



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
Documentación Técnica**



Presentado por Miguel Rodríguez Rico
en Universidad de Burgos — 27 de junio de 2017
Tutor: Dr. José Francisco Díez Pastor, Dr. César I.
García Osorio

Índice general

Índice general	I
Índice de figuras	III
Índice de tablas	V
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	24
Apéndice B Especificación de Requisitos	27
B.1. Introducción	27
B.2. Objetivos generales	27
B.3. Catalogo de requisitos	28
B.4. Especificación de requisitos	29
Apéndice C Especificación de diseño	37
C.1. Introducción	37
C.2. Diseño de datos	37
C.3. Diseño procedimental	38
C.4. Diseño arquitectónico	40
Apéndice D Documentación técnica de programación	42
D.1. Introducción	42
D.2. Estructura de directorios	42
D.3. Manual del programador	43
D.4. Compilación, instalación y ejecución del proyecto	45
D.5. Pruebas del sistema	47

Apéndice E Documentación de usuario	52
E.1. Introducción	52
E.2. Requisitos de usuarios	52
E.3. Instalación	53
E.4. Manual del usuario	54
Bibliografía	63

Índice de figuras

A.1. Issues del primer sprint	2
A.2. Progreso de resolución de tareas del primer sprint	3
A.3. Issues del segundo sprint	3
A.4. Progreso de resolución de tareas del segundo sprint	4
A.5. Issues del tercer sprint	5
A.6. Progreso de resolución de tareas del tercer sprint	5
A.7. Issues del cuarto sprint	6
A.8. Progreso de resolución de tareas del cuarto sprint	7
A.9. Issues del quinto sprint	7
A.10. Progreso de resolución de tareas del quinto sprint	8
A.11. Issues del sexto sprint	8
A.12. Progreso de resolución de tareas del sexto sprint	9
A.13. Issues del séptimo sprint	10
A.14. Progreso de resolución de tareas del séptimo sprint	10
A.15. Issues del octavo sprint	11
A.16. Progreso de resolución de tareas del octavo sprint	11
A.17. Issues del noveno sprint	12
A.18. Progreso de resolución de tareas del noveno sprint	13
A.19. Issues del décimo sprint	13
A.20. Progreso de resolución de tareas del décimo sprint	14
A.21. Issues del undécimo sprint	14
A.22. Progreso de resolución de tareas del undécimo sprint	15
A.23. Issues del duodécimo sprint	15
A.24. Progreso de resolución de tareas del duodécimo sprint	16
A.25. Issues del decimotercer sprint	17
A.26. Progreso de resolución de tareas del decimotercer sprint	17
A.27. Issues del decimocuarto sprint	18
A.28. Progreso de resolución de tareas del decimocuarto sprint	18
A.29. Issues del decimoquinto sprint	19

A.30. Progreso de resolución de tareas del decimoquinto sprint	19
A.31. Issues del decimosexto sprint	20
A.32. Progreso de resolución de tareas del decimosexto sprint	21
A.33. Issues del decimoseptimo sprint	21
A.34. Progreso de resolución de tareas del decimoseptimo sprint	22
A.35. Issues del decimooctavo sprint	23
A.36. Progreso de resolución de tareas del decimooctavo sprint	23
A.37. Issues del decimonoveno sprint	24
A.38. Progreso de resolución de tareas del decimonoveno sprint	24
 B.1. Diagrama de casos de uso	 29
 C.1. Diagrama de flujo para un caso de interacción habitual	 39
C.2. Diagrama de paquetes	40
C.3. Clases utilizadas en la herramienta	41
 D.1. Ubicación del código utilizado en el aprendizaje automático	 44
D.2. Funciones encargadas de la interacción con las pestañas	45
D.3. Nota obtenida antes de realizar refactorizaciones	48
D.4. Nota obtenida después de varias semanas, aplicando algunas técnicas de refactorización	49
D.5. Nota obtenida en los últimos días de desarrollo	49
D.6. Algunos de los factores que valora Pylint	50
D.7. Nota obtenida antes de aplicar cambios	51
D.8. Nota después de los primeros cambios	51
D.9. Nota después de haber aplicado la mayoría de los cambios	51
 E.1. Ejemplo de palabra clave de búsqueda en la herramienta	 54
E.2. Tabla de resultados	55
E.3. Explicación para la predicción de la clase de una noticia	55
E.4. Pestaña de lectura de PDF	56
E.5. Resultados de la lectura de PDF	57
E.6. Imagen inicial de la pestaña de lectura de RSS	58
E.7. Resultados de lectura de RSS	58
E.8. Pestaña de lectura de Corpus	59
E.9. Estructura de los corpus en XML	59
E.10. Tabla de resultados con el corpus	60
E.11. Pestaña de creación de Datasets	61
E.12. Gráfico de noticias por autor	62

Índice de tablas

A.1. Tabla de bibliotecas y sus licencias	26
B.1. Caso de uso 1	30
B.2. Caso de uso 2	31
B.3. Caso de uso 3	32
B.4. Caso de uso 4	33
B.5. Caso de uso 5	34
B.6. Caso de uso 6	35
B.7. Caso de uso 7	36

Apéndice A

Plan de Proyecto Software

A.1. Introducción

A continuación se va a describir en profundidad cuál ha sido la planificación del proyecto, semana a semana, organizado en base a sprints.

Para tener una visión apropiada de todo el proceso, se ha usado la extensión ZenHub que se puede añadir a GitHub y que permite observar con más detalle todo lo relacionado con la metodología ágil. El repositorio de este proyecto es <https://github.com/mrr0088/Gestor-de-Hemeroteca>, aunque ya se incluyó al usuario “ubutfgm” para que las observaciones necesarias sean realizadas.

Cabe destacar que, especialmente en los primeros sprints, por falta de experiencia con la herramienta ZenHub, los gráficos de progreso no son representativos del progreso real que se llevó a cabo, debido sobre todo a que los issues que los componen no estaban cerrados correctamente.

A.2. Planificación temporal

Como ya se ha mencionado, el desarrollo se dividió en sprints, equivalentes a una semana, excepto en un sprint de dos semanas. Cada comienzo y final de sprint coincide con la reunión semanal del alumno con los tutores.

En nuestro caso, hemos considerado que los story points asignados a cada tarea simbolicen las horas que se tarda en resolver ese issue. Por ejemplo, si una tarea tiene 5 story points se estima que el alumno la realizará en 5 horas.

Primer Sprint (2 de Febrero - 10 de Febrero):

Este tramo inicial fue dedicado a realizar investigación sobre las herramientas que se usarían para desarrollar el producto final. Se buscó información

sobre 3 elementos o partes distintas:

- Herramienta para desarrollar la GUI: Concretamente, se debatió acerca de si sería más adecuado crear una aplicación web o una aplicación interna. Para tomar una decisión, se tuvieron en cuenta 4 herramientas. Por la parte de aplicación web, las candidatas fueron Jupyter y Flask; y por la parte de aplicación de escritorio, PyQt y Kivy. Tras practicar con ambas herramientas, se tomó la decisión de programar la GUI en un notebook de Jupyter para posteriormente publicarlo en un Dashboard.
- Base de datos para almacenar la información: Como se debía almacenar numerosa y variada información sobre las noticias elegidas, se necesitaba de una estructura donde almacenar todos esos datos. La decisión más relevante en este apartado era si usar una base de datos relacional o no relacional, o dicho de otra manera, SQL o NoSQL. Las alternativas fueron SQLAlchemy (relacional), PyLucene (no relacional) y PyMongo (no relacional).
- Herramienta para escribir la documentación: Aquí sólo se plantearon dos alternativas inicialmente: OpenOffice o Latex. Por cuestiones de formato y de preferencia de los miembros del proyecto, se optó por Latex, aprovechando la plantilla para Trabajos de Fin de Grado que ya había sido creada y que estaba a disposición de los alumnos.

Completed Issues and Pull Requests					Story points
Reunión 1	Python_DataClassification #1 III New Issues ↑ Primer Sprint				1
	Investigar Latex Investigation				5
Investigar Latex	Python_DataClassification #2 III New Issues ↑ Primer Sprint				

Figura A.1: Issues del primer sprint

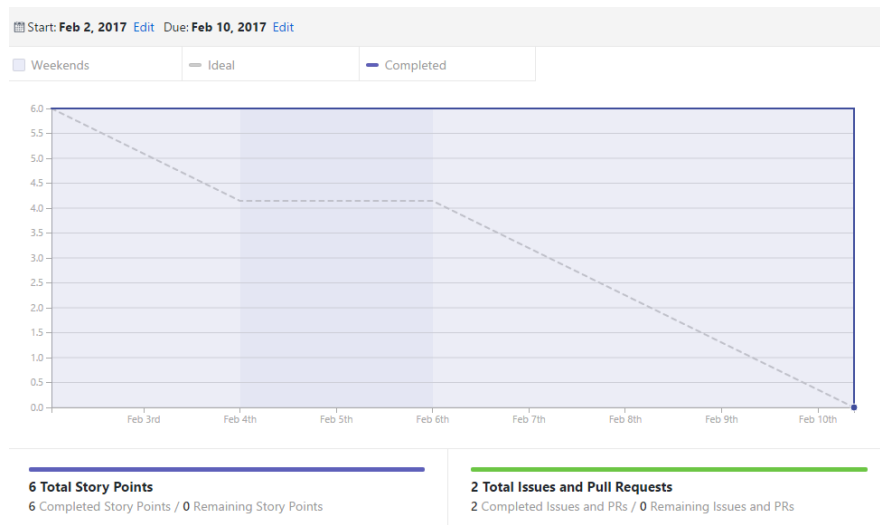


Figura A.2: Progreso de resolución de tareas del primer sprint

Segundo Sprint (10 de Febrero - 16 de Febrero):

La prioridad de este sprint era finalizar la investigación y búsqueda de herramientas para desarrollar la aplicación, que concluyó con la elección de Pymongo como base de datos, y los notebook de Jupyter como herramienta de interfaz. Una vez elegidas las diferentes herramientas necesarias para el desarrollo del proyecto, el siguiente paso consistió en crear un pequeño programa que probara la librería PDFMiner y que contuviera algo de funcionalidad básica como contar la frecuencia de una palabra dentro de un documento. Aprovechando este script, se realizaron pruebas con un snippet llamado fileupload, que permite cargar ficheros en el código usando una GUI clásica en la que puedes navegar por el equipo buscando la ubicación del mencionado fichero. El resultado final era un programa que permite al usuario buscar un fichero en el equipo y, si este archivo tiene una extensión pdf, se permitirá buscar la frecuencia de las palabras que introduzca el propio usuario.

Completed Issues and Pull Requests	Story points
<p>Investigar bases de datos de texto Investigation</p> <p>Python_DataClassification #3 III New Issues ↑ Segundo Sprint</p>	13
<p>Investigar alternativas para interfaz de usuario Investigation</p> <p>Python_DataClassification #4 III New Issues ↑ Segundo Sprint</p>	21
<p>Crear script de subida de pdf code</p> <p>Python_DataClassification #5 III New Issues ↑ Segundo Sprint</p>	Not estimated

Figura A.3: Issues del segundo sprint

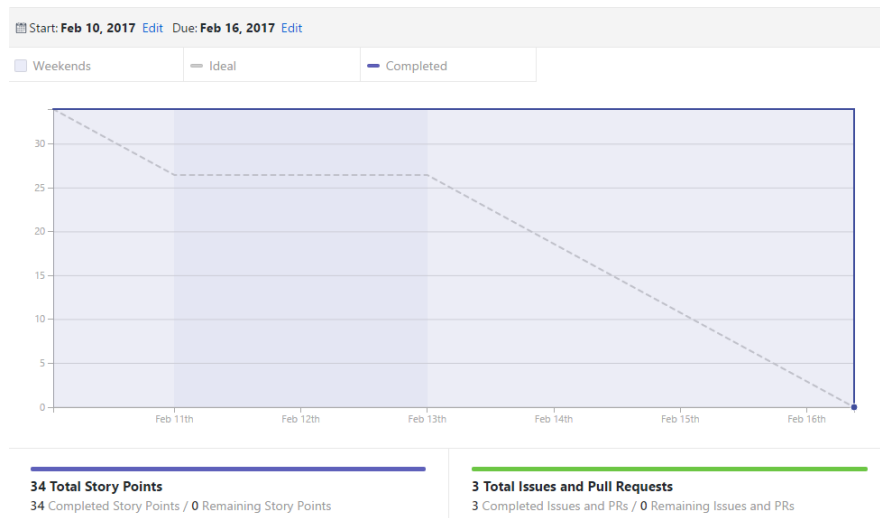


Figura A.4: Progreso de resolución de tareas del segundo sprint

Tercer Sprint (16 de Febrero - 23 de Febrero):

Habiendo finalizado la elección de todas las herramientas necesarias, y probado el funcionamiento básico para lectura de PDF subiendo archivos a mano, la siguiente prioridad fue crear un primer prototipo de web scrapping. Para lograr esto, se dividió la funcionalidad en dos etapas:

- Un primer script que realiza la lectura web de noticias en diferentes medios (para este prototipo, se leían noticias sólo del diario “Público” y, más en concreto, de la sección de Igualdad), almacenando en la base de datos estructuras formadas por el titular, autor, fecha, texto de la noticia, procedencia y un link a la misma. Para recoger el texto html de las páginas web de los diarios, se usaron las librerías de `url2lib` y `BeautifulSoup`; la primera para hacer una llamada a la url indicada y obtener todo (o casi todo, en las próximas etapas explicaremos qué faltaba) el texto HTML de la página de cada noticia, y la segunda para poder manipular todo el HTML recibido previamente de forma más sencilla sin tener que hacer una división manual por etiquetas, simplemente indicamos al contenido de qué etiquetas queremos acceder.
- Un segundo script que, en función de una palabra clave introducida por el propio usuario, consulte en la base de datos y extraiga todas las noticias que contengan esa palabra clave, en un formato de tabla HTML y mostrando la información más importante de la noticia junto con el número de apariciones que hace esa palabra en el texto de las mismas. Para poder mostrar esto en la salida del notebook, necesitamos las librerías de `pandas`, `ipywidgets` y `qgrid`. Con la librería de `pandas`

podemos organizar la información devuelta en la consulta de la base de datos a un dataframe, que puede ser convertido a una tabla html. Con los widgets, podemos mostrar controles básicos como botones, entradas de texto, listas desplegables, etc... sin necesidad de escribir a mano el código HTML. En esta versión, se usó la librería qgrid para transformar el dataframe en una tabla interactiva, que permitía reordenar el orden de las filas dando prioridad a unas columnas u otras. No obstante, no permitían formatear las celdas por separado, lo cual nos hizo descartarlo para posteriores versiones.

Completed Issues and Pull Requests				Story points
Python_DataClassification	#6	III New Issues	Tercer Sprint	13
	Script que introduzca datos de noticias en base de datos <code>code</code>			
Python_DataClassification	#7	III New Issues	Tercer Sprint	13
	Script que muestre enlaces a las noticias contenidas en la base de datos <code>code</code>			

Figura A.5: Issues del tercer sprint

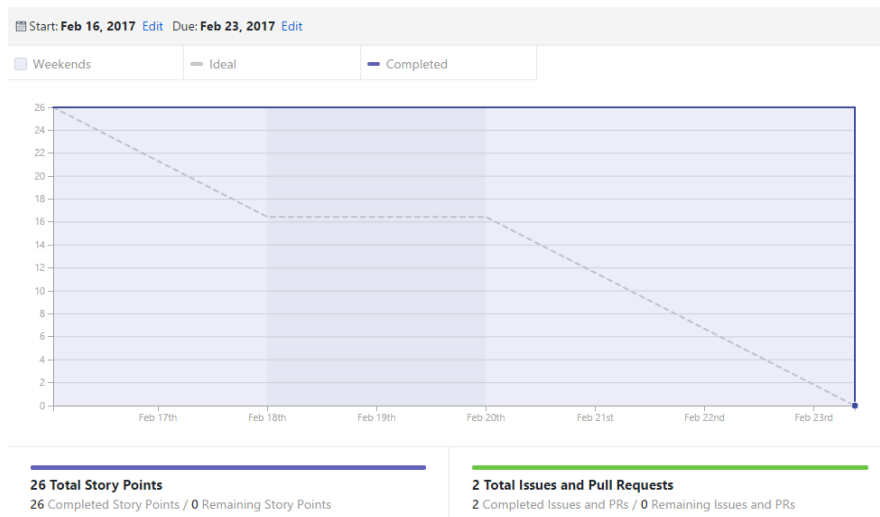


Figura A.6: Progreso de resolución de tareas del tercer sprint

Cuarto Sprint (23 de Febrero - 2 de Marzo):

Habiendo realizado una primera aproximación tanto a la lectura de PDF como de web, este sprint se centró en mejorar tres apartados:

- En la parte de PDF, se mejoró el script, aunque a costa de retirar temporalmente la librería que permitía seleccionar un fichero de forma manual, para que recorriera y almacenara todos los archivos PDF, tanto del directorio que se pasaba como argumento como de todos los que colgaban de él. Una vez recorridos todos, se guardan las noticias en la base de

datos, de forma similar a como se hacía con las noticias en la web pero cambiando el atributo de la procedencia.

