



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática



TFG del Grado en Ingeniería Informática

**Herramienta para el análisis y
gestión de Hemeroteca**



Presentado por Miguel Rodríguez Rico
en Universidad de Burgos — 15 de junio de 2017
Tutores: Dr. José Francisco Díez Pastor, Dr. César I.
García Osorio



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. Miguel Rodríguez Rico, 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, 15 de junio de 2017

Vº. Bº. del Tutor:

Vº. Bº. del Tutor:

Dr. José Francisco Díez Pastor

Dr. César Ignacio García Osorno

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 general	III
Índice de figuras	V
Índice de tablas	VI
Introducción	1
Objetivos del proyecto	2
2.1. Objetivos basados en los requisitos funcionales	2
2.2. Objetivos técnicos	2
2.3. Objetivos personales	3
Conceptos teóricos	4
3.1. Minería de Datos	4
3.2. Web Scraping	8
Técnicas y herramientas	10
4.1. Técnicas de trabajo	10
4.2. Herramientas Candidatas (Aplicación Web)	10
4.3. Herramientas candidatas (Bases de Datos)	12
4.4. BeautifulSoup	13
4.5. Sci-Kit Learn	13
4.6. feedparser	14
4.7. TextBlob	14
4.8. Lime	14
4.9. PDFMiner	15
4.10. Latex	15
Aspectos relevantes del desarrollo del proyecto	16

ÍNDICE GENERAL

IV

5.1. Conceptos avanzados de Bag of Words	16
5.2. Resultados de la explicación de la predicción	20
5.3. Experimentación en Jupyter	21
5.4. Intento de publicación de Jupyter Dashboard	22
5.5. Omisión de comentarios en el Scrapping	23
Trabajos relacionados	25
Conclusiones y Líneas de trabajo futuras	26
Bibliografía	27

Índice de figuras

3.1. Esquema de funcionamiento básico de Bagging [27]	7
3.2. Esquema de funcionamiento de bosque aleatorio [7]	8
3.3. Petición http	8
3.4. Lectura del contenido de la web	9
3.5. Acceso a la información deseada con el scraper	9

Índice de tablas

3.1. Frecuencias de palabras en las frases de ejemplo	5
5.2. Stopwords español de nltk	18
5.3. Stopwords español de nltk (Segunda Parte)	19

Introducción

Descripción del contenido del trabajo y del estructura 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 [11] [31] 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 fumadores” “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.

En el apartado “Conceptos Avanzados de Bag of Words” dentro de “Aspectos Relevantes” se explicarán con más detenimiento algunos aspectos más profundos de esta técnica.

Vectorizador de texto

Herramientas que convierten un documento de texto en matrices de tokens, que usualmente son palabras [18]. 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 Aggregation [2], 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.

Esta 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.

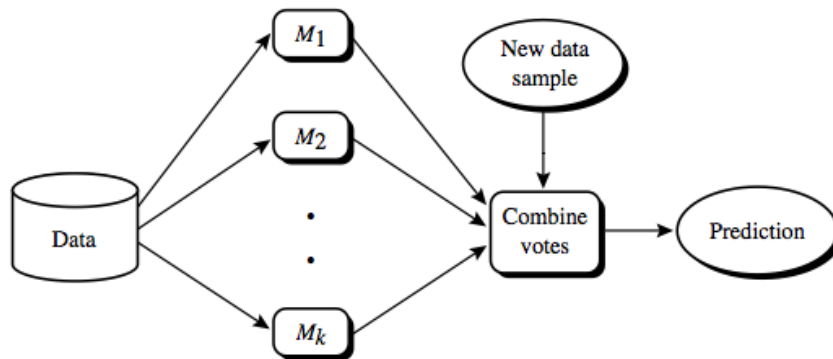


Figura 3.1: Esquema de funcionamiento básico de Bagging [27]

Clasificador Random Forest

El clasificador Random Forest es una técnica derivada del bagging aplicada a árboles de predicción [19]. 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 [3].

