos el número de descargas de un repositorio en SourceForge:

1. Hacemos una petición http con la url indicada:

```
#Indicamos la url:
url = "http://sourceforge.net/projects/sevenzip/files/stats/timeline"

#Creamos la request
request = urllib2.Request(url)

#Abrimos el contenido de la página por medio de la petición
handle = urllib2.urlopen(request)
```

Figura 3.3: Petición http

2. Leemos el contenido de la url y se la pasamos al scraper:

```
#Leemos el contenido
content = handle.read()

#Le pasamos al scraper el texto con el contenido de la página
soup = BeautifulSoup(content)
```

Figura 3.4: Lectura del contenido de la web

3. A través del scraper, accedemos al texto de la etiqueta que queramos directamente:

```
#Accedemos a la etiqueta que queremos y sacamos el texto
downloadsnum = soup.find('td', {'headers':'files_downloads_h'}).text
```

Figura 3.5: Acceso a la información deseada con el scraper

3.3. Referencias

Las referencias se incluyen en el texto usando cite [**?**]. Para citar webs, artículos o libros [**?**].

Técnicas y herramientas

Esta parte de la memoria tiene como objetivo presentar las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto. Si se han estudiado diferentes alternativas de metodologías, herramientas, bibliotecas se puede hacer un resumen de los aspectos más destacados de cada alternativa, incluyendo comparativas entre las distintas opciones y una justificación de las elecciones realizadas. No se pretende que este apartado se convierta en un capítulo de un libro dedicado a cada una de las alternativas, sino comentar los aspectos más destacados de cada opción, con un repaso somero a los fundamentos esenciales y referencias bibliográficas para que el lector pueda ampliar su conocimiento sobre el tema.

4.1. Herramientas Candidatas (Aplicación Web)

Este apartado detallará las herramientas que inicialmente fueron consideradas para la programación de la aplicación web de forma compatible con el uso de Python.

Los principales criterios a tener en cuenta para quedarnos con una de las alternativas han sido: la flexibilidad que ofreciera esa herramienta para trabajar con archivos de tipo PDF, la facilidad de distribución de la propia aplicación y el hecho de que adaptarse a ella no suponga un esfuerzo excesivo al programador.

PyQt

Permite crear aplicaciones de escritorio de forma sencilla, pudiendo crear elementos de interfaz sencillos como botones, cuadros de texto... etc(más aún si se usa su editor de GUI propio).

No obstante, las opciones gráficas no eran tan atractivas como podía parecer en un primer momento, y además no es adecuada si la intención es crear

una aplicación web.

Kivy

Es una alternativa interesante ya que permite hacer aplicaciones de escritorio con código Python con la ventaja de que es Çross-platform", es decir, que las aplicaciones que se creen usando esta herramienta se pueden distribuir en distintos sistemas operativos. No obstante, no se tuvo especialmente en cuenta ya que para cuando se estaba investigando esta opción, la opción de hacer una aplicación web en vez de una de escritorio estaba prácticamente decidida.

Flask

Permite combinar código HTML y Python de forma simple. La caraterística que más nos interesaba es su facilidad de combinar ficheros HTML y códigos escritos puramente en Python, además de poder incluir de forma sencilla librerías con estilos para la interfaz. No obstante, a la hora de practicar con la herramienta (siguiendo el mega-tutorial de miguelgrinberg.com), la estructura de ficheros daba numerosos problemas al intentar ejecutar aplicaciones, unas veces relacionadas con el intérprete de Python, y otras por errores al combinar los ficheros. Aunque finalmente se completó la guía, las sensaciones finales era que con Jupyter se desarrollaban aplicaciones sencillas con más rapidez.

Jupyter

En esta herramienta, y con ayuda de alguna librería de terceros (wand, display...) es especialmente sencillo combinar código HTML y Python en un mismo fichero. De hecho, haciendo uso de los llamados ïpywidgets"(o dicho de otra manera, los Widgets que ofrece IPython), se puede encapsular el código HTML de forma muy cómoda en un script normal de Python, especialmente el ipywidget HTML, que permite introducir una estructura HTML como si se tratara de una cadena de texto normal.

Además, la curva de aprendizaje ha sido bastante sencilla y conseguir un código inicial que muestre documentos PDF ha llevado poco tiempo. Las posibilidades de combinarlo con otros elementos, como Flask o MongoDB (PyMongo) hacen de esta herramienta una de las mejores alternativas. De hecho, fue la elegida para implementar la aplicación web.

4.2. Herramientas candidatas (Bases de Datos de texto)

De forma similar al apartado previo, en las fases iniciales del proyecto se

planteaban varias opciones para elegir cómo almacenar los datos necesarios para que el usuario pudiera obtener noticias y a la vez poder acceder a los datos para realizar la minería de los mismos.