- En la parte de web scrapping, se actualizaron dos aspectos importantes:
 - Por una parte, usando Selenium además de BeautifulSoup se consiguió obtener los comentarios de las noticias, que en el caso de “Público”, se generaban con una API de terceros que impedía acceder a su HTML de forma normal como se hacía con el resto de la noticia. El método era algo pesado, ya que selenium necesita abrir en el ordenador que se esté ejecutando las pestañas a las que accede, pero como ya se ha mencionado, fue la alternativa que se encontró más idónea para poder acceder al HTML de los comentarios de las noticias.
 - Por otro lado, se cambió el formato de la tabla, dejando de usar qgrid para el DataFrame y mostrándola en un HTML normal, al que luego se aplicaba estilo manualmente usando un script de Javascript. Ésto es posible ya que uno de los widgets de Ipython/Jupyter permite introducir código HTML de forma manual, y es la vía que también utilizamos para introducir código JavaScript dentro del output del notebook. Con este cambio, tenemos flexibilidad total para manipular las celdas de la tabla para, por ejemplo, formatear como hiperenlaces las url de las noticias que se muestran en la tabla.

Completed Issues and Pull Requests				Story points
🔗	Subida de múltiples pdf	code enhancement		8
	Python_DataClassification #8 III New Issues ↑ Cuarto Sprint			
🔗	Formateo de la tabla de noticias	code enhancement		13
	Python_DataClassification #9 III New Issues ↑ Cuarto Sprint			
🔗	Añadir los comentarios de las noticias (Parte Web)	code enhancement		13
	Python_DataClassification #10 III New Issues ↑ Cuarto Sprint			

Figura A.7: Issues del cuarto sprint

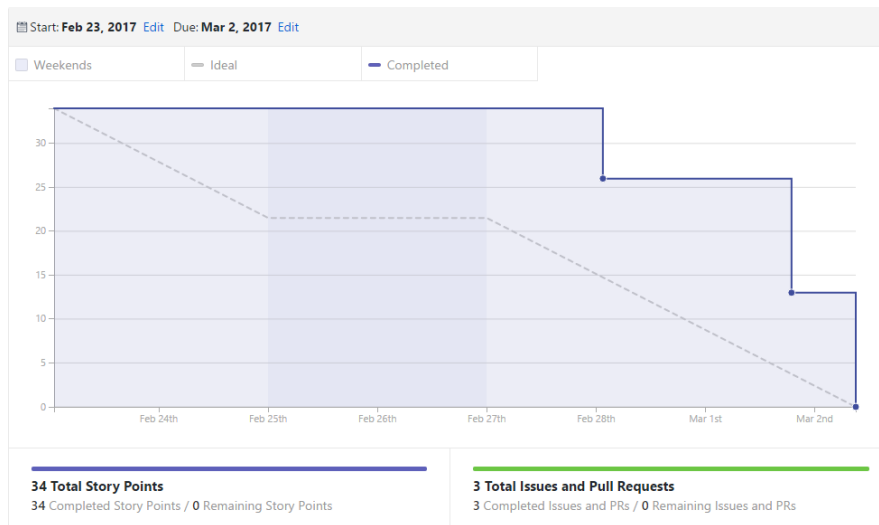


Figura A.8: Progreso de resolución de tareas del cuarto sprint

Quinto Sprint (2 de Marzo - 10 de Marzo):

En esta etapa se decidió que en el muestreo de resultados, sería interesante incluir gráficos que acompañaran al resto de la información, para poder mostrar la mayor cantidad de datos posibles de diversas formas. Por tanto, una de las tareas fue elegir una librería de gráficos adecuada a nuestro contexto. Las candidatas iniciales fueron Bokeh y SeaBorn. Se probó Bokeh y, debido a unos problemas relacionados con la generación de archivos HTML, se decidió descartar. Por falta de horas durante esa semana, se trasladaron las pruebas con SeaBorn al siguiente sprint. La otra gran tarea a la que se dedicó tiempo durante este sprint fue a aumentar los medios a los que se realizaba web scraping, creando un script para recorrer y almacenar las noticias de ElDiario. En este caso, los comentarios no se generaban a partir de una API de terceros, pero debido a la complejidad del código HTML obtenido, y visto necesario el uso de Selenium una vez más, se decidió separar la obtención de comentarios y aplazar esa parte al siguiente sprint.

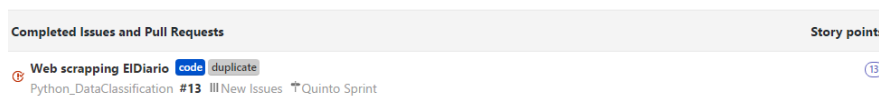


Figura A.9: Issues del quinto sprint

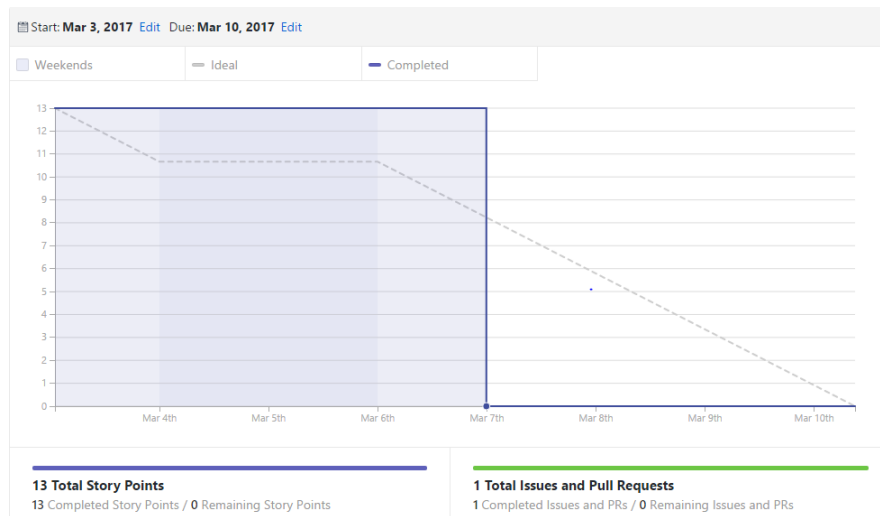


Figura A.10: Progreso de resolución de tareas del quinto sprint

Sexto Sprint (10 de Marzo - 16 de Marzo):

La mayor parte de esta semana se dedicó a terminar tareas de la semana anterior que habían sido subestimadas en horas, y separadas en tareas entre este sprint y el anterior. En el apartado de los gráficos, se realizaron pruebas con la otra librería candidata, Seaborn, y por su comodidad y sencillez para mostrar gráficos simples (Como gráficos de barras para evaluar las menciones a una palabra por autor o día de la semana), se decidió que por el momento sería la alternativa elegida para acompañar en la interfaz. Respecto al web scrapping de ElDiario, se implementó el código necesario junto con el uso de Selenium adecuado para poder obtener los comentarios de las noticias de ese medio, que era la funcionalidad que no dio tiempo a terminar en la anterior semana. Aparte de estas tareas, se dedicó cierta cantidad de tiempo a investigar formas de publicar esta aplicación web en algún servidor. Para comenzar, se realizó cierta investigación sobre Docker, además de ver vídeos de la PyCon de Londres 2016, donde se trataba el tema de subir notebook a un servidor y poder visualizarlos como un dashboard. Por problemas de límite de tiempo, no se pudo avanzar mucho más este aspecto del proyecto.

Completed Issues and Pull Requests	Story points
<p>Ⓢ Añadir gráficos a la interfaz <code>code</code> <code>Investigation</code></p> <p>Python_DataClassification #11 III New Issues ↗ Sexto Sprint</p>	13
<p>Ⓢ Obtener comentarios ElDiario y PDF <code>code</code> <code>enhancement</code></p> <p>Python_DataClassification #14 III New Issues ↗ Sexto Sprint</p>	8
<p>Ⓢ Hacer una prueba de publicación <code>code</code> <code>Investigation</code></p> <p>Python_DataClassification #15 III New Issues ↗ Sexto Sprint</p>	13

Figura A.11: Issues del sexto sprint

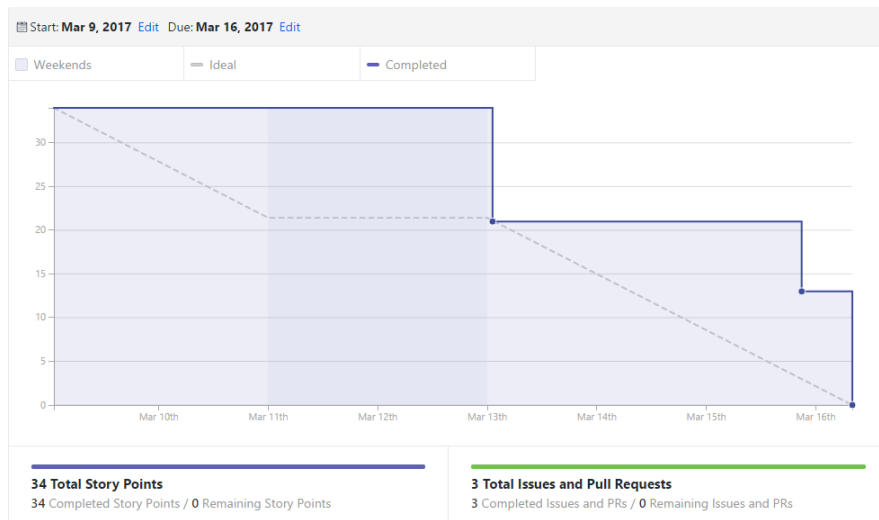


Figura A.12: Progreso de resolución de tareas del sexto sprint

Séptimo Sprint (16 de Marzo - 23 de Marzo):

En este sprint se dedicó parte del tiempo a solucionar un problema recurrente en el que se mostraban sucesivamente todas las tablas de resultados de todas las palabras buscadas, provocando que la interfaz resultante se llenara de tablas innecesarias. De la misma manera, con los gráficos de Seaborn, que usa Matplotlib para funcionar, sucedía algo similar, provocando que todos los gráficos generados se fueran acumulando en una misma zona de dibujo que comparten por requisitos de la librería. Para solucionar esto hicieron falta dos medidas distintas. Para la parte de las tablas html, el problema se solucionó desde el código de Python, cambiando el planteamiento de la función que se ejecutaba al pulsar en el botón de búsqueda, y haciendo globales algunas variables necesarias para la generación de estas tablas. Para la parte de los gráficos, como este problema se estaba produciendo por limitaciones de la librería, hubo que buscar una solución más “rudimentaria” y controlar, por medio de una función JavaScript (introducida en un widget HTML), la visualización de la tabla requerida y la no visualización del resto de tablas obsoletas. El resto del tiempo de esa semana se dedicó a la investigación de la herramienta TextBlob, una opción muy interesante para realizar minería de datos de texto. En términos generales, es una herramienta que, a partir de los textos recibidos, puede traducirlos, detectar idiomas, corregir palabras, dividir párrafos en frases, y frases en palabras, devolver etiquetas de las frases o palabras... etc. Aparte de toda esta funcionalidad, muy útil para casos como el de este proyecto, en el que se manejan textos, tiene sus propios clasificadores de datos. Para usar esta herramienta, se necesita descargar el NLTK (Natural Language ToolKit), que contiene todos los conjuntos de datos necesarios para poder llevar a cabo estas manipulaciones de texto y clasificaciones de

los mismos. Debido a problemas con la descarga y tamaño de los archivos, se cerró la tarea en ese punto, y se retomó en el siguiente sprint. Al término de esta semana, se realizó una reunión con el cliente para enseñar los progresos alcanzados hasta ese punto y tomar decisiones acerca de qué pasos tomar a continuación.

Completed Issues and Pull Requests		Story points
Probar text blob Python_DataClassification #16 III New Issues ↑ Séptimo Sprint		13
Reunión con cliente Python_DataClassification #19 III New Issues ↑ Séptimo Sprint		2
Solucionar display repetido Python_DataClassification #20 III New Issues ↑ Séptimo Sprint		5

Figura A.13: Issues del séptimo sprint

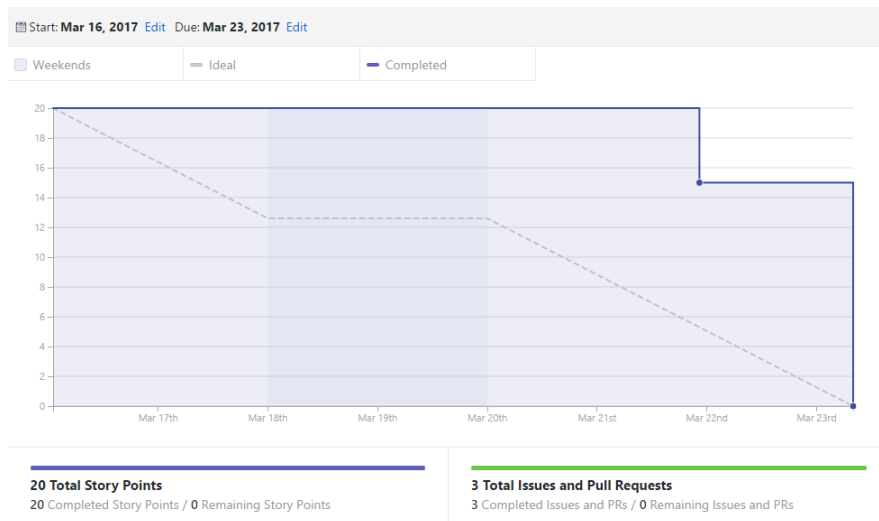


Figura A.14: Progreso de resolución de tareas del séptimo sprint

Octavo Sprint (23 de Marzo - 30 de Marzo):

La tarea más importante realizada esa semana fue la de hacer otra aproximación al web scrapping basada en obtener los datos a partir de los RSS de los medios en los que, hasta el momento, se estaban leyendo desde las páginas web convencionales. El resultado fue un pequeño programa que te permitía seleccionar un medio de una lista de diarios y una palabra clave y, en tiempo real, realizar el web scrapping del RSS de ese medio y almacenarlo en la base de datos a la vez que se mostraba en una tabla los resultados obtenidos. Para ello se usó la librería feedparser. Este acercamiento era mucho más rápido que el web scrapping convencional, pero a cambio sacrifica la posibilidad de poder acceder a los comentarios de las noticias, y los casos de algunos medios, no todo el texto de la noticia es accesible a través del RSS. Como ha sido

mencionado en el resumen del sprint anterior, debido a la falta de capacidad en la máquina virtual en la que se estaba trabajando, no se podía descargar todo el conjunto de datos de NLTK de forma correcta, lo cual se intentó solucionar inicialmente clonando la máquina virtual a una de almacenamiento dinámico y editando la capacidad de almacenamiento de las particiones de la misma. Esta solución no tuvo resultados positivos, por tanto, se reinstaló desde el principio una nueva máquina virtual que evitara estos problemas y que además corrigiera la instalación de algunas librerías que habían producido problemas previamente. Por último, se probó una nueva librería de gráficos que sustituyera a Seaborn, denominada Bqplot, la cual se asemejaba en funcionalidad a Bokeh, pero evitando esa necesidad de archivos HTML que nos habían llevado a descartar Bokeh en un primer momento. Las pruebas fueron muy positivas en gráficos de barras y lineales.