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

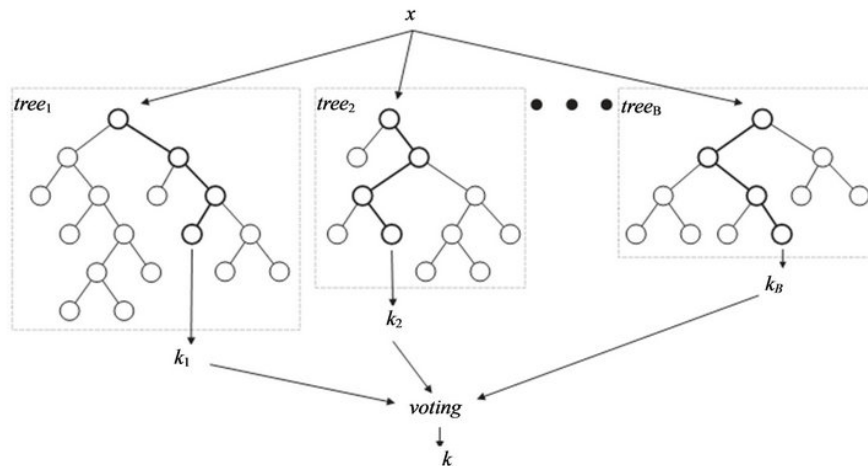


Figura 3.2: Esquema de funcionamiento de bosque aleatorio [7]

3.2. Web Scraping

El Web Scraping [17] 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/sevenzzip/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

Técnicas y herramientas

Este capítulo está dedicado a la mención de una serie de herramientas y técnicas que, o han sido esenciales en el desarrollo de la aplicación, o su uso fue propuesto para realizar partes de la programación en ellas.

4.1. Técnicas de trabajo

Metodología ágil

El desarrollo de este proyecto ha sido llevado a cabo en un contexto de metodología ágil, que se basa en seguir los principios del manifiesto ágil [8], como dar prioridad a la comunicación con el cliente y entre los miembros del equipo, permitir que los requisitos sean flexibles a lo largo del desarrollo, hacer progresos de forma constante en periodos que comprendan una o pocas semanas... etc.

En el apartado "Planificación del proyecto" de los anexos, se profundizará más sobre este concepto, explicando en detalle cómo se organizaba el desarrollo y el progreso realizado a lo largo de las semanas.

4.2. 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

PyQt es una herramienta [32] que 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

Kivy [12] es una alternativa interesante ya que permite hacer aplicaciones de escritorio con código Python con la ventaja de que es “Cross-platform”. En otras palabras, 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

Flask es una herramienta [9] que permite combinar código HTML y Python de forma simple. La característica 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.

A pesar de que inicialmente Jupyter parecía una alternativa más cómoda, los objetivos respecto del producto final que se quería conseguir provocaron que la aplicación fuera desarrollada en Flask, ya que permitía crear una aplicación web convencional basada en templates de forma mucho más fácil.

En el aprendizaje de esta herramienta, cabe destacar lo útil que resultó el Mega-Tutorial de Miguel Grinberg [10] a la hora de familiarizarse con los conceptos básicos.

Jupyter

La herramienta Jupyter [22] permite con ayuda de alguna librería de terceros (wand, display...) combinar código HTML y Python en un mismo fichero de manera especialmente sencilla. De hecho, haciendo uso de los llamados “ipywidgets”(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. La posibilidad de crear y ejecutar funcionalidades diferentes y mostrarlas en una interfaz de forma muy rápida hacía muy atractiva esta herramienta.

De hecho, fue elegida inicialmente para implementar la aplicación web. No obstante, debido a los problemas sufridos de configuración de los notebook como una aplicación web y su posible subida a servidor, esta plataforma tuvo que ser descartada para el producto final, a pesar de que parte importante de la funcionalidad fuera inicialmente desarrollada usando esta herramienta. En el repositorio del proyecto en GitHub se encuentran todos los experimentos de funcionalidad probados en notebooks.

4.3. Herramientas candidatas (Bases de Datos)

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

PyLucene [1] es una extensión para Python de Apache Lucene [30], motor de búsqueda de texto, usada frecuentemente en los buscadores de texto de Internet.

Debido a la adecuación de una base de datos NoSQL como MongoDB para nuestro proyecto, PyLucene no se adecuaba con exactitud a la funcionalidad para la que queríamos una base de datos para almacenar la información de las noticias.

PyMongo

Es la extensión de MongoDB [4] 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 (no de forma convencional, se pueden incluir referencias a documentos como campos de otros), estas

estructuras son muy flexibles, y se puede, por ejemplo, introducir unas estructuras como campos de otras, introducir cualquier tipo de dato como valor, o tener documentos con diferentes estructuras y campos dentro de una misma colección.

Por la flexibilidad mencionada, y por su fácil uso en Python a través de las librerías, ésta fue la alternativa elegida para almacenar los datos.