PyLucene

PyMongo

Es la extensión de MongoDB para Python. Su principal característica es que es una base de datos no relacional, o en otras palabras, no SQL. De modo que funciona con un modelo distinto al que es más habitual ver en una base de datos convencional. Por ejemplo, en vez de estructurarse como tablas con relaciones entre ellas, la forma de almacenar los datos se asemeja mucho más a lo que se puede ver en JSON: los datos se almacenan en una estructura clave-valor similar a un diccionario. No obstante, aunque no se puedan usar relaciones, estas estructuras son muy flexibles, y se puede, por ejemplo, introducir unas estructuras como campos de otras, o introducir cualquier tipo de dato como valor.

SQLAlchemy

4.3. BeautifulSoup

https://www.crummy.com/software/BeautifulSoup/

BeautifulSoup es una librería de Python cuyo principal objetivo es facilitar la lectura y manipulación de documentos tipo XML. Uno de sus usos más frecuentes, y para el que ha sido utilizada en este proyecto, es para poder navegar por el texto de un documento HTML y leer su texto de forma muy sencilla cuando se están realizando labores de Web Scrapping.

Cuando recibe un documento XML o HTML, parsea la estructura de árbol de los mismos, de forma que permite acceder a los contenidos de la etiqueta que se desee con un simple método, y evitando al programador tener que buscar manualmente a lo largo del documento obtenido, que puede llegar a ser una tarea tediosa. Además, tiene operaciones muy interesantes como la de sacar todo el texto plano de un documento, de forma que nos ahorramos tener que iterar por toda la estructura separando etiquetas de texto.

En este proyecto, BeautifulSoup se usó en combinación con Urllib2, una librería que permite hacer peticiones que abren URLs, de modo que con la segunda obteníamos todo el código HTML de la página web a la que queríamos acceder, y con la primera parseamos el documento filtrando todo el texto de los artículos.

4.4. Sci-Kit Learn

4.5. feedparser

4.6. TextBlob

https://textblob.readthedocs.io/en/dev/

Librería Python usada para el procesamiento de datos de texto. Se centra en el procesamiento de lenguaje natural (NLP). Sus principales funciones consisten en reconocimiento de lenguaje y traducción de unos idiomas a otros, extracción de palabras y frases, análisis de emociones en una frase, clasificación de textos... etc.

En el proyecto se utilizó para organizar el texto extraído previamente con BeautifulSoup en frases, que se dividían en las dos posibles clases disponibles para realizar posteriormente un entrenamiento de un conjunto y clasificación de las mismas. Inicialmente usamos el clasificador NaiveBayes que incluía TextBlob para las primeras pruebas de clasificación, pero a medida que avanzó el proyecto se decidió pasar al uso de Sci-Kit Learn, por la mayor gama de opciones para manipular los datos obtenidos, y para un uso en combinación con Lime.

4.7. Lime

4.8. PDFMiner

Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

5.1. Experimentación en Jupyter

Debido a los problemas que se mencionan a continuación, la opción de implementar la aplicación web basada en notebooks de Jupyter tuvo que ser descartada en pleno desarrollo del proyecto. No obstante, los primeros meses del mismo fueron dedicados casi completamente a la experimentación de funcionalidades en esta plataforma. De hecho, en el repositorio del proyecto se encuentra un grupo de notebooks que no llegaron a formar parte del producto final, pero que a su vez contienen funcionalidad avanzada que tampoco pudo incluirse en la aplicación de flask. Los ejemplos más interesantes son los

siguientes:

- Web Scraping avanzado: Antes de enfocar el web scraping en los RSS de los medios de comunicación. Se realizaron dos prototipos de web scraping convencional en la página web de los periódicos Público y ElDiario.

  Cabe destacar que este web scrapping estaba más perfeccionado que el que hay en el código final, pero al mismo tiempo estaba demasiado personalizado para cada medio en concreto, y no podía replicarse con facilidad para otros portales web. En la versión final, se pueden añadir más medios con menos esfuerzos, a costa de sacrificar calidad en el texto extraído de cada noticia.

5.2. Intento de publicación de Jupyter Dashboard

Uno de los objetivos iniciales que se comentaron al comenzar el proyecto consistía en, aparte de conseguir implementar la funcionalidad y la interfaz de usuario en un notebook de Jupyter, poder publicar estos archivos en un formato de aplicación web usando unas librerías llamadas Jupyter Dashboards. La finalidad de estas librerías es la de, teniendo un notebook con la funcionalidad completa, quitar de la pantalla todos los cuadros de código, dejando solo el .ºutputz los elementos interactivos a la vista; ideal para una situación como la que se nos presentaba, en la que la aplicación la iban a usar otros usuarios.