Completed Issues and Pull Requests			Story points
🔗	Prueba de Web Scrapping basada en RSS code investigation	Python_DataClassification #21 III New Issues 🔗 Octavo Sprint	(8)
🔗	Prueba de interfaz con bqplot code enhancement investigation	Python_DataClassification #22 III New Issues 🔗 Octavo Sprint	(5)
🔗	Clonar y editar almacenamiento máquina virtual enhancement	Python_DataClassification #24 III New Issues 🔗 Octavo Sprint	(3)
🔗	Reinstalar Máquina Virtual bug enhancement	Python_DataClassification #25 III New Issues 🔗 Octavo Sprint	(5)

Figura A.15: Issues del octavo sprint

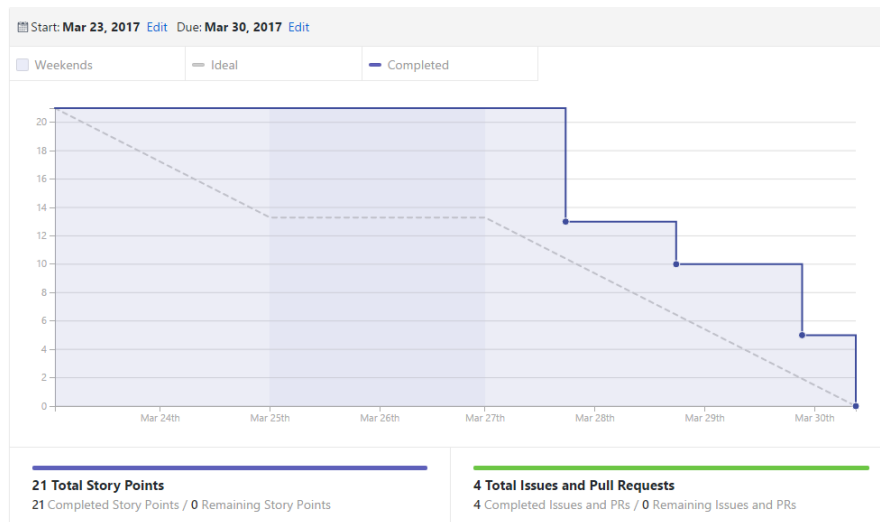


Figura A.16: Progreso de resolución de tareas del octavo sprint

Noveno Sprint (31 de Marzo - 13 de Abril):

La duración de este sprint fue el doble de lo normal (dos semanas en vez de una) debido a que coincidió con las vacaciones de Semana Santa. Por tanto,

el día que debía haberse realizado la reunión habitual no era lectivo, y hubo que posponerlo. Por otro lado, como había suficientes tareas/horas de trabajo como para dividirlo en más de un sprint, se dio por finalizado el sprint a los 14 días aunque la siguiente reunión fuera después de 21. En lo relacionado a tareas relacionadas con sprints anteriores, se consiguió solucionar el problema por el cual no se aplicaban los scripts de JavaScript, aunque para ello hubo que buscar una forma alternativa de implementarles. Además, se consiguió finalizar con éxito la instalación y pruebas con TextBlob, llevando a cabo la instalación del NLTK de una manera alternativa que ocupaba mucho menos espacio en memoria. Una vez instalado y probado, se hizo un sencillo prototipo con un conjunto de datos de prueba minúsculo de diferenciador de frases machistas y feministas, tanto con TextBlob como con SciKitLearn, aprovechando código de scripts ya existentes. También se mejoraron las operaciones de almacenamiento en base de datos, evitando introducir noticias duplicadas, para que el usuario final no se tenga que preocupar de filtrar las que estén duplicadas de las que no. En lo respectivo a la publicación del notebook como aplicación web, se realizaron instalaciones de varias herramientas de Jupyter Dashboards y de Docker, pero debido a impedimentos de configuración de la BIOS y del anfitrión, no se pudo avanzar más en la máquina virtual, y se decidió repetir el proceso en el anfitrión en el siguiente sprint. Por último, se añadió el diario “El Mundo” a los disponibles en el web scrapping por RSS, para ir aumentando el rango ideológico de las alternativas disponibles.

Completed Issues and Pull Requests			Story points
🔗	Pruebas de publicación 2 - Docker <code>code</code> <code>investigation</code>		13
	Python_DataClassification #18 III New Issues ↑ Noveno Sprint		
🔗	Prueba de TextBlob (Parte 2) <code>code</code> <code>investigation</code>		8
	Python_DataClassification #23 III New Issues ↑ Noveno Sprint		
🔗	Solucionar problemas Scripts <code>bug</code> <code>code</code> <code>enhancement</code>		13
	Python_DataClassification #26 III New Issues ↑ Noveno Sprint		
🔗	Actualizar la base de datos sin duplicadas <code>code</code> <code>enhancement</code>		3
	Python_DataClassification #27 III New Issues ↑ Noveno Sprint		
🔗	Prototipo de Clasificación de frases <code>code</code> <code>investigation</code>		13
	Python_DataClassification #28 III New Issues ↑ Noveno Sprint		
🔗	RSS el Mundo <code>code</code> <code>enhancement</code>		3
	Python_DataClassification #29 III New Issues ↑ Noveno Sprint		

Figura A.17: Issues del noveno sprint

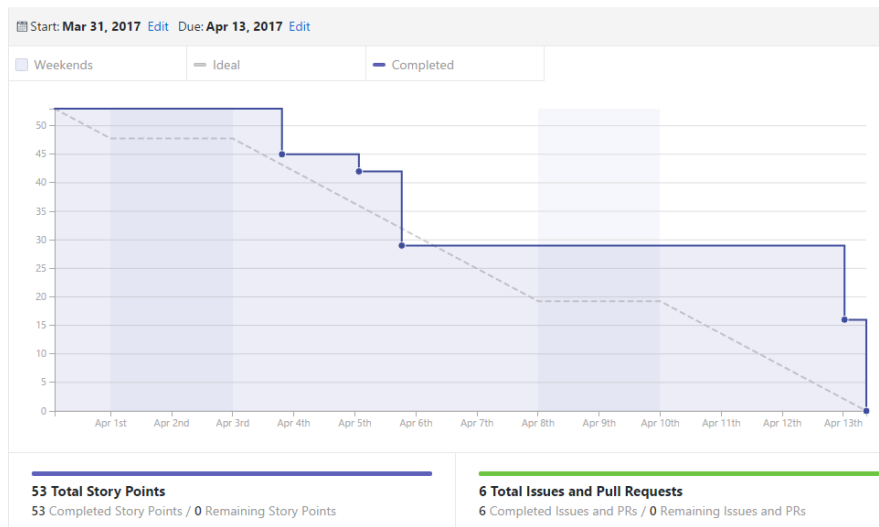


Figura A.18: Progreso de resolución de tareas del noveno sprint

Décimo Sprint (14 de Abril - 21 de Abril):

En este periodo se continuó la investigación y las pruebas de despliegue de aplicación de los notebook en una máquina de Docker, aunque en esta ocasión se realizaron en el Sistema Operativo anfitrión (Windows) en vez de en la máquina virtual, para comprobar si al cambiar el entorno conseguíamos solucionar los problemas técnicos sufridos durante la configuración de Docker.

Se realizaron además las primeras pruebas de clasificación de textos con un dataset real, compuesto por un grupo de noticias de diferentes medios de comunicación tratando el debate de los vientos de alquiler, etiquetadas a partes iguales como *“A Favor”* y *“En Contra”* del tema que hablaban. Las pruebas fueron realizadas tanto en TextBlob, dividiéndolas en frases, como en SciKit-Learn, dividiéndolas en palabras.

También se dedicó parte de la semana a la documentación del trabajo realizado hasta la fecha.

Completed Issues and Pull Requests	Story points
<p>Documentar avances realizados (Semanas 0-9) documentation</p> <p>Python_DataClassification #12 III New Issues ↑ Décimo Sprint</p>	8
<p>Pruebas de publicación (Docker) - Anfitrión enhancement investigation</p> <p>Python_DataClassification #30 III New Issues ↑ Décimo Sprint</p>	13
<p>Primeras clasificaciones con DataSet real code enhancement</p> <p>Python_DataClassification #31 III New Issues ↑ Décimo Sprint</p>	8

Figura A.19: Issues del décimo sprint

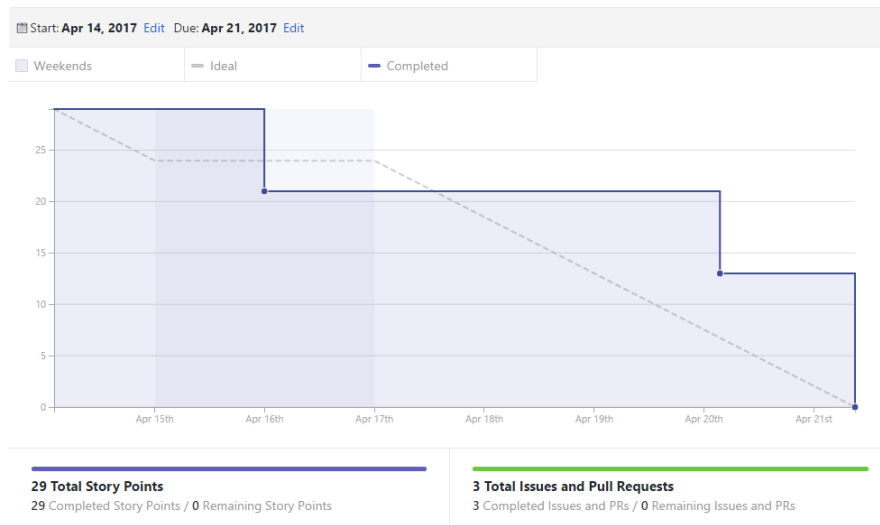


Figura A.20: Progreso de resolución de tareas del décimo sprint

Undécimo Sprint (22 de Abril - 27 de Abril):

Tras los fracasos en la configuración sufridos con Docker, se investigó la alternativa de Cloud Foundry, plataforma open source basada en contenedores que da soporte a la subida de aplicaciones a la nube.

En lo respectivo a la clasificación del Dataset real, se continuaron las pruebas, en este caso probando distintos permutadores (Leave One Out, Leave P Out y ShuffleSplit), para hacer validación cruzada y separar las noticias que teníamos en conjuntos de entrenamiento y conjunto de prueba.

Además, se investigó la librería de Lime, que permite crear explicaciones detalladas de las predicciones, y se realizaron los primeros experimentos con la misma.

Completed Issues and Pull Requests				Story points
Ⓢ	Pruebas de publicación (Cloud Foundry)	enhancement	investigation	15
	Python_DataClassification #33 III New Issues ↑ Undécimo Sprint			
Ⓢ	Mejorar clasificación Sklearn	code	enhancement	8
	Python_DataClassification #34 III New Issues ↑ Undécimo Sprint			
Ⓢ	Probar Lime	code	enhancement	5
	Python_DataClassification #35 III New Issues ↑ Undécimo Sprint			

Figura A.21: Issues del undécimo sprint

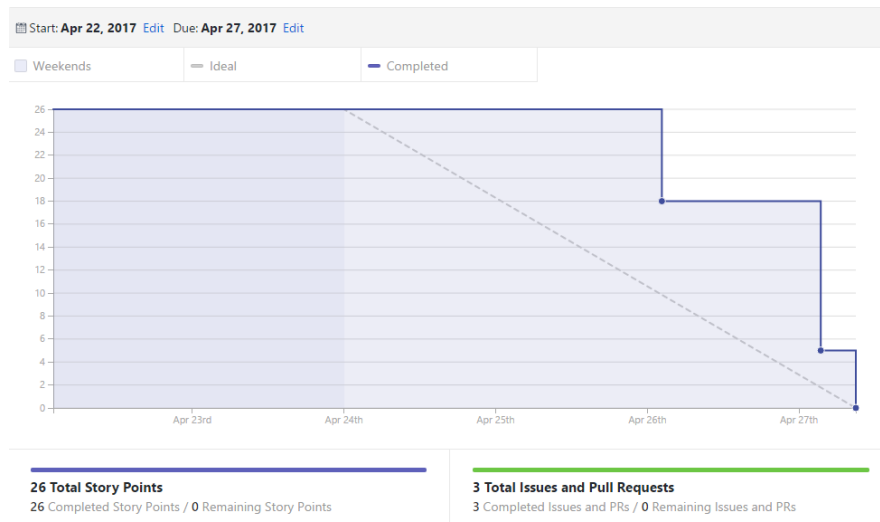


Figura A.22: Progreso de resolución de tareas del undécimo sprint

Duodécimo Sprint (28 de Abril - 4 de Mayo):

En este punto, se decidió dar más prioridad a crear una aplicación con funcionalidad completa que a seguir invirtiendo más tiempo en intentar configurar un contenedor para poder subir los notebooks de Jupyter con formato de aplicación web.

Por ello, se comenzó el desarrollo de una aplicación con Flask que reuniera toda la funcionalidad implementada hasta el momento, que estaba repartida en varios archivos sin relación entre ellos, con objetivo de desplegarla en modo local en lugar de subirla a un servidor.

Se creó un esqueleto con la funcionalidad más básica de lectura de PDF, Web Scraping y persistencia en una base de datos NoSQL, sobre la que construir en las siguientes semanas.

Completed Issues and Pull Requests	Story points
<p>Crear proyecto flask code investigation</p> <p>Python_DataClassification #37 III New Issues ↑ Duodécimo Sprint</p>	2
<p>Añadir Funcionalidad leer pdf al prototipo en Flask code enhancement investigation</p> <p>Python_DataClassification #38 III New Issues ↑ Duodécimo Sprint</p>	6
<p>Añadir funcionalidad web scrapping al prototipo en Flask code enhancement investigation</p> <p>Python_DataClassification #39 III New Issues ↑ Duodécimo Sprint</p>	6
<p>Añadir funcionalidad clasificación de datos al prototipo en Flask code enhancement investigation</p> <p>Python_DataClassification #40 III New Issues ↑ Duodécimo Sprint</p>	6
<p>Añadir funcionalidad PyMongo al prototipo en Flask code enhancement investigation</p> <p>Python_DataClassification #41 III New Issues ↑ Duodécimo Sprint</p>	6

Figura A.23: Issues del duodécimo sprint

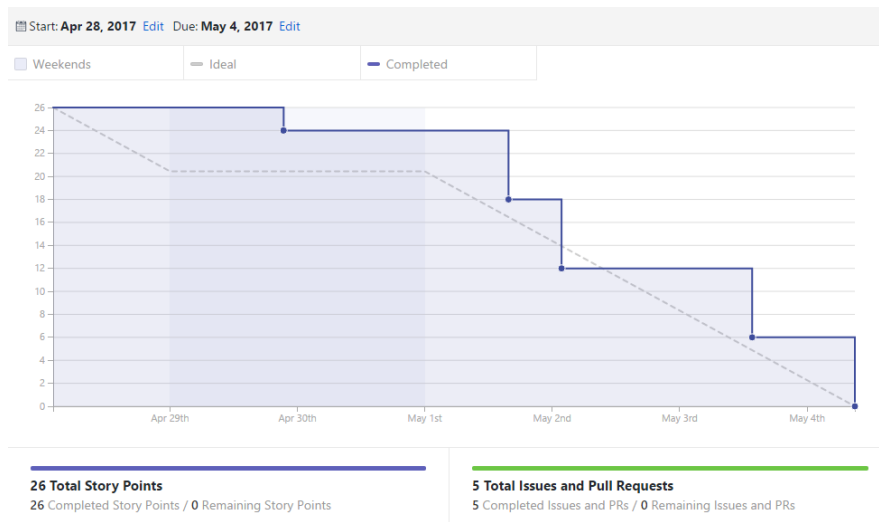


Figura A.24: Progreso de resolución de tareas del duodécimo sprint

Decimotercer Sprint (5 de Mayo - 11 de Mayo):

Durante esta semana se continuó la construcción de la aplicación en Flask, como la implementación de las inserciones desde la web a la base de datos, el traslado del código de clasificación de texto de Jupyter a Flask y una mejora en la lectura de PDF.

Aparte de esto, se dedicó un tiempo a solucionar problemas más pequeños como correcciones en el uso de rutas literales, sustituir el uso de librerías obsoletas por sus alternativas vigentes, la inclusión de un fichero .txt con las librerías necesarias para ejecutar la aplicación, y completar la experimentación de Lime con los ejemplos de la documentación.