SQLAlchemy

SQLAlchemy [5] es una base de datos relacional usada frecuentemente en Python. En nuestro caso, nos encontramos con ella al realizar el Mega-Tutorial de Flask, y por su sencilla implementación y uso, se decidió investigar como candidata para la aplicación final.

No obstante, es una base de datos SQL, y debido a la flexibilidad que se quería conseguir en el almacenamiento de las noticias, no se adaptaba a nuestras necesidades tanto como las alternativas NoSQL.

4.4. BeautifulSoup

BeautifulSoup [24] [6] 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.

4.5. Sci-Kit Learn

Sci-kit Learn [21] [20] es una librería de Python enfocada en el machine learning y la minería de datos. Ofrece múltiples funcionalidades que se

pueden aplicar en diferentes etapas de la propia minería como preprocesado, clustering, reducción de dimensionalidad, selección de modelo, regresión y clasificación.

En nuestro proyecto, sólo se usaron herramientas de preprocesado, para transformar el texto de las noticias que se recibe en algo utilizable por la librería, y de clasificación, para poder realizar la predicción de etiquetas.

4.6. feedparser

Librería para Python cuya funcionalidad es similar a la de un Web Scraper. A diferencia de BeautifulSoup, feedparser [13] está diseñada para parsear los RSS de las webs, y no el template HTML convencional.

Su forma de uso si que es similar a la de BeautifulSoup, pero en lugar de llamar a los elementos por medio de métodos, cuando leemos el código HTML del RSS se almacena en una variable con formato de diccionario, siendo las distintas partes de las entradas claves en este diccionario.

4.7. TextBlob

TextBlob [15] es una 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.

4.8. Lime

Lime [23] es una librería para Python pensada para usarse en combinación con otras librerías de minería de datos como por ejemplo sci-kit learn. Su propósito es el de realizar explicaciones de predicciones realizadas por clasificadores externos donde sólo se sabe el resultado, pero no cómo se ha llegado a él.

Además de prestar este servicio, la librería también es muy flexible a la hora de exportar los resultados de estas explicaciones, permitiendo devolver los resultados con formato de lista, de gráfico de barras... aunque el más interesante es el de bloque de código HTML, compuesto de varios bloques distintos incluyendo una muestra del texto clasificado con las palabras relevantes resaltadas.

Esta última opción es muy interesante en desarrollo web, donde se puede inyectar este bloque de código HTML en nuestra página de forma muy sencilla, sin necesidad de tener que crear manualmente el contenido necesario para mostrar la explicación.

Lime se puede usar tanto en clasificación de imágenes como en clasificación de textos, pero en el caso de nuestro proyecto únicamente nos interesaban las explicaciones para predicciones de texto. En el caso de las imágenes, también hay un formato de resultado devuelto donde se indica qué fragmentos de la imagen se han tenido en cuenta para elegir una clase u otra.

4.9. PDFMiner

Librería, también para Python, utilizada para la manipulación, a través de código, de archivos PDF [25] [26]. Además de para la lectura del texto que contenga, que sería el uso más convencional, también se pueden obtener otros datos como el nombre del usuario que creó el archivo y la fecha en la que fue creada, además de otros metadatos. No sólo eso, si no que además incluye un convertidor de formato, que da la posibilidad de transformar los ficheros PDF en otros formatos como HTML (Cita al repositorio).

En lo respectivo a nuestro proyecto, sólo nos interesaba la funcionalidad de lectura de texto del archivo, puesto que como lo que se almacena en base de datos es información relacionada con la noticia, los metadatos como creador del PDF o fecha de creación seguramente no coincidan con el autor de la noticia o la fecha de publicación de la misma.

4.10. Latex

Por último, es necesario mencionar la herramienta en la que está escrita esta memoria. Latex [14] es un editor de texto caracterizado por realizar la escritura los documentos en texto plano y aplicar los formatos en una compilación posterior. Es frecuentemente usado en documentación de índole científica [29](Artículos, libros, etc...).

Aspectos relevantes del desarrollo del proyecto

En este capítulo se desarrollarán en profundidad aspectos sólo mencionados en capítulos previos, relacionados tanto con conceptos teóricos como con las técnicas y herramientas, además de dar explicación a cuestiones relevantes respecto de la aplicación o de su desarrollo.

5.1. Conceptos avanzados de Bag of Words

Además de lo mencionado en los conceptos teóricos, hay diferentes matices dentro del bag of words que merecen la pena ser explicados con más detalle, ya que fueron relevantes en la implementación del proyecto.