No obstante, los requisitos para poder realizar esta publicación fueron complicándose hasta el punto de tener que descartar esta posibilidad. En lo referente al formato de la salida por pantalla, si que se consiguió configurar el notebook para que tuviera este aspecto de dashboard, pero todos los problemas estaban relacionados con la publicación del mismo. Por un lado, era necesario configurar una máquina virtual Docker que ejecutara los comandos necesarios para instalar y configurar todos los elementos que requería este tipo de aplicación.

Para realizar este proceso, y debido a lo prematuro de estas herramientas (Muchos de los problemas que sufrimos aparecen como issues sin resolver en los respectivos repositorios de GitHub), la opción más viable era seguir el único repositorio de GitHub con guías de despliegue de dashboards (https://github.com/jupyter-incubator/dashboards_setup) que los propios autores habían facilitado. De estas guías, tres necesitaban de docker para funcionar y la cuarta recurría a CloudFoundry. Mientras que CloudFoundry se descartó porque requería de versiones de pago o versiones de prueba de alguna de las plataformas certificadas que usaban esta tecnología, en el caso de Docker las complicaciones eran de otra naturaleza.

Debido a las condiciones del equipo en el que se realizó el proyecto (Sistema Operativo Windows con Hyper-V), no se podían configurar máquinas

docker desde la máquina virtual, porque las máquinas virtuales con Linux que estábamos usando(en este caso, de Oracle VirtualBox) no funcionan con Hyper-V, y las máquinas virtuales de docker necesitaban crearse con Hyper-V activado. Ante la obligación de configurar todo desde el Sistema Operativo anfitrión, un conjunto de incompatibilidades y errores sin aparente solución entre Windows y estas máquinas virtuales, hizo que los recursos necesarios para solventar esta situación fueran demasiado grandes para las limitaciones de tiempo del proyecto.

5.3. Omisión de comentarios en el Scrapping

Una de las funcionalidades que se pretendía incluir en el producto, era la lectura y clasificación por separado de los comentarios de las noticias buscadas. El principal motivo era que los comentarios de una noticia no tenían por qué tener la misma ideología que la noticia, y de hecho muchos de ellos podrían ser de crítica de la propia noticia, aunque se suponga que la mayoría de lectores de un diario en concreto probablemente tenga la misma línea ideológica que el propio medio.

Para realizar este scrapping por separado, se probaron diferentes alternativas, aunque ninguna era muy compatible con las técnicas de Web Scrapping que se usaban para leer el resto de la página y, en resumen, provocó la aparición de dos inconvenientes importantes:

Uso de Selenium para comentarios ocultos

En la mayoría de medios en los que se realiza la lectura de noticias, surgió el problema de que al hacer web scrapping con urllib2, no aparecían los comentarios en el código HTML que llegaba al código del programa. Esto se debe a que en muchos casos los comentarios de los medios se cargan usando herramientas de terceros que dependen de iframes y que, al usarun tipo concreto de request para coger el código HTML, no se podía alcanzar desde el código Python.

Por tanto, para atajar este problema, la alternativa más exitosa fue usar Selenium para poder acceder a esos iframe. No obstante, la url con el HTML de los comentarios tenía que cargarse en una pestaña aparte, además de que Selenium abre físicamente el navegador al cliente que esté ejecutando el código, lo cual a su vez generaba otros dos problemas:

- Según el medio o la herramienta que use el mismo para cargar los comentarios en las noticias, el hecho de que haya que hacer nuevas peticiones con las urls de los comentarios para cada noticia es una operación muy pesada que conllevaba varios minutos de carga en muchos casos, algo totalmente no deseable si la aplicación la va a utilizar algún usuario.

- Además del tiempo que se tarda en realizar estas operaciones, el usuario además tendría que aguantar que se estuvieran constantemente abriendo pestañas del navegador mientras se realiza la lectura de comentarios, lo cual tampoco es deseable si es un producto que va a usar un tercero.

Falta de comentarios en RSS

Uno de los principales inconvenientes producidos por el cambio de enfoque de Web Scrapping al RSS de los diarios. Debido a las características de este formato, no se incluyen comentarios en las fichas de las noticias en los RSS de los medios. Debido a ello, a pesar de las ventajas que supone este formato en otros aspectos, como facilidad de lectura de noticias y velocidad de la propia lectura, una de las prestaciones que había que sacrificar era la lectura de comentarios.

Incluso en el caso de aquellos medios en los cuáles hay que acceder al link que ofrece el RSS para leer el cuerpo de la noticia, si también quisiéramos aprovechar para leer comentarios, se volvería al problema mencionado previamente de falta de opciones sencillas de recogida de los mismos.

Por estos dos motivos, se decidió prescindir de la lectura de comentarios en el producto entregable, a falta de encontrar una alternativa menos intrusiva y con menos costes de tiempo y recursos.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.