Completed Issues and Pull Requests	Story points
<p>🔗 Añadir funcionalidad clasificación (Parte 2) <code>code</code> <code>enhancement</code> Python_DataClassification #42 III New Issues 🔼 Decimotercer sprint</p>	6
<p>🔗 Implementar inserciones a base de datos desde la aplicación web <code>code</code> <code>enhancement</code> Python_DataClassification #43 III New Issues 🔼 Decimotercer sprint</p>	3
<p>🔗 Evitar lecturas redundantes de pdf <code>code</code> <code>enhancement</code> Python_DataClassification #44 III New Issues 🔼 Decimotercer sprint</p>	3
<p>🔗 Completar aprendizaje de Lime <code>code</code> <code>investigation</code> Python_DataClassification #45 III New Issues 🔼 Decimotercer sprint</p>	6
<p>🔗 Evitar el uso de deprecated <code>code</code> <code>enhancement</code> Python_DataClassification #47 III New Issues 🔼 Decimotercer sprint</p>	0.5
<p>🔗 Incluir el archivo requirements.txt <code>code</code> <code>enhancement</code> Python_DataClassification #48 III New Issues 🔼 Decimotercer sprint</p>	1
<p>🔗 Evitar el uso de rutas literales <code>code</code> <code>enhancement</code> Python_DataClassification #49 III New Issues 🔼 Decimotercer sprint</p>	0.5
<p>🔗 Añadir declaración de encoding en los archivos .py <code>code</code> <code>enhancement</code> Python_DataClassification #50 III New Issues 🔼 Decimotercer sprint</p>	0.5
<p>🔗 Evitar el uso de literales que dependen del entorno local <code>code</code> <code>enhancement</code> Python_DataClassification #51 III New Issues 🔼 Decimotercer sprint</p>	1
<p>🔗 Evitar la creación de rutas a archivos mediante manipulación directa de cadenas <code>bug</code> <code>code</code> Python_DataClassification #52 III New Issues 🔼 Decimotercer sprint</p>	0.5

Figura A.25: Issues del decimotercer sprint

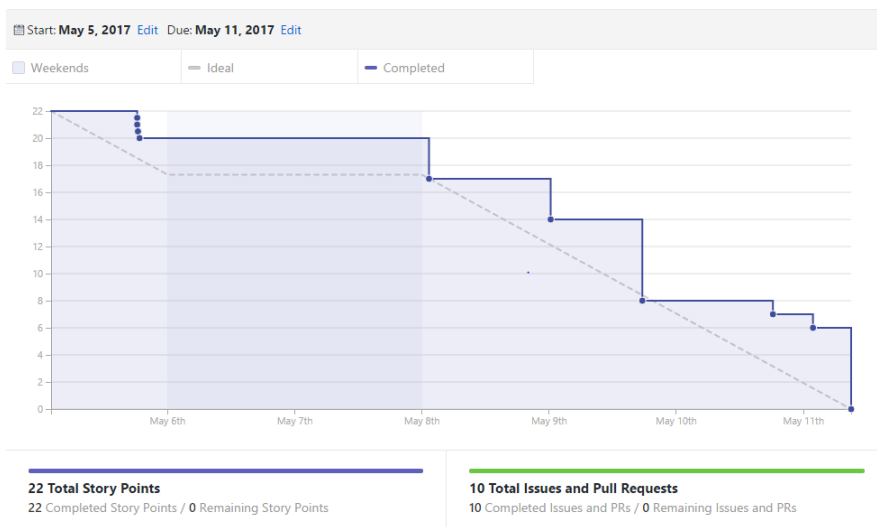


Figura A.26: Progreso de resolución de tareas del decimotercer sprint

Decimocuarto Sprint (12 de Mayo - 18 de Mayo):

Este sprint se centró en avanzar todo lo posible la funcionalidad relacionada con la clasificación de texto. Concretamente, se consiguió desde la librería de NLTK una lista de palabras irrelevantes en español a tener en cuenta para evitar ruido en los entrenamientos y predicciones.

En lo respectivo a la aplicación web, también se centraron esfuerzos en la vista de resultados de noticias y las posibilidades de etiquetado de las mismas,

permitiendo que las noticias tuvieran varios datasets, que se pudiera añadir un dataset entero mediante un documento XML y se hizo una primera aproximación a un etiquetado manual, pero que sería sustituido por una alternativa más coherente en la siguiente semana.

Completed Issues and Pull Requests				Story points
🔗	Buscar StopWords en Español	code	enhancement	3
	Python_DataClassification #56 III New Issues ⬆ Decimocuarto Sprint			
🔗	Implementar varios Dataset por Noticia	code	enhancement	8
	Python_DataClassification #57 III New Issues ⬆ Decimocuarto Sprint			
🔗	Posibilidad de añadir un dataset entero mediante documentos	code	enhancement	8
	Python_DataClassification #58 III New Issues ⬆ Decimocuarto Sprint			
🔗	Funcionalidad Clasificación (Parte 3)	code	enhancement	6
	Python_DataClassification #59 III New Issues ⬆ Decimocuarto Sprint			
🔗	Añadir etiquetas de forma manual a las noticias	code	enhancement	3
	Python_DataClassification #60 III New Issues ⬆ Decimocuarto Sprint			

Figura A.27: Issues del decimocuarto sprint

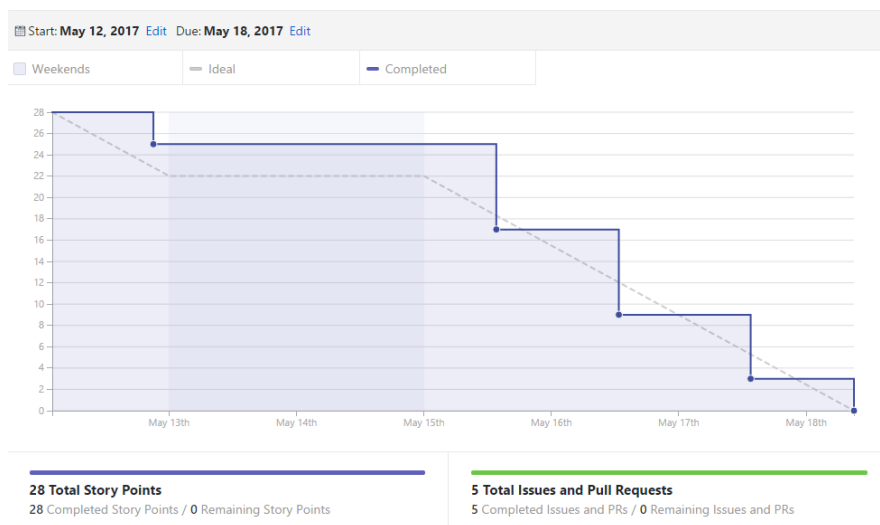


Figura A.28: Progreso de resolución de tareas del decimocuarto sprint

Decimoquinto Sprint (19 de Mayo - 25 de Mayo):

En esta semana se completó casi toda la funcionalidad restante en la tabla de resultados de noticias, incluyendo una barra de progreso que indicaba las probabilidades que tenía una noticia de pertenecer a una clase o a otra, la posibilidad de que el usuario realice un etiquetado manual a cada noticia, y la posibilidad de ver la explicación de la predicción en una pestaña aparte, acompañada de un gráfico para comprender mejor los resultados.

También se dedicó parte del tiempo a mejorar la coherencia de urls para

separar distintas funcionalidades en distintas pestañas, para evitar sobrecargar de complejidad las peticiones de los formularios, sobre todo en las pestañas de lectura de PDF y XML.

Completed Issues and Pull Requests				Story points
🔖	Mostrar barra de vista previa de probabilidades.	code enhancement		5
	Python_DataClassification #61 III New Issues ⬆ Decimoquinto Sprint			
🔖	Cambiar modo de etiquetado manual	code enhancement		5
	Python_DataClassification #63 III New Issues ⬆ Decimoquinto Sprint			
🔖	Mejorar coherencia de url y funcionalidad (Pestaña de PDF y XML)	code enhancement		5
	Python_DataClassification #64 III New Issues ⬆ Decimoquinto Sprint			
🔖	Añadir gráficos a la explicación	code enhancement investigation		3
	Python_DataClassification #65 III New Issues ⬆ Decimoquinto Sprint			
🔖	Mejorar sistema de clasificación	code enhancement		5
	Python_DataClassification #66 III New Issues ⬆ Decimoquinto Sprint			
🔖	Cambiar titular por índice como elemento diferenciador de noticias	code enhancement		3
	Python_DataClassification #67 III New Issues ⬆ Decimoquinto Sprint			

Figura A.29: Issues del decimoquinto sprint

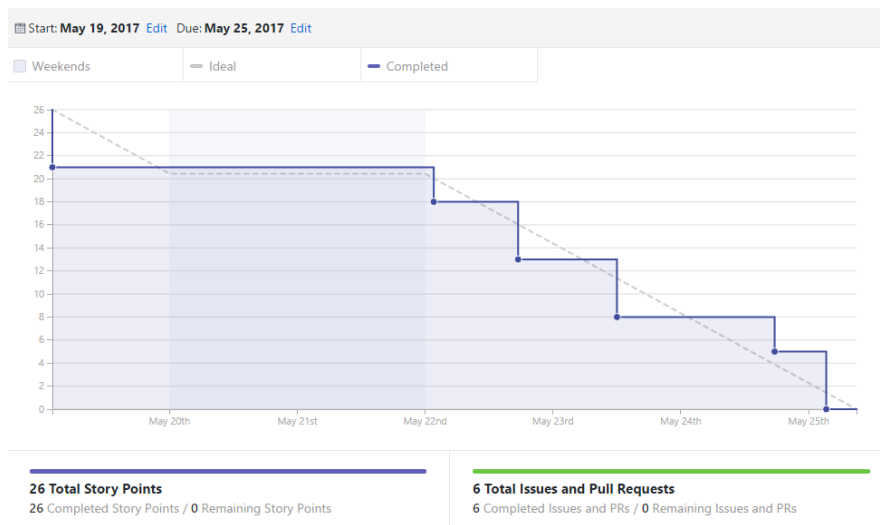


Figura A.30: Progreso de resolución de tareas del decimoquinto sprint

Decimosexto Sprint (26 de Mayo - 2 de Junio):

Teniendo gran parte de la funcionalidad implementada, en este Sprint se centró en mejorar el estilo gráfico de la interfaz de la aplicación. Para ello, se usó Bootstrap, primero experimentando con la versión para Flask y posteriormente se decidió permanecer con el modo de uso que tendría en un proyecto web normal, importando los archivos CSS y JS correspondientes. Aprovechando que había que utilizar JQuery para poder aplicar Bootstrap, se refactorizó el código existente de JavaScript en JQuery, reduciendo en un gran tamaño el

mismo.

Además del estilo, también se actualizó el modo de lectura de los PDF, creando una estructura dentro de los archivos simple que facilitara la posterior lectura e inserción en base de datos, obligando a todos los documentos PDF que sigan la misma estructura para que puedan ser leídos.

También se modificó la vista de resultados de la explicación de Lime, aprovechando su característica de poder mostrar una vista de resultados propia exportada como HTML, sin tener que recurrir a librerías de gráficos de terceros.

Completed Issues and Pull Requests				Story points
🔖	Estructura PDF <code>code</code> <code>enhancement</code>			3
	Gestor-de-Hemeroteca #17 III New Issues ↑ Decimosexto sprint			
🔖	Hacer que el conjunto de entrenamiento se actualice progresivamente <code>code</code> <code>enhancement</code>			3
	Gestor-de-Hemeroteca #62 III New Issues ↑ Decimosexto sprint			
🔖	Aplicar Bootstrap a la aplicación <code>enhancement</code> <code>investigation</code>			8
	Gestor-de-Hemeroteca #68 III New Issues ↑ Decimosexto sprint			
🔖	Añadir breadcrumbs a las pestañas			3
	Gestor-de-Hemeroteca #69 III New Issues ↑ Decimosexto sprint			
🔖	Refactorizar funciones de JavaScript con JQuery <code>code</code> <code>enhancement</code>			3
	Gestor-de-Hemeroteca #70 III New Issues ↑ Decimosexto sprint			
🔖	Exportar la explicación de Lime como html <code>code</code> <code>enhancement</code> <code>investigation</code>			3
	Gestor-de-Hemeroteca #71 III New Issues ↑ Decimosexto sprint			
🔖	Guardar la predicción desde la pestaña de explicación <code>code</code> <code>enhancement</code>			3
	Gestor-de-Hemeroteca #72 III New Issues ↑ Decimosexto sprint			
🔖	Arreglar codificación de los nombres de autores PDF <code>bug</code> <code>code</code>			3
	Gestor-de-Hemeroteca #73 III New Issues ↑ Decimosexto sprint			
🔖	Eliminar acentos de los textos en las operaciones <code>code</code> <code>enhancement</code>			3
	Gestor-de-Hemeroteca #75 III New Issues ↑ Decimosexto sprint			

Figura A.31: Issues del decimosexto sprint

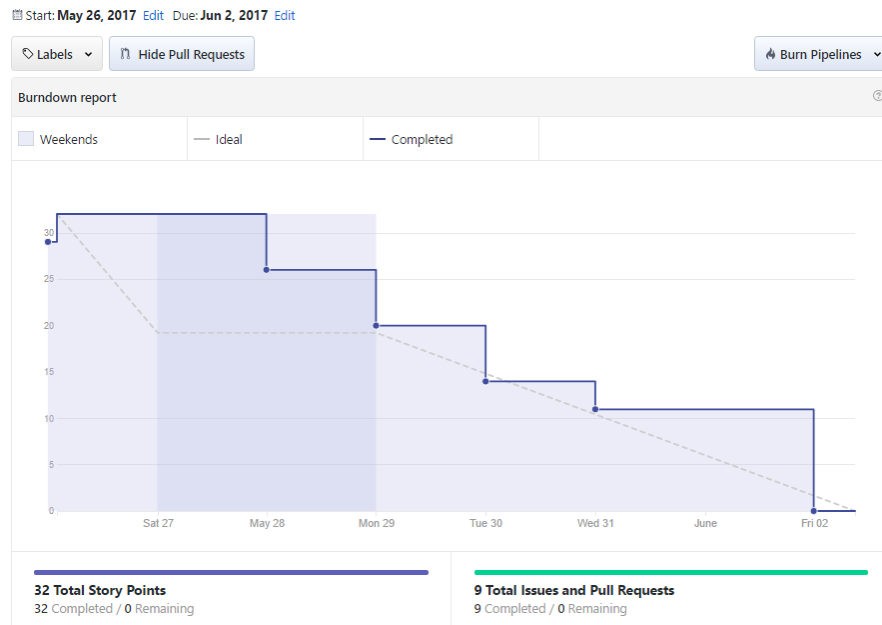


Figura A.32: Progreso de resolución de tareas del decimosexto sprint

Decimoséptimo Sprint (2 de Junio - 9 de Junio):

A estas alturas ya se estaban realizando las últimas mejoras en la funcionalidad, como una actualización de la lectura de los RSS de las páginas web, de los XML y de los PDF. Aparte de esto, se terminaron los capítulos de la memoria dedicado a los conceptos teóricos y objetivos del proyecto.

Completed Issues and Pull Requests	Story points
<p>Documentar conceptos teóricos documentation</p> <p>Gestor-de-Hemeroteca #36 III New Issues ↑ Decimoseptimo sprint</p>	6
<p>Mejorar lectura de Dataset code enhancement</p> <p>Gestor-de-Hemeroteca #74 III New Issues ↑ Decimoseptimo sprint</p>	5
<p>Mejorar lectura RSS bug code enhancement</p> <p>Gestor-de-Hemeroteca #76 III New Issues ↑ Decimoseptimo sprint</p>	5
<p>Crear nuevos dataset code enhancement</p> <p>Gestor-de-Hemeroteca #79 III New Issues ↑ Decimoseptimo sprint</p>	8
<p>Documentar Objetivos documentation</p> <p>Gestor-de-Hemeroteca #80 III New Issues ↑ Decimoseptimo sprint</p>	2

Figura A.33: Issues del decimoseptimo sprint

Completed Issues and Pull Requests	Story points
<p>Documentar herramientas utilizadas documentation</p> <p>Gestor-de-Hemeroteca #55 III New Issues ⬆ Decimo octavo sprint</p>	13
<p>Funcionalidad "Ver más" en la tabla de resultados code enhancement</p> <p>Gestor-de-Hemeroteca #77 III New Issues ⬆ Decimo octavo sprint</p>	6
<p>Añadir visualización de gráficos en la tabla de resultados code enhancement investigation</p> <p>Gestor-de-Hemeroteca #78 III New Issues ⬆ Decimo octavo sprint</p>	6
<p>Arreglar problema de coherencia de dataset y noticias bug code enhancement</p> <p>Gestor-de-Hemeroteca #81 III New Issues ⬆ Decimo octavo sprint</p>	5
<p>Crear fichero de arranque code enhancement</p> <p>Gestor-de-Hemeroteca #82 III New Issues ⬆ Decimo octavo sprint</p>	1
<p>Documentar aspectos relevantes documentation</p> <p>Gestor-de-Hemeroteca #83 III New Issues ⬆ Decimo octavo sprint</p>	8
<p>Documentar Conclusiones documentation</p> <p>Gestor-de-Hemeroteca #85 III New Issues ⬆ Decimo octavo sprint</p>	2

Figura A.35: Issues del decimoctavo sprint

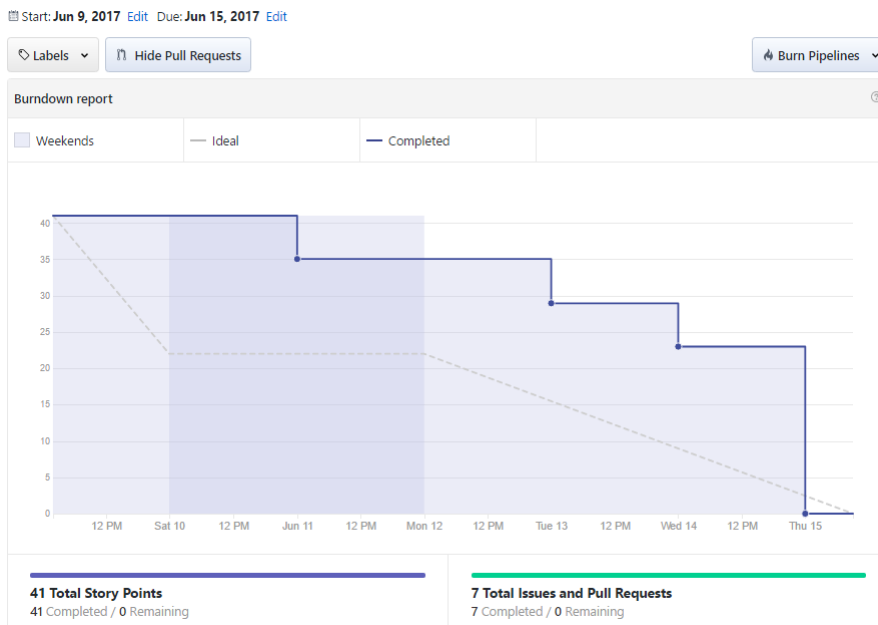


Figura A.36: Progreso de resolución de tareas del decimoctavo sprint

Decimonoveno Sprint (16 de Junio - 22 de Junio):

Habiendo finalizado la funcionalidad, los objetivos de esta semana se enfocaron en realizar la documentación de los anexos y terminar la de la memoria, a la vez que se realizaban pruebas sobre el código y se realizó refactorización sobre el mismo.