Stopwords

Se conoce como “stopwords” al conjunto de palabras prefijadas por el programador (o por el usuario, si se le da la opción) que no queremos que se tengan en cuenta a la hora de formar un vocabulario con el texto de entrada.

Cuando se está dando forma al modelo, se le pasa este conjunto a la herramienta que va a transformar el texto en un vocabulario con frecuencia de palabras, de modo que ésta filtra las palabras mencionadas para eliminar el “ruido” de forma consecuente, aumentar la relevancia del resto de palabras que almacene.

Habitualmente, lo que se busca evitar son las palabras que tienen menos significado propio dentro de una frase. Dicho de otra manera, las primeras palabras que deben ser filtradas son artículos, determinantes, preposiciones... que son las menos interesantes en una frase.

En este proyecto, se hizo uso de un stopwords de palabras en español que viene incluido en la librería de NLTK [28], que contiene las palabras de las tablas [?] y [?].

Ya que estas palabras son muy frecuentes en la mayoría de frases en español, no sólo restaban importancia a las verdaderamente relevantes, si no que incluso podrían ser consideradas de las más relevantes simplemente por su frecuencia de aparición, lo cuál restaría mucha calidad a la predicción posterior.

Cabe destacar que ésto no soluciona todo el problema, ya que en los textos de las noticias podrían aparecer alguna de estas palabras con mayúsculas, sin tildes, u otras palabras no relevantes que evitarían el filtro. No obstante, con este filtro ya se puede asegurar una cierta calidad en las predicciones de forma general.

Normalización de las frecuencias

Como se ha mencionado previamente, lo habitual es que una predicción de clases basada en un bag of words se realice basando la clasificación en la frecuencia de aparición de determinadas palabras en los textos de cada una de las clases.

Para poder medir esta frecuencia, no es suficiente con tener los vocabularios de cada clase con sus frecuencias, pues no es una medida lo suficientemente precisa. Lo que se debe realizar es una normalización de estas frecuencias, teniendo en cuenta los siguientes cálculos [16]

- Frecuencia normalizada de una palabra en una clase:

$$FrecuenciaClase1[palabra] = apariciones[palabra]/numeroNoticias[Clase1]$$

Se recomienda añadir una aparición por defecto, para evitar divisiones entre 0.

- Ratio de pertenencia de una palabra a una clase:

$$RatioClase1[palabra] = FrecuenciaClase1[palabra]/FrecuenciaClase2[palabra]$$

Si, siguiendo el ejemplo que mencionamos en los conceptos teóricos, tenemos que la palabra “salud” tiene un mayor ratio en la clase “En contra de los fumadores” que en “A favor de los fumadores”, se puede anticipar que la próxima vez que se reciba una frase o texto con la palabra “salud”, tendrá mayores posibilidades de pertenecer a la clase “En contra de los fumadores”.

Palabras			
de	la	que	el
en	y	a	los
del	se	las	por
un	para	con	no
una	su	al	lo
como	más	pero	sus
le	ya	o	este
porque	esta	entre	sí
cuando	muy	sin	sobre
también	me	hasta	hay
donde	quien	desde	todo
nos	durante	todos	uno
les	ni	contra	otros
ese	eso	ante	ellos
e	esto	mí	antes
algunos	qué	unos	yo
otro	otras	otra	él
tanto	esa	estos	mucho
quienes	nada	muchos	cual
poco	ella	estar	estas
algunas	algo	nosotros	mi
mis	tú	te	ti
tu	tus	ellas	nosotras
vosotros	vosotras	os	mío
mía	míos	mías	tuyo
tuya	tuyos	tuyas	suyo
suya	suyos	suyas	nuestro
nuestra	nuestros	nuestras	vuestra
vuestros	vuestras	esos	esas
estoy	estás	está	estamos
estáis	están	esté	estés
estemos	estéis	estén	estaré
estarás	estará	estaremos	estaréis
estarán	estaría	estarías	estaríamos
estaríais	estarían	estaba	estabas
estábamos	estabais	estaban	estuve
estuviste	estuvo	estuvimos	estuvisteis
estuvieron	estuviera	estuvieras	estuviéramos
estuvierais	estuvieran	estuviese	estuvieses
estuviésemos	estuvieseis	estuviesen	estando
estado	estada	estados	estadas
estad	he	has	ha
hemos	habéis	han	haya
hayas	hayamos	hayáis	hayan
habré	habrás	habrá	habremos
habréis	habrán	habría	habrías
habríamos	habríais	habrían	había
habías	habíamos	habíais	habían
hube	hubiste	hubo	hubimos

