Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

EVALUACIÓN DE MATERIAL DIDÁCTICO EN L2

Alumno/Alumna: Delgado, Candel, Julen
Director/Directora (1): Bengoetxea, Kortazar, Kepa Xabier

Curso: 2018-2019

Fecha: Bilbao, 15 de julio de 2019

Resumen

Los materiales didácticos utilizados para la enseñanza en niveles del último ciclo de primaria y secundaria de asignaturas curriculares que enseñan nuevos contenidos de distintas áreas (geografía, historia, ciencias, etc.) utilizando una segunda lengua (*L2*) requieren de unas características sintácticas, de vocabulario y discurso que faciliten el aprendizaje del mismo.

Por esta razón, este proyecto surge de la necesidad de evaluar la idoneidad de estos materiales. Por ello, se ha desarrollado una aplicación web (llamada *AzterTest*) que es capaz de evaluar varios textos en inglés mediante el análisis y cálculo de las distintas métricas e indicadores que sirven, especialmente, para determinar el grado de complejidad de dichos textos, de manera que esta herramienta servirá de apoyo a los profesores que necesiten determinar y clasificar textos según su complejidad.

Laburpena

Lehen eta bigarren mailakotako azken zikloan, bigarren hizkuntza (*L2*) batean irakasten diren area desberdineko eduki berrien (geografia, historia, zientziak, etab.) material didaktikoak testu hauen ikasketa errazten duten ezaugarri sintaktiko batzuk dituzte, baita hiztegiarekin eta diskurtsoarekin erlazionatuta dauden ezaugarriak ere.

Arrazoi honengatik, proiektu hau agertzen da material hauen egokitasuna ebaluatzeko beharretik. Horregatik, garatu da ingelesez idatzita dauden hainbat testu aztertzeko gai den web-aplikazioa (*AzterTest* deituta). Aplikazio honek testu baten konplexutasuna zehazteko balio duten hainbat metrika kalkulatzeko gai da. Beraz, erreminta hau behar duten irakasleentzako laguntza gisa balio izango du haien testuak bere konplexutasunaren arabera sailkatzeko.

Abstract

The teaching materials used for teaching at the last cycle of primary and secondary levels of curricular subjects that teach new contents from different areas (geography, history, sciences, etc.) using a second language (*L2*) require syntactic, vocabulary and speech characteristics that facilitate the learning of it.

For this reason, this project arises from the need to evaluate the suitability of these materials. Therefore, a web application (called *AzterTest*) has been developed. This application is capable of evaluating several texts in English through the analysis and calculation of the different metrics and indicators that serve, especially, to determine the degree of complexity of these texts, in such a way that this tool will serve as a support for teachers who need to determine and classify texts according to their complexity.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

En la actualidad, existen materiales didácticos utilizados para la enseñanza en niveles del último ciclo de primaria y secundaria de asignaturas curriculares que utilizan la metodología CLIL (*Content and Language Integrated Learning*). Mediante esta metodología, se aprenden nuevos contenidos curriculares de distintas áreas (geografía, historia, ciencias, etc.) utilizando una segunda lengua (*L2*).

Estos materiales requieren de unas características en cuanto a sintaxis, vocabulario y discurso que faciliten el aprendizaje de los contenidos. Este proyecto surge de la necesidad de evaluar la idoneidad de estos materiales, ya que es interesante analizar y presentar de manera automatizada las características de estos textos mediante métricas para poder valorar la complejidad de los textos, anticiparnos a los problemas potenciales y ofrecer ayuda en la medida de lo posible.

Por lo tanto, especialmente en el ámbito de la educación, el análisis de la complejidad del texto es una tarea muy útil, puesto que, por ejemplo, puede ayudar a los profesores a seleccionar los textos más apropiados para sus alumnos en función de su nivel educacional.

Por ello, se va a desarrollar una aplicación que evalúe textos en inglés mediante el análisis y cálculo de las distintas métricas e indicadores que sirven para medir el grado de complejidad de los textos. Además, la aplicación será capaz de clasificar los textos según su nivel de complejidad (baja, media o alta).