También se arreglaron algunos problemas relacionados con el almacenamiento de nuevas noticias.

Completed Issues and Pull Requests			Story points
Documentar avances realizados (A partir semana 9)	documentation		5
Gestor-de-Hemeroteca #32 III New Issues ↑ Decimonoveno Sprint			
Documentar Introducción	documentation		Not estimated
Gestor-de-Hemeroteca #84 III New Issues ↑ Decimonoveno Sprint			
Documentar trabajos relacionados	documentation		3
Gestor-de-Hemeroteca #86 III New Issues ↑ Decimonoveno Sprint			
Refactorizar Código	code enhancement		3
Gestor-de-Hemeroteca #87 III New Issues ↑ Decimonoveno Sprint			
Documentar Requisitos	documentation		8
Gestor-de-Hemeroteca #89 III New Issues ↑ Decimonoveno Sprint			
Añadir gifs de carga	code enhancement		3
Gestor-de-Hemeroteca #93 III New Issues ↑ Decimonoveno Sprint			
Arreglar "Guardar Nuevas"	bug code enhancement		2
Gestor-de-Hemeroteca #94 III New Issues ↑ Decimonoveno Sprint			

Figura A.37: Issues del decimonoveno sprint

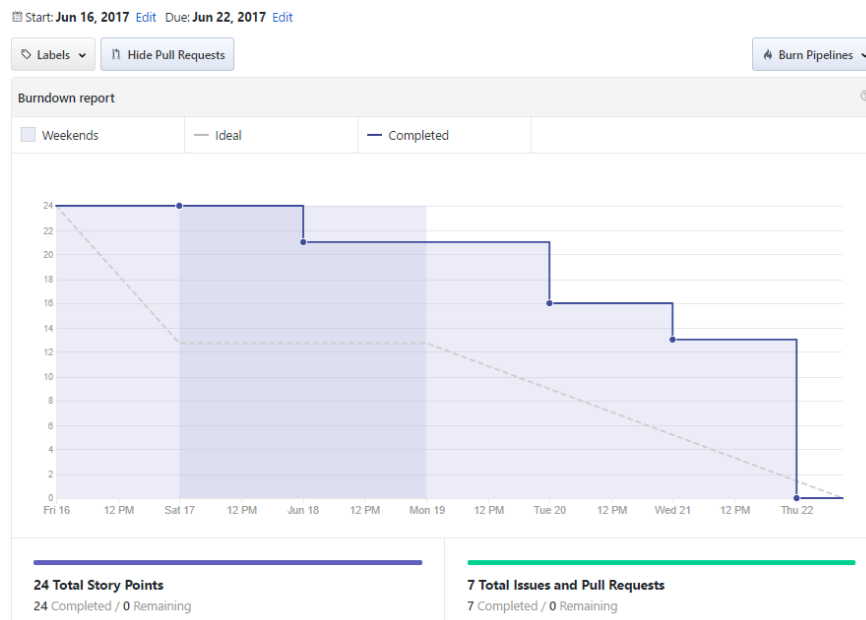


Figura A.38: Progreso de resolución de tareas del decimonoveno sprint

A.3. Estudio de viabilidad

Viabilidad económica

Para poder hacer un uso correcto de la aplicación, no hay que hacer ninguna inversión económica en software de ningún tipo, ya que todas las librerías que se han usado son open-source.

Más allá de lo comprendido por la propia herramienta, el único coste im-

prescindible sería el del propio equipo en el que se instale y configure. Aunque algunas funcionalidades tengan tiempos de carga más notables que otras, no debería ser impedimento para que la aplicación funcionara correctamente en equipos de diferentes prestaciones. Respecto al Sistema Operativo, como la aplicación se ejecuta en sistemas Linux, no es necesario ni realizar la compra de un Sistema Operativo de pago.

Cabe destacar que en el contexto de este proyecto la aplicación no está publicada en un servidor web, pero que en un futuro podría llevarse a cabo esta operación. De ser así, habría que evaluar diferentes alternativas de host compatibles y comprobar si alguna de estas alternativas son de pago o gratuitas.

Viabilidad económica del desarrollo

En lo relacionado con costes de personal, hay varios datos de los que se podrían obtener algunos presupuestos hipotéticos.

Por ejemplo, según la información proporcionada por *ZenHub* en el repositorio del proyecto, se estima que se han invertido 556 horas en el desarrollo, a lo largo de los 6 meses del mismo.

Si se propusiera un sueldo para el programador de 10€ la hora, tendríamos el siguiente cálculo:

$$10/\text{hora} * 556\text{horas} = 5560$$

De esta cantidad inicial, habría que reflexionar acerca de cuánto tiempo ha sido dedicado a programar la aplicación final, y cuánto tiempo fue dedicado a la investigación y programación de funcionalidades que no acabaron incluidas en el producto, ya que podría considerarse que esas horas no podrían considerarse parte del desarrollo de la herramienta, y por tanto ser descontadas de la cantidad calculada.

Viabilidad legal

A continuación, se expondrán de forma breve las licencias de todas las herramientas y librerías utilizadas en la creación del producto final.

Tabla A.1: Tabla de bibliotecas y sus licencias

Librería	Versión	Descripción	Licencia
Flask	0.12.2	Biblioteca para crear aplicaciones web en Python.	BSD
Jinja2	2.8	Motor para integrar Python documentos HTML.	BSD
Pymongo	3.4.0	Extensión de MongoDB (Base de datos NoSQL) para Python	Apache license v2.0
Scikit-Learn	0.18	Biblioteca para aprendizaje automático en Python.	BSD
Lime	(Desconocida)	biblioteca Python para crear explicaciones de clasificaciones.	BSD 2-clause
Beautiful Soup	4.6.0	Biblioteca para hacer web scraping en Python.	MIT
PDFMiner	(Desconocida)	Biblioteca para lectura de PDF en Python.	OSI Approved
Feedparser	5.2.1	Biblioteca Python para hacer lectura de RSS.	OSI Approved
Bootstrap	3.3.7	Biblioteca para desarrollar aplicaciones web responsive.	MIT
JQuery	3.2.1	Biblioteca para optimizar JavaScript.	MIT
Chart.js	2.0	Biblioteca para crear gráficos en Javascript y HTML.	MIT

Especificación de Requisitos

B.1. Introducción

En esta sección se describirán las funciones para las que la herramienta fue desarrollada. Se hará una detallada definición tanto de objetivos como de sus requisitos (funcionales y no funcionales) y también se hará un estudio en profundidad de los casos de uso.

B.2. Objetivos generales

Como ya se describió previamente en a memoria, hay un conjunto de objetivos en torno a los cuales se estructura el desarrollo del proyecto. A continuación se volverán a resumir los relacionados con la funcionalidad y las técnicas empleadas:

- El principal objetivo es crear una aplicación web en la que el usuario pueda almacenar noticias tanto por medio de web scraping como lectura de PDF.
- Se busca que el usuario pueda añadir etiquetas relacionadas con diferentes temas de debate a estas noticias, tanto manualmente como guardando una predicción realizada con aprendizaje automático.
- Otro objetivo es que el usuario pueda añadir temas de debate nuevos que contengan dos clases, para luego poder etiquetar estas noticias en función de estas nuevas clases.
- En caso de que el usuario ya tuviera un corpus preparado antes de comenzar a usar la aplicación, tiene la posibilidad de añadir un corpus mediante un archivo XML con todas las noticias que lo forman y sus etiquetas correspondientes.

- La aplicación debe ser responsive en los diferentes equipos en los que se use, y fácil de usar para nuevos usuarios.

B.3. Catalogo de requisitos

Requisitos Funcionales

Estos son los requisitos funcionales que la herramienta debe cumplir:

- RF - 1: Los usuarios deben poder visualizar un conjunto de noticias a partir de la búsqueda por palabra clave.
- RF - 2: Los usuarios podrán etiquetar manualmente las noticias en función del tema de debate seleccionado.
 - RF - 2.1: Los usuarios podrán elegir de una lista de opciones el tema de debate mostrado en ese momento.
- RF - 3: Los usuarios pueden ver una explicación gráfica para la predicción de la etiqueta de una noticia.
 - RF - 3.1: Los usuarios también podrán guardar esa predicción como la etiqueta final de la noticia.
- RF - 4: Los usuarios podrán guardar las noticias guardadas en archivos PDF almacenados en una carpeta concreta del proyecto.
- RF - 5: Los usuarios podrán guardar nuevas noticias de los RSS de varios medios de comunicación españoles.
 - RF - 5.1: Los usuarios podrán elegir de una lista de opciones el medio de comunicación del que ver las nuevas noticias.
- RF - 6: Los usuarios podrán añadir un corpus de noticias entero, indicando el tema de debate y la etiqueta de cada noticia, a través de un archivo XML.
- RF - 7: Los usuarios podrán crear nuevos temas de debate, escribiendo el nombre del propio tema y el nombre de dos clases, que se usarán posteriormente de etiquetas para ese tema.

Requisitos No Funcionales

Por otro lado, estos son los requisitos no relacionados con la funcionalidad que se deben tener en cuenta:

- RNF - 1: La interfaz de las pestañas no debe impedir su correcto funcionamiento en pantallas de diferentes tamaños.
- RNF - 2: A pesar de que se produzcan picos de carga de trabajo, el funcionamiento general no debe producir tiempos de carga excesivos.
- RNF - 3: La herramienta debe poder ejecutarse en cualquier navegador (siempre que soporten librerías como Bootstrap o JQuery).

B.4. Especificación de requisitos

A continuación, se describirán con detalle los casos de uso de los que se compone el proyecto.

Diagrama de casos de uso

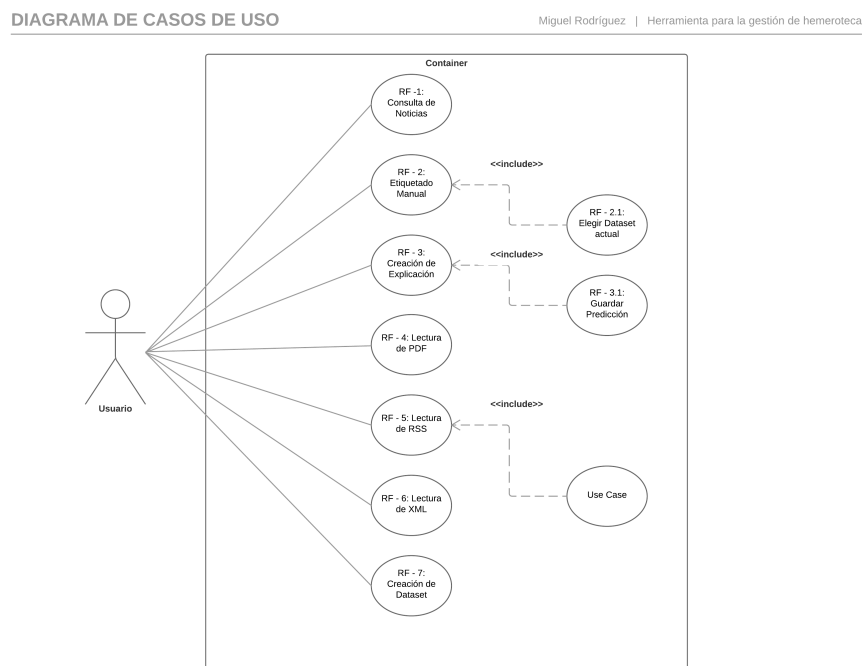


Figura B.1: Diagrama de casos de uso

Descripción de casos de uso

CU1 - Consulta de Noticias

Caso de uso	Consulta de Noticias
Autor	Miguel Rodríguez
Versión	1.0
Requisitos	RF - 1: Los usuarios deben poder visualizar un conjunto de noticias a partir de la búsqueda por palabra clave
Descripción	El usuario introduce una palabra clave para la búsqueda de las noticias que la contengan
Precondiciones	Las noticias tienen que estar almacenadas en la base de datos. La palabra debe aparecer en la noticia para que sea devuelta.
Acciones	En la pestaña inicial, el usuario introduce en el cuadro de texto la palabra clave con la que quiere realizar la búsqueda y pulsa el botón “Buscar”
Postcondiciones	Se cargará una tabla con las noticias que contenían la palabra
Excepciones	En caso de que la palabra no se encuentre en ninguna noticia, no se devolverán filas de la tabla de resultados
Importancia	Alta

Tabla B.1: Caso de uso 1

CU2 - Etiquetado Manual

Caso de uso	Etiquetado Manual
Versión	1.0
Autor	Miguel Rodríguez
Requisitos	RF - 2: Los usuarios pueden etiquetar manualmente las noticias en función del tema de debate seleccionado RF - 2.1: Los usuarios podrán elegir de una lista de opciones el tema de debate mostrado en ese momento
Descripción	El usuario selecciona un tema de debate y una de las dos clases para la noticia que quiera etiquetar
Precondiciones	La noticia que se va a etiquetar tiene que estar en la tabla de resultados. El tema de debate sobre el que se quiera etiquetar tiene que existir previamente.
Acciones	El usuario selecciona de una lista desplegable el tema de debate en el que quiera etiquetar las noticias. Una vez seleccionado, pulsando el botón de “Etiquetar” de cada fila podrá marcar cuál de las dos etiquetas quiere aplicar a esa noticia. Pulsando en “Guardar Cambios”, se actualizarán todas las etiquetas modificadas.
Postcondiciones	Se añadirá o etiquetará esa etiqueta a esa noticia. Esa noticia pasa a formar parte del corpus de entrenamiento para ese tema de debate.
Excepciones	Si se realizan etiquetados pero no se guardan los resultados, no se aplicarán los cambios
Importancia	Alta

Tabla B.2: Caso de uso 2

CU3 - Creación de Explicación

Caso de uso	Creación de explicación
Versión	1.0
Autor	Miguel Rodríguez
Requisitos	RF - 3: Los usuarios pueden ver una explicación gráfica para la predicción de la etiqueta de una noticia. RF - 3.1: Los usuarios también podrán guardar esa predicción como la etiqueta final de la noticia.
Descripción	El usuario selecciona la opción de explicación para una noticia de la tabla de resultados.
Precondiciones	La noticia que se va a etiquetar tiene que estar en la tabla de resultados. Debe existir un corpus de entrenamiento para el tema de debate sobre el que se va a realizar la predicción.
Acciones	El usuario selecciona de una lista desplegable el tema de debate en el que quiera etiquetar las noticias. Una vez seleccionado, pulsando el botón de “Clasificar” de una noticia en concreto cargará una página con la explicación correspondiente. Pulsando en “Guardar Predicción”, se guardará el resultado de la predicción como etiqueta.
Postcondiciones	Se etiquetará la noticia con el valor de la clase ganadora en la predicción
Excepciones	Si no hay un corpus de entrenamiento, la explicación no se cargará correctamente
Importancia	Alta