Tabla 5.2: Stopwords español de nltk

Palabras			
hubisteis	hubieron	hubiera	hubieras
hubiéramos	hubierais	hubieran	hubiese
hubieses	hubiésemos	hubieseis	hubiesen
habiendo	habido	habida	habidos
habidas	soy	eres	es
somos	sois	son	sea
seas	seamos	seáis	sean
seré	serás	será	seremos
seréis	serán	sería	serías
seríamos	seríais	serían	era
eras	éramos	erais	eran
fui	fuiste	fue	fuimos
fuisteis	fueron	fuera	fueras
fuéramos	fuerais	fueran	fuese
fueses	fuésemos	fueseis	fuesen
sintiendo	sentido	sentida	sentidos
sentidas	siente	sentid	tengo
tienes	tiene	tenemos	tenéis
tienen	tenga	tengas	tengamos
tengáis	tengan	tendré	tendrás
tendrá	tendremos	tendréis	tendrán
tendría	tendrías	tendríamos	tendríais
tendrían	tenía	tenías	teníamos
teníais	tenían	tuve	tuviste
tuvo	tuvimos	tuvisteis	tuvieron
tuviera	tuvieras	tuviéramos	tuvierais
tuvieran	tuviese	tuvieses	tuviésemos
tuvieseis	tuviesen	teniendo	tenido
tenida	tenidos	tenidas	

Tabla 5.3: Stopwords español de nltk (Segunda Parte)

Realizando estos simples cálculos y ordenando las palabras en función de sus respectivos ratio, se puede deducir cuáles serán las palabras más influyentes a la hora de saber si una frase o un texto se corresponde con una clase u otra.

5.2. Resultados de la explicación de la predicción

Una de las funcionalidades que más peso tiene en el producto final es el de mostrar una explicación de la predicción realizada sobre una noticia, respecto de un Dataset determinado.

Siguiendo la filosofía de “no reinventar la rueda”, para parte de la programación de este apartado hemos usado la librería Lime, de la cual se habla en el apartado de “Técnicas y Herramientas”, y que permite obtener una explicación de cualquier clasificación realizada, ya sea usando texto o imagen, y preferiblemente usando Scikit-learn como librería para practicar la minería de datos.

Se ha aprovechado la posibilidad que ofrece Lime de devolver esta explicación con un formato de bloque HTML, en el cuál a su vez se incluyen tres apartados distintos:

- Barras de progreso que indican las probabilidades de pertenencia a cada una de las clases.
- Un gráfico de barras en el que se muestran por orden las palabras más influyentes a la hora de tomar la decisión, ya sea para una clase o para otra.
- El texto de prueba entero que se ha recibido, resaltando en él las palabras que se han incluido en el gráfico del punto anterior, para que el usuario pueda observar estas palabras en su contexto real (antes de hacer el bag of words), y no simplemente en la lista de palabras influyentes.

En la imagen (Insertar captura de pantalla de explicación) se observan de forma diferenciada los tres apartados, y se le ofrece al usuario la posibilidad de, después de haber estudiado con detenimiento los detalles de la predicción, decidir etiquetar esta noticia en concordancia con la predicción realizada.

Puede que al usar la aplicación de la sensación de que el estilo visual de estos resultados de la explicación no encaja completamente con el estilo visual del resto de las pestañas. Esto es debido a que se inyecta directamente el bloque HTML sin alterar que devuelve Lime en la pestaña antes de cargar la misma. Debido a limitaciones de tiempo, era más productivo aprovechar este código HTML ya escrito que solamente recibir de Lime los datos y dedicar tiempo extra a escribir el código HTML y las funciones de Javascript necesarias para

preparar la misma vista de resultados pero con otros estilos diferentes. De nuevo, se ha decidido seguir el razonamiento de que “no hay que reinventar la rueda”.