Tabla B.3: Caso de uso 3

CU4 - Lectura de PDF

Caso de uso	Lectura de PDF
Versión	1.0
Autor	Miguel Rodríguez
Requisitos	RF - 4: Los usuarios podrán guardar las noticias guardadas en archivos PDF almacenados en una carpeta concreta del proyecto
Descripción	Se muestra una lista de noticias leídas en archivos PDF y el usuario escoge si guardar las nuevas o no.
Precondiciones	Debe haber archivos PDF que contengan noticias en el directorio adecuado. Además, deben de seguir el formato especificado para que se puedan leer correctamente.
Acciones	En la pestaña de lectura de PDF, el usuario pulsará el botón de leer noticias. Cuando se hayan leído y se muestre la lista, el usuario puede pulsar el botón de “Guardar Nuevas” para almacenar las noticias no previamente guardadas.
Postcondiciones	Se almacenarán las noticias no guardadas previamente en la base de datos.
Excepciones	Si no hay archivos PDF no se mostrará nada en la tabla de resultados. Si no tienen el formato indicado, no se podrá leer esas noticias.
Importancia	Media

Tabla B.4: Caso de uso 4

CU5 - Lectura de RSS

Caso de uso	Lectura de RSS
Versión	1.0
Autor	Miguel Rodríguez
Requisitos	RF - 5: Los usuarios podrán guardar nuevas noticias de los RSS de varios medios de comunicación españoles. RF - 5.1: Los usuarios podrán elegir de una lista de opciones el medio de comunicación del que ver las nuevas noticias.
Descripción	El usuario elige un medio de comunicación y después puede guardar las nuevas noticias de ese medio.
Precondiciones	El medio de comunicación tiene que ser uno de los incluidos en la lista. Hay que tener conexión a internet para acceder a los RSS de estos medios.
Acciones	En la pestaña de lectura de RSS, el usuario elegirá en un desplegable el medio de comunicación del que quiere leer la noticia. Pulsando el botón de “Guardar Nuevas”, el usuario podrá almacenar las que no estuvieran ya en la base de datos.
Postcondiciones	Se mostrará una tabla con las últimas noticias del medio de comunicación seleccionado.
Excepciones	En caso de que no haya conexión a Internet, la lectura de noticias no se podrá realizar.
Importancia	Alta

Tabla B.5: Caso de uso 5

CU6 - Lectura de XML

Caso de uso	Lectura de XML
Versión	1.0
Autor	Miguel Rodríguez
Requisitos	RF - 6: Los usuarios podrán añadir un corpus de noticias entero, indicando el tema de debate y la etiqueta de cada noticia, a través de un archivo XML.
Descripción	El usuario ordena leer los documentos XML que contienen los corpus situados en el directorio especificado.
Precondiciones	Los documentos XML tienen que tener el formato determinado.
Acciones	En la pestaña de lectura de nuevos conjuntos, el usuario pulsará el botón de “Leer”. Se cargará una nueva pestaña donde si el usuario pulsa “Guardar Nuevas”, se almacenarán las noticias, con las etiquetas ya aplicadas de antemano.
Postcondiciones	Se guardarán las nuevas noticias con etiquetas ya aplicadas, al contrario que en los otros formatos.
Excepciones	En caso de que no hubiera archivos XML o que no tengan el formato correcto, la lectura no se realizará correctamente.
Importancia	Alta

Tabla B.6: Caso de uso 6

CU7 - Creación de Dataset

Caso de uso	Creación de dataset
Versión	1.0
Autor	Miguel Rodríguez
Requisitos	RF - 7: Los usuarios podrán crear nuevos temas de debate, escribiendo el nombre del propio tema y el nombre de dos clases, que se usarán posteriormente de etiquetas para ese tema.
Descripción	El usuario tiene la opción de introducir nuevos dataset para las noticias indicando su nombre y el de sus 2 posibles clases.
Precondiciones	Los temas de debate deben de tener 2 clases posibles.
Acciones	En la pestaña inicial, el usuario debe pulsar el botón de “Añadir Nuevo Dataset”. En la pestaña a la que ha sido redirigido, escribirá el nombre del nuevo dataset y el de sus dos clases, que serán las etiquetas que luego se podrán añadir a las noticias. Al pulsar el botón de guardar cambios, se actualizará la base de datos con esta información.
Postcondiciones	Se añadirá a la lista de datasets el escrito en la aplicación, y a todas las noticias almacenadas también se les añadirán campos para soportar las nuevas etiquetas.
Excepciones	En caso de que no se rellenen los tres campos, no se guardará correctamente el nuevo dataset.
Importancia	Alta

Tabla B.7: Caso de uso 7

Especificación de diseño

C.1. Introducción

En este apartado se analizará el diseño de la aplicación desde las diferentes perspectivas que componen el sistema. Se hablará del diseño de los datos, del flujo de ejecución de la aplicación y del diseño de la estructura de la misma.

C.2. Diseño de datos

Como ya se ha mencionado en la memoria, la base de datos que se usa en la aplicación es NoSQL, lo cual quiere decir que no tendrá un sistema relacional de tablas como es costumbre.

También hay que destacar que, como la herramienta no está pensada para el uso abierto a cualquier usuario, si no a uno en concreto, no se consideró necesario hacer un sistema de registro de usuarios, con lo cual las colecciones/tablas habituales con la información de los usuarios aquí son inexistentes.

Sin embargo, si que existen dos colecciones (Se entiende por colección el equivalente a una tabla en SQL) en torno a las cuáles se estructura el almacenamiento de la información. A continuación describimos qué contienen cada una de ellas:

- **Noticias:** Es la colección que almacena los datos importantes de cada una de las noticias guardadas en la hemeroteca.

Los campos de esta colección son los siguientes:

- **Title:** El titular de la noticia.
- **Author:** El autor de la noticia.
- **Text:** El texto de la noticia.

- **PublishDate:** La fecha de publicación de la noticia.
 - **Source:** La fuente de la noticia.
 - **Link:** Ubicación o url de la noticia.
 - **Tag:** El conjunto de etiquetas para cada dataset de la noticia. El formato es el de diccionario: dentro de esta variable hay un conjunto de claves representando los conjuntos de datos y unos valores que representan las etiquetas de esa noticia para esos conjuntos.
- **Datasets:** Es la colección que almacena los diferentes temas de debate sobre los que luego se podrá etiquetar y realizar predicciones con las noticias, siendo las etiquetas el nombre de las clases.

Los campos que contiene son éstos:

- **Dataset:** El nombre del tema de debate.
- **Clase1:** El nombre de la primera clase.
- **Clase2:** El nombre de la segunda clase.

C.3. Diseño procedimental

Teniendo en cuenta que la mayoría de casos de uso de la aplicación son independientes entre ellos, y que no hay ningún orden específico de realización de los mismos, vamos a mostrar un diagrama de flujo de los más cruciales que son los relacionados con el etiquetado de noticias:

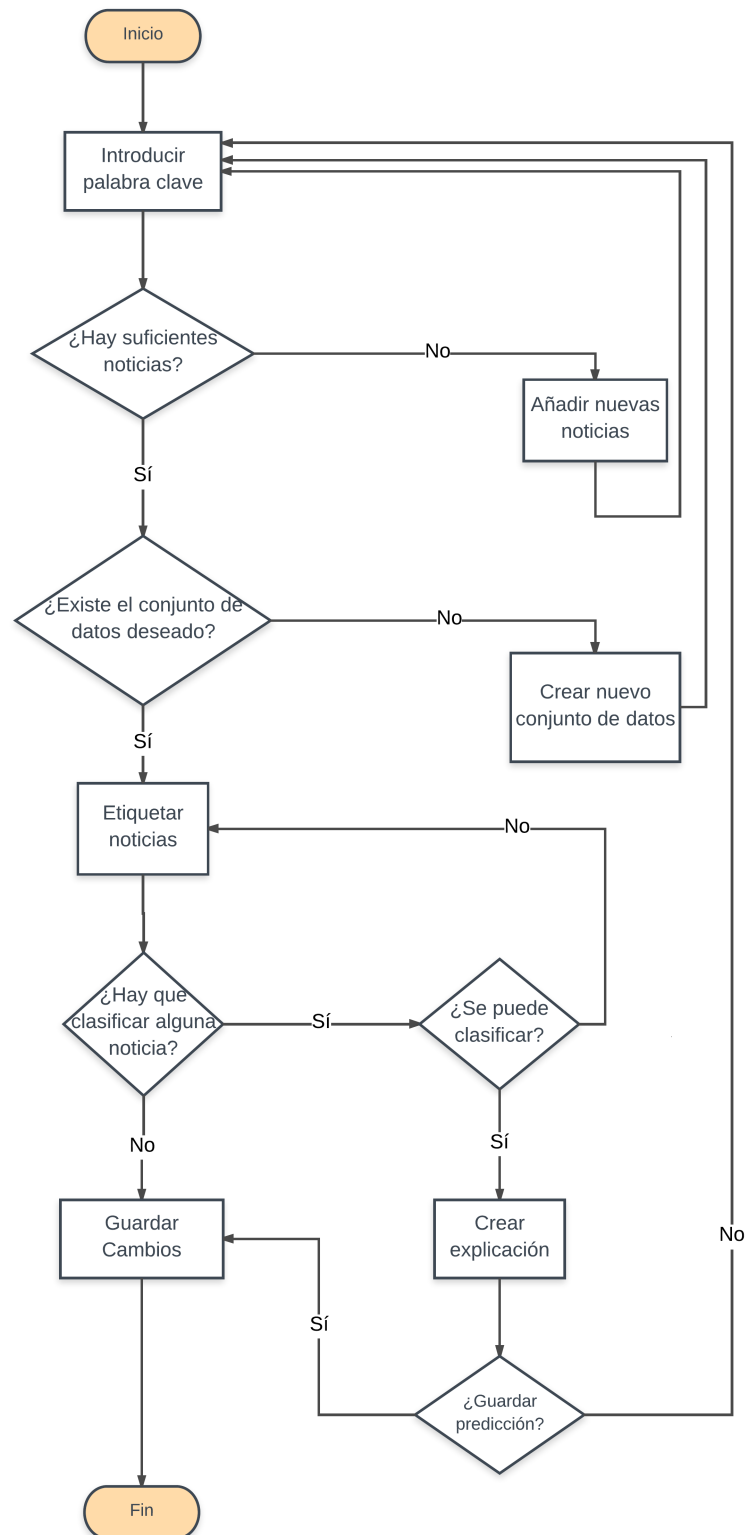


Figura C.1: Diagrama de flujo para un caso de interacción habitual

C.4. Diseño arquitectónico

Aunque se desarrollará con más profundidad en el siguiente apartado, cabe destacar que todo el proyecto se ubica bajo un mismo directorio. En esta ubicación, se encuentra, aparte del directorio que contiene el código de la aplicación, otros en los que se almacenan las noticias de PDF, los corpus en XML, y las pruebas de casos de uso realizadas con Selenium.

En el siguiente diagrama se muestra como interaccionan los paquetes entre ellos.

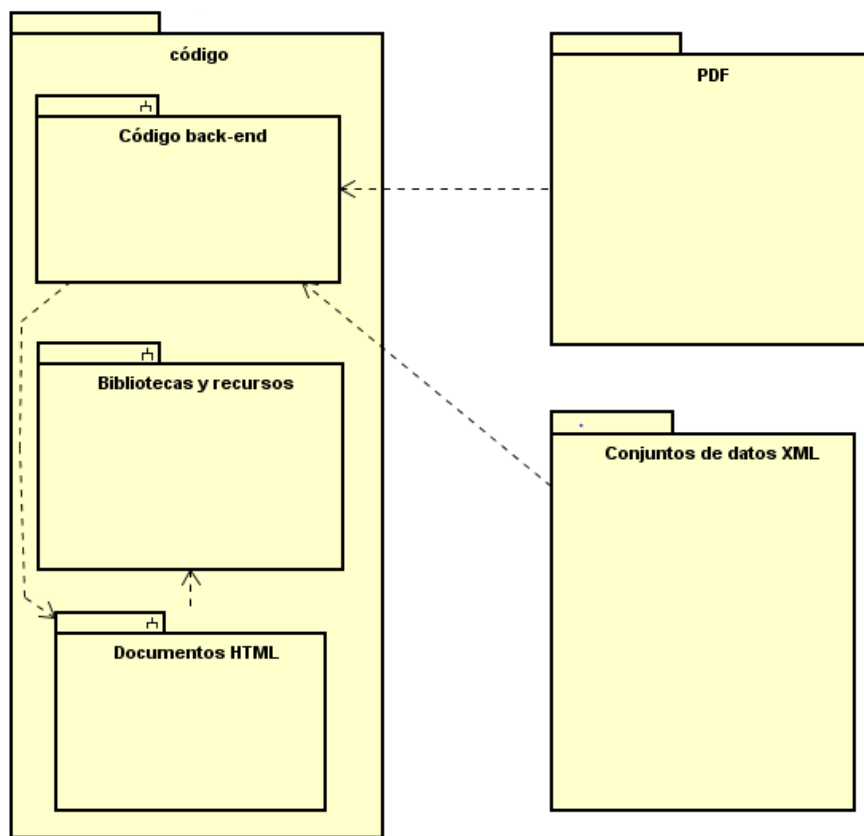


Figura C.2: Diagrama de paquetes

Dentro del directorio *app*, encontramos los archivos de Python que manejan la funcionalidad de la herramienta. Al contrario que una aplicación de escritorio, al ser una aplicación web construida con Flask, no hay un conjunto de clases interconectadas que construyan el sistema, si no que la mayor parte de la funcionalidad está implementada en funciones escritas en Python conectadas con cada una de las pestañas de la herramienta.

No obstante, si que hay un pequeño grupo de clases de las que hacen uso las funciones que se han mencionado previamente, y que son las siguientes:

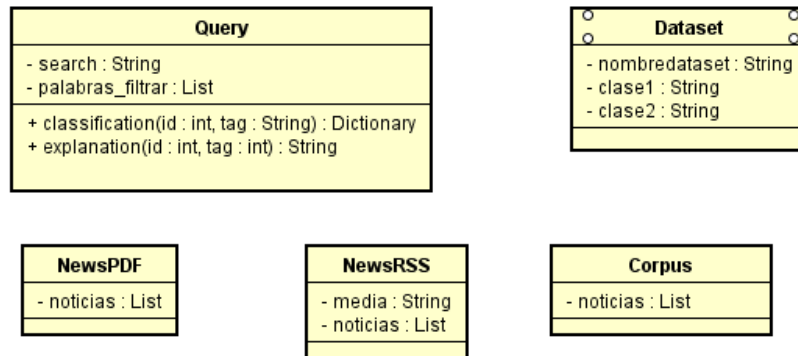


Figura C.3: Clases utilizadas en la herramienta

Como se puede observar, la mayoría no tienen ni siquiera métodos debido a que su papel es la de hacer las funciones de formularios, donde los atributos van a ser campos para introducir datos desde la aplicación, o para almacenar información que va a ser devuelta por pantalla. Tampoco hay relación entre las clases porque cada una es llamada por la función de una pestaña distinta. La clase *Query*, sin embargo, sí que tiene dos métodos y son cruciales, pues implementan toda la funcionalidad relacionada con el aprendizaje automático, y es usada por las pestañas de consulta de noticias y creación de explicaciones.

Documentación técnica de programación

D.1. Introducción

En esta sección se explicarán todos los apartados necesarios para que, en el caso de que un programador externo entrara a trabajar en esta aplicación, pudiera familiarizarse rápido con ella y su desarrollo.

D.2. Estructura de directorios

Prototipo Flask es el directorio en el que se ubican todos los archivos de la herramienta. En esta carpeta podemos encontrar los siguientes directorios:

- **app:** Es el directorio más importante de la propia aplicación, ya que contiene todos los archivos de *back-end* (scripts de Python con la funcionalidad) y *front-end* (documentos HTML para las pestañas) que forman la herramienta, junto con las carpetas con los estilos CSS, los fuentes de *Bootstrap*, *JQuery* y alguna biblioteca adicional, y las imágenes que se usan en las pestañas.
- **datasets:** En este directorio es donde se deben de dejar los archivos XML que contengan corpus de noticias enteros. Este formato está pensado para casos en los que el usuario ya tuviera un conjunto de noticias con las clases asignadas de forma previa al uso de la herramienta.
- **pdf:** En esta ubicación se deben de depositar los archivos PDF que sigan el formato indicado para su posterior lectura por parte de la aplicación.

- **tests:** Aquí se encuentran los ficheros de código Python con las pruebas en Selenium para los casos de uso. Con la perspectiva de probarles por separado, se creó un archivo con una prueba para cada caso de uso.

Aparte de lo anterior, en el directorio raíz tenemos los siguientes archivos:

- *config.py*: Fichero que contiene las variables de configuración (En este caso, las rutas a las carpetas donde se ubican los PDF, XML... etc)s.
- *requirements.txt*: Este fichero de texto plano contiene una lista de todos los recursos usados, para que cuando se instale la aplicación en otro equipo se puedan instalar a la vez usando este archivo en vez de uno a uno.
- *run.py*: Este código de Python es el que arranca la aplicación, haciendo uso del directorio *app*.
- *setup.sh*: Contiene un conjunto de comandos simples, como el arranque del entorno virtual y de la base de datos, para que sólo sea necesario ejecutar este archivo antes de usar la aplicación.

D.3. Manual del programador

En este apartado se mencionarán los distintos puntos a tener en cuenta en relación al desarrollo de la aplicación.

Minería de datos

Todo lo relacionado con el aprendizaje automático de los textos de las noticias se encuentra en el script *forms.py*, que se encuentra dentro de la carpeta *app*.

En la clase *Query*, que es la que se llama desde la aplicación para estas operaciones, tenemos dos métodos principales: *classification* y *explanation*. Si se quisiera modificar la predicción de noticias que se realiza en la tabla de resultados, habría que cambiar el código de la primera función, mientras que si se quieren hacer cambios en las explicaciones se deberá modificar la segunda función.

En lo que respecta al modelo, se refactorizó código extrayendo en una función apartada, llamada *prepare model*, el preparado de los datos de entrenamiento a partir de las noticias.

A continuación se muestra una captura del código donde se puede observar esta estructura.

```
def prepare_model(training_data, tag):

class Query(Form):
    """Clase usada en la generación de la tabla de resultados y explicaciones"""
    noticiasSample = []
    search = StringField('')
    def classification(self, noticia_id, tag):
    def explanation(self, noticia_id, tag):
```

Figura D.1: Ubicación del código utilizado en el aprendizaje automático

Lectura de noticias

Toda la funcionalidad relacionada con el almacenamiento y obtención de noticias, e interacción con la web se encuentra en el fichero *views.py*. Allí, se puede encontrar una función para cada pestaña de la aplicación, ocupada de la funcionalidad *back-end*.

En la siguiente imagen se muestran estas funciones.

```

77
78 @app.route('/')
79 @app.route('/index/', methods=['GET', 'POST'])
80 @register_breadcrumb(app, '.', 'Inicio')
81 def index():
179
180
181 @app.route('/statistics/', methods=['GET', 'POST'])
182 def statistics():
197
198
199 @app.route('/explanation/<string:arguments>', methods=['GET', 'POST'])
200 @register_breadcrumb(app, '.Explicacion', 'Explicacion')
201 def explanation(arguments):
238
239
240 @app.route('/createdataset', methods=['GET', 'POST'])
241 @register_breadcrumb(app, '.Nuevo Dataset', 'Nuevo Dataset')
242 def createdataset():
260
261
262 @app.route('/scanpdf', methods=['GET', 'POST'])
263 @register_breadcrumb(app, '.Lectura PDF', 'Lectura PDF')
264 def scanpdf():
268
269
270 @app.route('/savepdfnews', methods=['GET', 'POST'])
271 @register_breadcrumb(app, '.Resultados PDF', 'Resultados PDF')
272 def savepdfnews():
355
356 @app.route('/websearch', methods=['GET', 'POST'])
357 @register_breadcrumb(app, '.Busqueda RSS', 'Busqueda RSS')
358 def websearch():
462
463
464 @app.route('/newdataset', methods=['GET', 'POST'])
465 @register_breadcrumb(app, '.Lectura Dataset', 'Lectura Dataset')
466 def newdataset():
469
470
471 @app.route('/savenewdataset', methods=['GET', 'POST'])
472 @register_breadcrumb(app, '.Resultados Dataset', 'Resultados Dataset')
473 def savenewdataset():
496
497
498 if __name__ == '__main__':
499     app.run(host='0.0.0.0', port=5000)
500

```

Figura D.2: Funciones encargadas de la interacción con las pestañas

D.4. Compilación, instalación y ejecución del proyecto

En los siguientes puntos se van a detallar qué elementos son imprescindibles para la configuración e instalación de la aplicación. Este proceso es muy similar al que se realiza en el Mega-Tutorial de Flask de Miguel Gringberg [2].

Python

Es esencial que en el equipo que vayamos a usar la herramienta esté Python instalado. Más concretamente, la versión 2.7 es en la que se ha desarrollado nuestro código.

En las últimas versiones de los Sistemas Operativos Ubuntu, ya viene una versión de Python 2 y Python 3 instalada, por lo tanto no requiere acciones

extra por nuestra parte.

Pip

Otra herramienta muy útil a la hora de instalar bibliotecas de Python es Pip. Se puede instalar simplemente con el siguiente comando:

```
sudo apt-get install python-pip
```

NLTK

Para poder usar las palabras vacías, necesitamos descargarnos el corpus *stopwords* de nltk, para ello, en un script Python, basta con los siguientes comandos:

```
import nltk
nltk.download()
```

Se abrirá el menú de instalación de nltk, y desde ahí se podrá elegir que biblioteca queremos descargarnos (en este caso, la llamada “stopwords”).

Pymongo

Antes de comenzar la instalación de la aplicación, hay que configurar en el equipo la base de datos NoSQL. Antes de la extensión de Python, necesitamos tener MongoDB en nuestro equipo, lo cual conseguiremos siguiendo los pasos que indican en la guía de la página oficial de MongoDB [1], en este enlace: <https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-ubuntu/>.

Para instalar Pymongo, teniendo Pip en el equipo basta con escribir lo siguiente:

```
sudo python -m pip install pymongo
```

Para activar el servicio de MongoDB en el Sistema Operativo sólo hace falta escribir en la terminal:

```
sudo service mongod start
```

Entorno Virtual

Para poder usar Flask, se creará un entorno virtual en el que se va a realizar la instalación de todo lo relacionado con Flask y se ejecutará la aplicación.

Para poder crear este entorno virtual, lo primero que hay que hacer es instalar *virtualenv*, con el siguiente comando:

```
sudo apt-get install python-virtualenv
```

Flask

Una vez esté instalado *virtualenv*, desde la ventana de comandos habrá que situarse en la ubicación del directorio raíz del proyecto y ejecutar el siguiente comando:

```
virtualenv flask
```

Aprovechando que hay un archivo llamado *requirements* que contiene todas las bibliotecas que se han usado en el desarrollo, basta con el siguiente comando para que se instalen todas en orden sin necesidad de instalarlas una a una:

```
sudo flask/bin/pip install -r requirements.txt
```

Una vez está instalado todo lo necesario, hay que modificar los permisos del fichero *run.py* para que nos permita ejecutarle. Se hace de la siguiente manera:

```
sudo chmod a+x run.py
```

Una vez sea accesible este archivo, ya se podría ejecutar la aplicación. Es crucial mencionar que la primera vez que se ejecuta la aplicación en un ordenador nuevo, no va a tener ningún dataset, y por tanto se debe usar el script *createdataset.py*, ubicado en el directorio raíz, para crear los dataset iniciales con los que poder empezar a etiquetar noticias.

D.5. Pruebas del sistema

Pruebas de funcionalidad

Aparte de las pruebas convencionales de funcionamiento realizadas por usuarios, se han creado un conjunto de pruebas simples automatizadas con la herramienta *Selenium*.

No obstante, al contrario de su uso habitual, que consistiría en grabar un conjunto de pasos con el plugin del navegador y exportarlo en algún lenguaje como Java, hemos usado las bibliotecas de *Selenium* para Python, evitando así usar diferentes lenguajes de programación, aún a costa de no usar el plugin para Mozilla Firefox.

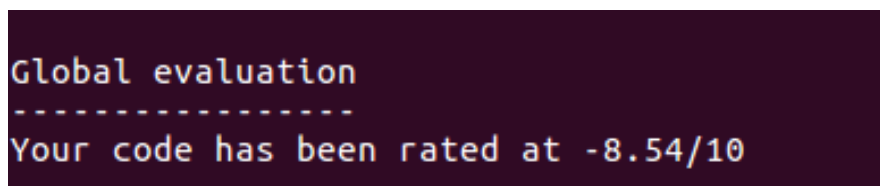
En la carpeta *tests* del proyecto, se pueden encontrar un conjunto de scripts numerados del 1 al 7, que contienen una prueba simple para cada caso de uso. Para evitar hacer cambios innecesarios en la base de datos, se ha evitado ejecutar la funcionalidad de guardar nuevas noticias en las pruebas, ya que si por ejemplo, en el caso de uso de lectura de RSS, se estuvieran haciendo pruebas constantes, podría llenarse la base de datos de noticias que no se querían almacenar en un primer momento.

Además de esto, hay un script auxiliar llamado *deletedataset* que se puede usar para borrar un conjunto de datos de la aplicación. El motivo de que esté guardado es que, realizando pruebas, se acababan almacenando demasiados conjuntos de pruebas innecesarios en la aplicación, y esto podía afectar al rendimiento. Por tanto, ejecutando este script con el nombre del conjunto que queramos borrar ahorra bastante trabajo al programador que esté probando la herramienta.

Calidad del código

Además de las pruebas de funcionalidad, se llevó a cabo una evolución progresiva de la calidad del código usando la herramienta *Pylint*, que detecta todos los errores, avisos y mejoras de refactorización que se pueden realizar en programas escritos en Python. Esta herramienta se usó tanto en *views.py* como en *forms.py*, que son los dos scripts que reúnen la mayoría de la funcionalidad de la aplicación.

Por ejemplo, en las siguientes imágenes podemos ver la evolución de calidad del código del fichero *views* en función de la nota recibida por *Pylint*:



```
Global evaluation
-----
Your code has been rated at -8.54/10
```

Figura D.3: Nota obtenida antes de realizar refactorizaciones

```
Global evaluation
-----
Your code has been rated at 7.70/10 (previous run: 7.70/10, +0.00)
```

Figura D.4: Nota obtenida después de varias semanas, aplicando algunas técnicas de refactorización

```
Global evaluation
-----
Your code has been rated at 7.99/10 (previous run: 7.99/10, +0.00)
```

Figura D.5: Nota obtenida en los últimos días de desarrollo

message id	occurrences
bad-continuation	27
invalid-name	17
bare-except	8
line-too-long	6
too-many-nested-blocks	5
too-many-locals	4
too-many-statements	3
too-many-branches	3
broad-except	3
redefined-builtin	1
missing-final-newline	1
import-error	1

Figura D.6: Algunos de los factores que valora Pylint

Del mismo modo, podemos observar de forma breve la evolución de *forms* a lo largo de las semanas, aplicando algunas técnicas de refactorización:


```
Global evaluation
-----
Your code has been rated at -6.10/10
```

Figura D.7: Nota obtenida antes de aplicar cambios

```
Global evaluation
-----
Your code has been rated at 6.36/10 (previous run: 3.21/10, +3.15)
```

Figura D.8: Nota después de los primeros cambios

```
Global evaluation
-----
Your code has been rated at 8.15/10 (previous run: 6.36/10, +1.79)
```

Figura D.9: Nota después de haber aplicado la mayoría de los cambios

Cabe destacar que, aunque sigue habiendo margen de mejora, era imposible obtener un 10/10 con esta herramienta, pues es conflictiva en algunos aspectos. Por poner un ejemplo, evalúa con mala nota las líneas con más de 100 columnas de código, pero si se introduce un salto de línea para dividir la longitud, penaliza que haya un salto de línea inapropiado.

Documentación de usuario

E.1. Introducción

En este apartado se explicarán las condiciones que tiene que cumplir el equipo en el que se ejecute la aplicación, además de mostrar visualmente cómo se debe usar la misma.

E.2. Requisitos de usuarios

Software

Aunque en el apartado “Manual del programador” se explica con más detalle qué programas y librerías son necesarias para ejecutar la aplicación, los resumiremos en un listado breve como el siguiente:

- Sistema Operativo: La herramienta se ha desarrollado y está pensada para utilizarse en Ubuntu y, más específicamente, la versión 16.
- Navegador Web: Durante el desarrollo se utilizó Mozilla Firefox, que además es el que usa *Selenium* para realizar las pruebas. No obstante, la herramienta ha sido probada en otros navegadores como el incluido en Ubuntu por defecto, y funciona correctamente.
- Python: Ya viene incluido en Ubuntu. Si aun así se deseara descargar, habría que elegir la versión 2.7.
- Pip: Para realizar la instalación de la mayoría de bibliotecas que usa la aplicación.
- MongoDB: Necesario para poder montar una base de datos con las noticias.

- Virtualenv: El entorno virtual en el que se instalarán todo y desde donde se ejecutará la aplicación.
- NLTK: Al menos, el corpus correspondiente a las palabras vacías.

Hardware

En lo relacionado con los aspectos técnicos del equipo en el que se ejecute la aplicación, no hay que tener en cuenta requisitos especialmente exigentes. Si que hay que contemplar que, a mejor *hardware* esté disponible, las operaciones más costosas se realizarán con más facilidad.

Por tomar un punto de referencia mínimo, vamos a citar los requisitos hardware de Mozilla Firefox para los Sistemas Operativos convencionales [3]:

- Procesador: Pentium 4 o superior
- Tamaño de Memoria RAM: 512MB (Si se va a ejecutar en una máquina virtual, habrá que controlar que el equipo tenga suficiente RAM asignada a la misma para que funcione correctamente).

En términos de almacenamiento, se recomienda dejar libre alrededor de 1 GB para poder instalar varias de las bibliotecas adicionales que vienen incluidas en *requirements*. No ocupan tanto espacio, pero lo más aconsejable es dejar un margen por si se implementaran nuevas funcionalidades o se llegaran a instalar nnuevas bibliotecas de terceros.

E.3. Instalación

El conjunto de elementos que hay que instalar para poder ejecutar la aplicación (Python, MongoDB ... etc) viene explicado con detalle en el apartado “Manual del Programador”, pues los pasos a seguir son los mismos.

Habiendo realizado todas las acciones mencionadas antes, lo único necesario ahora sería ejecutar el archivo *setup.sh* que hay en el directorio raíz, con el siguiente comando:

```
sudo bash setup.sh
```

Este archivo simplemente reúne el conjunto de comandos que harían falta para ejecutar la herramienta, que consiste en:

- Arrancar base de datos.

- Preparar entorno virtual.
- Ejecutar *run.py*

De esta manera, solo hay que ejecutar un comando en vez de varios. Una vez escrito, la aplicación estará preparada y entrando en la url que se indica por la terminal, se puede acceder a la aplicación.

E.4. Manual del usuario

En este manual se van a explicar cómo se realizan las diferentes acciones que se pueden llevar a cabo dentro de la aplicación.

Consulta de noticias

Estando en la página de inicio, como se puede ver en la imagen E.1, hay que introducir la palabra clave que queramos buscar en el cuadro de texto, y pulsar en el botón “Buscar”.

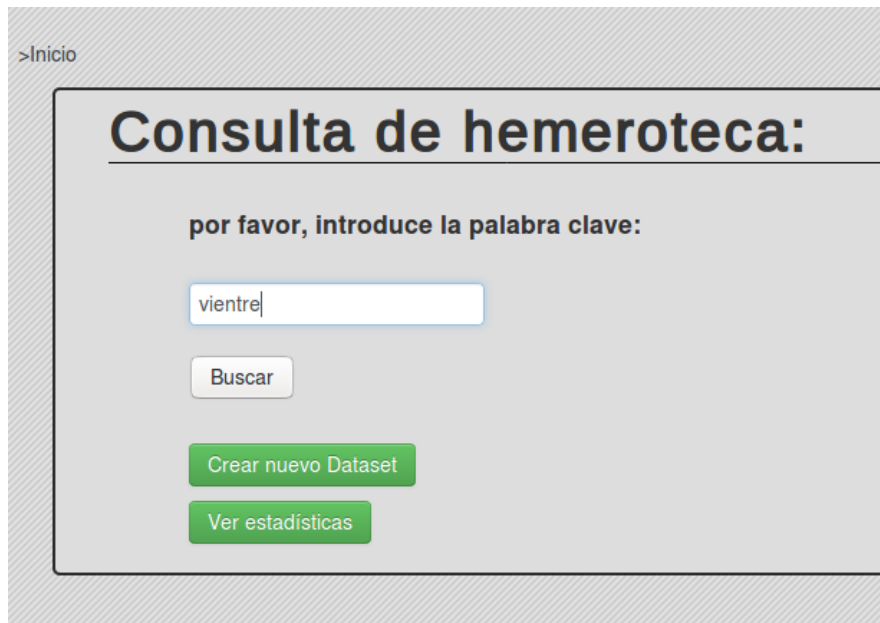
The image shows a web application interface. At the top left, there is a link labeled ">Inicio". The main heading is "Consulta de hemeroteca:". Below this, a prompt says "por favor, introduce la palabra clave:". There is a text input field containing the word "vientre". Below the input field is a button labeled "Buscar". At the bottom of the form area, there are two green buttons: "Crear nuevo Dataset" and "Ver estadísticas".