5.3. 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.
- Clasificación con Bag of Words: En algunos de los notebooks hay muestras de las primeras pruebas de clasificación de texto que se hicieron usando Scikit-Learn, en las que además se puede ver como se usan otros elementos como la validación cruzada para crear conjuntos de entrenamiento y de prueba a partir de los mismos elementos iniciales. Como en la aplicación final sabemos a ciencia cierta cuál son las noticias de entrenamiento y cuál sobre la que hay que hacer la predicción, no hace falta usar esta validación cruzada.
- Lectura más exhaustiva de PDF: También hay un apartado de notebooks dedicado a la lectura de PDF y extracción de su texto, donde se implementaron los primeros acercamientos antes de programar la versión final en la aplicación de Flask. No obstante, esa versión tenía el problema de que los PDF de prueba que se habían recibido tenían diferentes formatos y no se había acordado una estructura común para el contenido, provocando que el texto que se recogía no fuera útil.

5.4. 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 “output” 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.

5.5. 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 usar un 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.

Bibliografía

- [1] Apache. Apache lucene - welcome to pylucene, 2017. [Online; accessed 13-June-2017].
- [2] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
- [3] Leo Breiman. Random forests, 2017. [Online; accessed 13-June-2017].
- [4] Kristina Chodorow and Michael Dirolf. *MongoDB: The Definitive Guide*. O’Reilly Media, Inc., 1st edition, 2010.
- [5] Rick Copeland. *Essential sqlalchemy*. .O’Reilly Media, Inc.”, 2008.
- [6] Crummy. Beautiful soup: We called him tortoise because he taught us., 2017. [Online; accessed 13-June-2017].
- [7] Ha Nam Nguyen Cuong Nguyen, Yong Wang1. Random forest classifier combined with feature selection for breast cancer diagnosis and prognostic, 2017. [Online; accessed 14-June-2017].
- [8] Martin Fowler and Jim Highsmith. The agile manifesto. *Software Development*, 9(8):28–35, 2001.
- [9] Miguel Grinberg. *Flask Web Development: Developing Web Applications with Python*. O’Reilly Media, Inc., 1st edition, 2014.
- [10] Miguel Grinberg. The flask mega tutorial, 2017. [Online; accessed 15-June-2017].
- [11] Ralph Grishman. Bag-of-words methods for text mining, 2017. [Online; accessed 13-June-2017].
- [12] Kivy. Kivy: Cross-platform python framework for nui development, 2017. [Online; accessed 13-June-2017].

- [13] Mark Pilgrim Kurt McKee. feedparser 5.2.1 : Python package index, 2017. [Online; accessed 13-June-2017].
- [14] Leslie Lamport. *LATEX: a document preparation system: user's guide and reference manual*. Addison-wesley, 1994.
- [15] Steven Loria. Textblob: simplified text processing. *Secondary TextBlob: Simplified Text Processing*, 2014.
- [16] Kevin Markham. pycon-2016-tutorial, 2017. [Online; accessed 15-June-2017].
- [17] Ryan Mitchell. *Web Scraping with Python: Collecting Data from the Modern Web*. O'Reilly Media, Inc.", 2015.
- [18] Andreas Müller et al. Countvectorizer documentation, 2017. [Online; accessed 13-June-2017].
- [19] Andreas Müller et al. Randomforest classifier documentation, 2017. [Online; accessed 13-June-2017].
- [20] Andreas Müller et al. scikit-learn: machine learning in python — scikit-learn 0.18.1 documentation, 2017. [Online; accessed 13-June-2017].
- [21] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.
- [22] Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.
- [23] Marco Tulio Correia Ribeiro. Lime: Explaining the predictions of any machine learning classifier, 2017. [Online; accessed 13-June-2017].
- [24] Leonard Richardson. Beautiful soup documentation, 2007.
- [25] Yusuke Shinyama. Pdfminer: Python pdf parser and analyzer. *Retrieved on*, 11, 2015.
- [26] Yusuke Shinyama. Python pdf parser, 2017. [Online; accessed 13-June-2017].
- [27] Google Sites. Classification and prediction - wiki, 2017. [Online; accessed 14-June-2017].
- [28] Mikhail Korobov Steven Bird et al. Nltk, 2017. [Online; accessed 15-June-2017].

- [29] Wikipedia. Latex — wikipedia, la enciclopedia libre, 2015. [Internet; descargado 30-septiembre-2015].
- [30] Wikipedia. Apache lucene — wikipedia, the free encyclopedia, 2017. [Online; accessed 13-June-2017].
- [31] Wikipedia. Bag-of-words model — wikipedia, the free encyclopedia, 2017. [Online; accessed 13-June-2017].
- [32] Wikipedia. Pyqt — wikipedia, the free encyclopedia, 2017. [Online; accessed 13-June-2017].