Figura E.1: Ejemplo de palabra clave de búsqueda en la herramienta

Tras un tiempo de espera, aparecerá en pantalla una tabla como la de la imagen E.2

Titular	Autor	Fecha de publicación	Fuente	Probabilidades (A favor/En contra)	Etiquetas	
Mi vientre no se alquila	Judith Bosch	25 de Marzo, 2017	DataSet Isabel	<div><div></div></div>	<input type="radio"/> A Favor <input type="radio"/> En Contra	Clasificar Etiquetar
Dones Juristes señala los intereses de lobbies	Redacción	25 de Marzo,	DataSet	<div><div></div></div>	En Contra	Clasificar

Figura E.2: Tabla de resultados

Etiquetado y Explicación

Una vez tengamos acceso a la tabla de resultados de la búsqueda, podemos realizar dos acciones con las noticias, siempre respecto al dataset seleccionado en el desplegable que hay encima de la tabla:

- Primero, podemos realizar un etiquetado manual de las noticias. Para ello, basta con pulsar el botón “Etiquetar” de la noticia para que en la celda de la etiqueta aparezcan opciones de selección única, como se ve en la imagen E.2. Cuando hayamos etiquetado las noticias que queramos, pulsamos en “Guardar Resultados” para aplicar los cambios.
- La otra opción sería ver una explicación de la predicción de la noticia, pulsando en el botón “Clasificar”, que abrirá otra pestaña en la que se puede ver una explicación de los criterios para elegir una clase como en la imagen E.3.



Figura E.3: Explicación para la predicción de la clase de una noticia

Lectura de Noticias

Si, por el contrario, lo que se quiere es aumentar el corpus de noticias en la base de datos, hay varias opciones de hacerlo.

Lectura de PDF

Pulsando en el apartado “Lectura PDF” de la barra de navegación, accedemos a la pestaña de lectura de documentos PDF. Allí, se indica al usuario en qué directorio debe dejar los archivos. La interfaz es la que se puede ver en E.4.

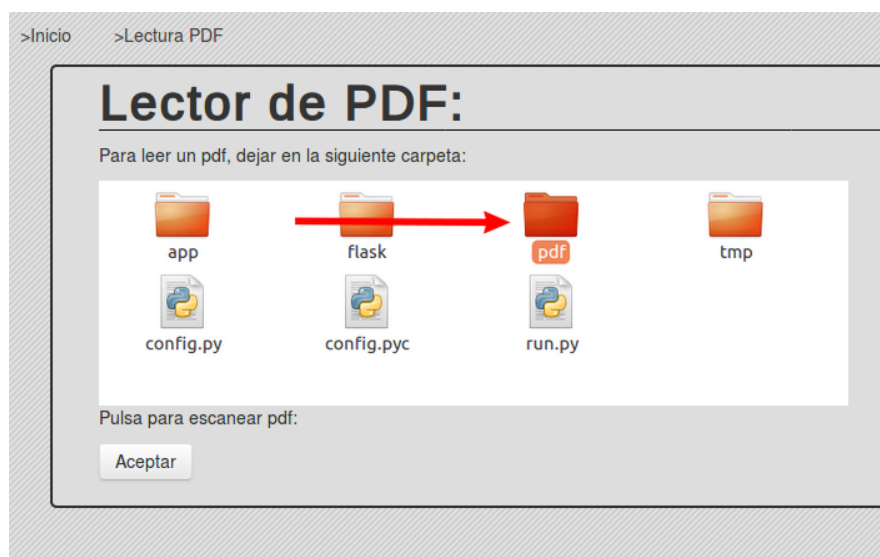


Figura E.4: Pestaña de lectura de PDF

Pulsando en el botón, nos llevará a una tabla de resultados igual que la de E.5.

Noticias leídas:			
Autor	Fecha	Nombre Archivo	Almacenada en BD
Miguel Rodríguez	25-05-2017	Nueva versión de lectura de PDF	No

Figura E.5: Resultados de la lectura de PDF

Y si allí se pulsa en “Guardar Nuevas”, se almacenarán las noticias que no estuvieran ya en la base de datos.

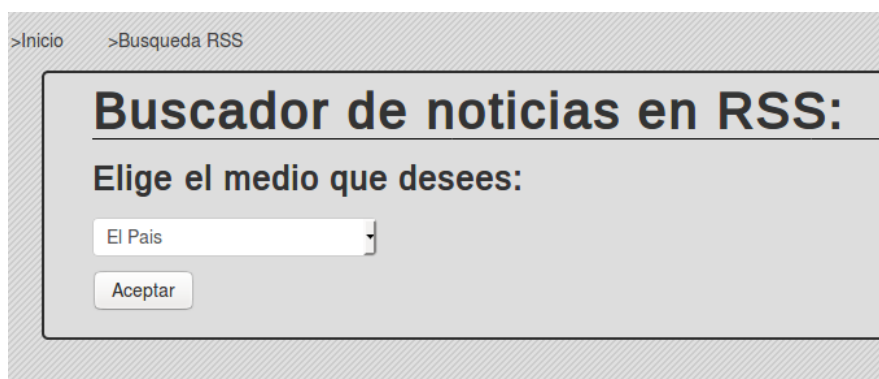
Es crucial mencionar que las noticias se deben guardar en los PDF en un formato concreto, de lo contrario no se podrá realizar la lectura de la noticia correctamente. El formato es:

- (Titular) ***titular de la noticia*** (Titular)
- (Autor) ***autor de la noticia*** (Autor)
- (Fecha) ***fecha de la noticia*** (Fecha)
- (Cuerpo) ***texto de la noticia*** (Cuerpo)

Esta decisión se tomó debido a que inicialmente había numerosos PDF con formatos distintos de los que leer noticias, y no se podía programar una solución satisfactoria para todos los casos. En la carpeta *pdf* de la aplicación, hay un documento de ejemplo con el formato adecuado.

Lectura de RSS

Pulsando en el apartado “RSS Scraping” de la barra de navegación, accedemos a una pestaña que muestra un desplegable con los nombres de varios medios de comunicación, como se puede ver en E.6. Si elegimos uno de ellos y hacemos click en “Aceptar”, nos mostrará una tabla con las últimas noticias en el RSS de ese medio de comunicación.



>Inicio >Busqueda RSS

Buscador de noticias en RSS:

Elige el medio que desees:

El Pais

Aceptar

Figura E.6: Imagen inicial de la pestaña de lectura de RSS

Tras un breve tiempo de cargar, aparecerán los resultados en una tabla como la de E.7.



Buscador de noticias en RSS:

Elige el medio que desees:

El Pais

Aceptar

Autor	Título	Fecha	Almacenada en BD
Miguel González, Pablo Guimón	España advierte que el pacto con Londres sobre el Brexit no afecta al Peñón	Mon, 26 Jun 2017 21:39:41 +0200	No
Jan Martínez Ahrens	El Supremo permite aplicar provisionalmente el núcleo del veto migratorio de Trump	Mon, 26 Jun 2017 21:45:36 +0200	No

Guardar Nuevas

Figura E.7: Resultados de lectura de RSS

Igual que con el resto de formatos, pulsando en “Guardar Nuevas” almacenaremos en la base de datos las que no estén ya guardadas.

Lectura de Dataset

Este apartado está pensado para introducir corpus de noticias enteros que el usuario pudiera tener ya preparados antes de haber usado la aplicación, por medio de un archivo XML. En esta pestaña, de forma similar a la de lectura de PDF, se indica cuál es la carpeta en la que hay que dejar los archivos, como se puede observar en E.8.

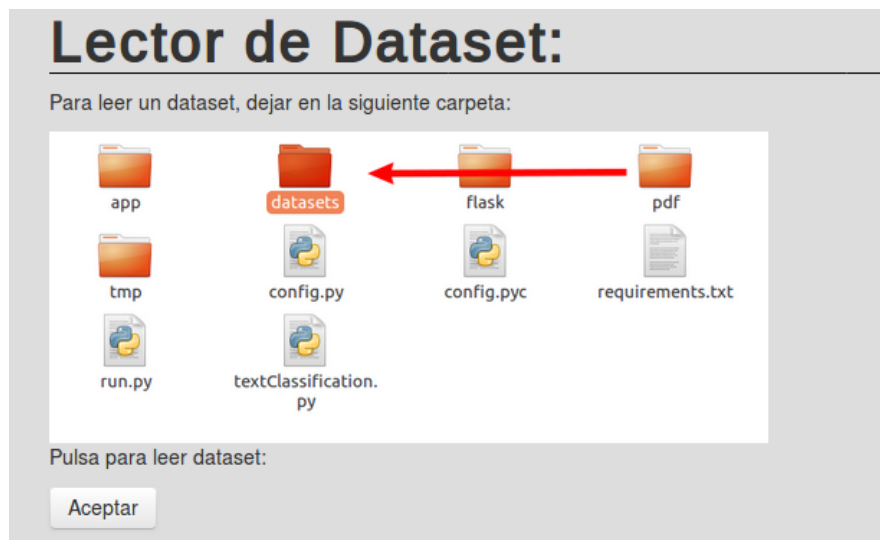


Figura E.8: Pestaña de lectura de Corpus

El formato que deben de tener los XML es el siguiente:

[illegible]

Figura E.9: Estructura de los corpus en XML

Cuando se hayan leído las noticias del corpus, se devolverá en una tabla de resultados similar al resto, pero en esta ocasión las noticias ya tendrán una etiqueta asignada. Esta tabla es como la de la imagen [E.10](#)



Autor	Fecha	Titular	etiquetas
EDUARDO MENDICUTTI	11-2-2017	La necesaria maternidad subrogada	VentreAlquiler - aFavor
Marta Martinez	9-10-2017	Mucho más que un vientre	VentreAlquiler - aFavor
J. Mouzo Quintáns, L. Rivas Martínez	1-5-2014	Las familias españolas buscan vientres de alquiler 'baratos'	VentreAlquiler - aFavor
Elisa Delos	22-2-2017	El alto precio de conseguir un donante masculino	VentreAlquiler - aFavor

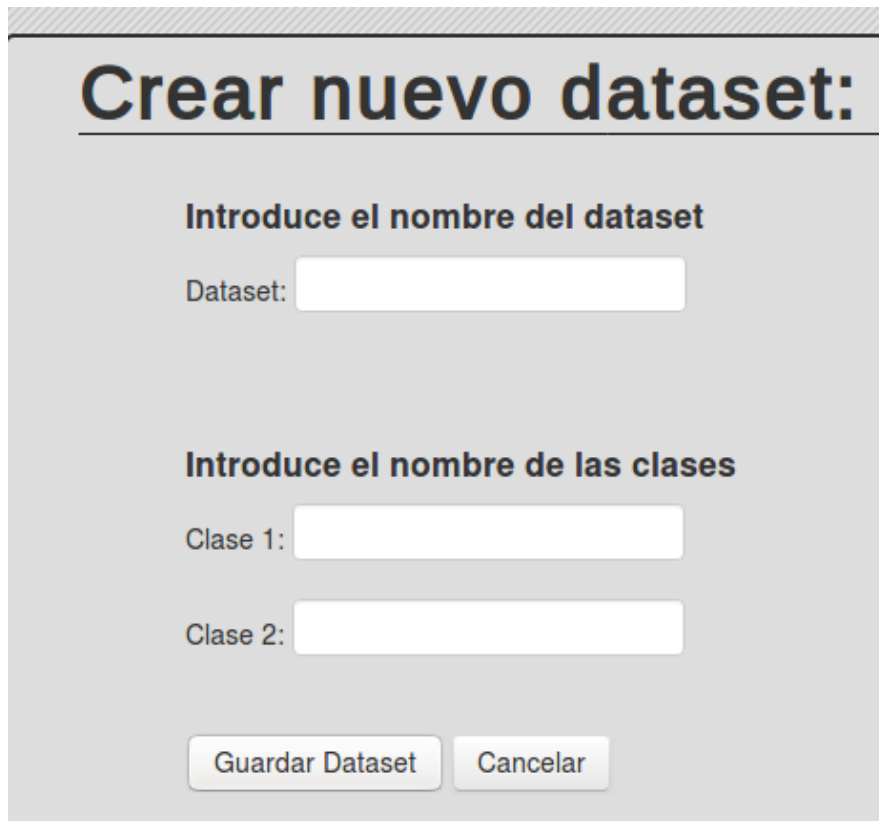
Guardar Nuevas Cancelar

Figura E.10: Tabla de resultados con el corpus

Al igual que en los otros apartados, se pueden guardar las nuevas noticias pulsando el botón correspondiente.

Creación de nuevos Conjuntos de datos

Desde la pestaña de inicio, si pulsamos el botón “Crear nuevo Dataset”, nos llevará a una nueva ventana donde podemos introducir en los cuadros de texto el nombre del conjunto y de las 2 clases que lo van a formar, y añadirle a la lista de los existentes.



Crear nuevo dataset:

Introduce el nombre del dataset

Dataset:

Introduce el nombre de las clases

Clase 1:

Clase 2:

Figura E.11: Pestaña de creación de Datasets

Una vez creado, en la tabla de resultados de noticias podrán realizarse etiquetados para ese conjunto de datos.

Estadísticas de las noticias

La herramienta permite ver, de forma básica, un conjunto de gráficos para controlar la cantidad de noticias que hay almacenadas en relación a varios factores. En la pestaña de inicio, si se pulsa “Ver estadísticas”, aparecerá una vista en la que podemos elegir si ver la cantidad de noticias en función del autor, de la fecha, o de la fuente, y mostrará el gráfico correspondiente.

No es una funcionalidad con mucha profundidad pero el usuario puede considerar interesante saber este tipo de información para añadir datos cuando realice estudios sobre las noticias. Los gráficos tienen el aspecto de la imagen

[E.12](#)

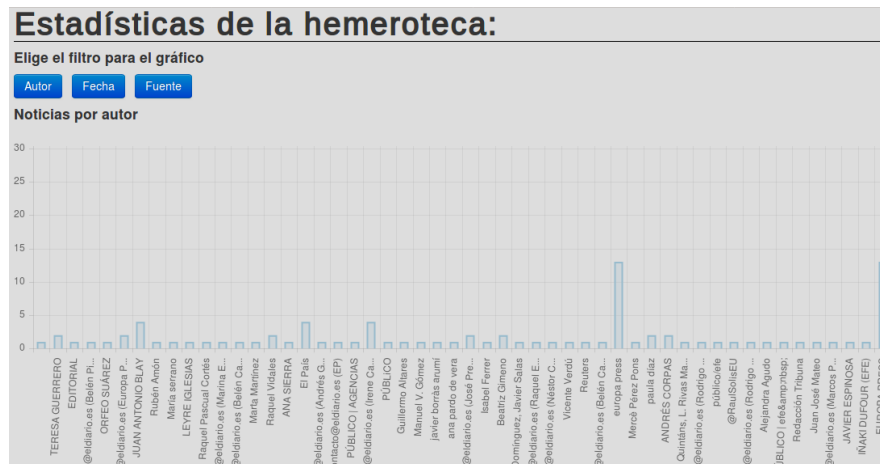


Figura E.12: Gráfico de noticias por autor

Consideraciones adicionales

Al usar la herramienta en un nuevo ordenador, hay que tener en cuenta que no habrá ni noticias ni conjuntos de datos almacenados en la base de datos.

Se podría solucionar este problema usando las opciones que da la propia herramienta, pero para facilitar la toma de contacto, hay un script llamado *createdatasets* que añade a la base de datos los dos conjuntos iniciales con los que se ha trabajado, y en la carpeta *dataset* está el corpus facilitado por nuestra colaboradora María Isabel Menéndez. La idea es que con estos dos sencillos pasos ya haya una base sobre la que seguir trabajando con las funcionalidades de la aplicación.

Bibliografía

- [1] Kristina Chodorow and Michael Dirolf. *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2010.
- [2] Miguel Grinberg. The flask mega tutorial, 2017. [Online; accessed 15-June-2017].
- [3] Mozilla. Firefox — 54.0 system requirements — mozilla, 2017. [Online; accessed 15-June-2017].



UNIVERSIDAD
DE BURGOS