Por último, cabe mencionar que este proyecto surge de la propuesta de *Kepa Xabier Bengoetxea Kortazar*, profesor del Departamento de Lenguajes y Sistemas Informáticos de la Escuela de Ingeniería de Bilbao y fue escogido por diversas razones. Entre ellas, se destacan las siguientes:

- Se aprende a utilizar herramientas de procesamiento del lenguaje natural (principalmente, la librería NLTK de Python, que es una de las más utilizadas).

- Se crea una aplicación con la intención de que ésta pueda ser utilizado en un entorno real, y que profesores y profesoras de distintos centros educativos puedan hacer uso de ella.

- Se desarrolla principalmente en Python, el cual es uno de los lenguajes con mayor tasa de crecimiento en los últimos años (Rodríguez, 2019) y que, por

tanto, dispone de una comunidad de desarrolladores muy amplia, por lo que no resultará complicado aprenderla y resolver las posibles dudas que surjan durante el proceso de aprendizaje. Por estas razones, resulta de gran interés realizar el desarrollo de la aplicación en un lenguaje como este, puesto que aprender un lenguaje tan importante siempre es un gran aliciente.

- Resulta interesante el hecho de realizar una aplicación que analice textos (en este caso, en inglés) y muestre los resultados en una página web. Esto hace que cualquier usuario pueda acceder a la aplicación para poder analizar los textos que desee.

- El análisis del texto (que se realiza mediante indicadores) está basado en diversos estudios y artículos previamente identificados que son de gran utilidad.

- Se aplican los conocimientos que se han ido adquiriendo durante el grado, así como el análisis, diseño e implementación de una aplicación, y la capacidad de solucionar los problemas que puedan ir surgiendo durante el transcurso del proyecto. Por otro lado, además de aplicar las competencias obtenidas durante el grado, también se adquirirán nuevos conocimientos.

2 Planteamiento inicial

En este capítulo se exponen los objetivos del proyecto y, por tanto, se define cuál es el alcance del mismo. Además, se describen las herramientas y entornos que se utilizarán, se realiza una estimación (tanto económica como temporal) del proyecto, y se especifica cómo se gestionarán los riesgos.

2.1 OBJETIVOS

Los principales objetivos del proyecto, tanto funcionales como de aprendizaje personal, son los expuestos a continuación:

- Desarrollar una aplicación que permita la obtención de métricas sólidas y validadas referentes a la idoneidad de textos escritos en inglés con objetivos didácticos en una etapa educativa primaria o secundaria. Además, la aplicación debe proporcionar información acerca de dichas métricas, para que los usuarios que utilicen la aplicación puedan ser capaces de interpretarlas correctamente.

- La aplicación, además, hará uso de un clasificador creado en base a las métricas desarrolladas, de manera que será capaz de determinar la complejidad de un texto.

- Crear una aplicación web intuitiva, de manera que no se requieran amplios conocimientos informáticos para poder utilizar el sistema de una manera simple y eficaz.

- Adquirir experiencia con el lenguaje de programación *Python*. Pese a que se dispone de nociones básicas de este lenguaje, se pretende ampliar estos conocimientos y terminar el proyecto conociendo bien el lenguaje.

2.2 ALCANCE

Para conseguir completar de manera satisfactoria los objetivos establecidos, resulta de vital importancia definir y dividir correctamente las tareas a realizar. En este apartado, se definirá la estructura de descomposición del trabajo y, por tanto, las tareas a realizar.

2.2.1 Estructura de Descomposición del Trabajo

El trabajo se ha organizado mediante módulos o tareas que, a su vez, están compuestas por subtareas. En el diagrama correspondiente al EDT (*Estructura de Descomposición del Trabajo*), el cual se puede observar en la Ilustración 1, se pueden visualizar los paquetes agrupados por características similares, por ejemplo, todos los paquetes de formación personal del proyecto están en un mismo módulo, el módulo de "Formación y aprendizaje".

Ilustración 1. Estructura de Descomposición del Trabajo.

Cada una de las tareas (o módulos) dispondrá de una serie de subtareas (o paquetes de trabajo) más específicas que deberán ser completadas. Además, existe la

posibilidad de que un paquete de trabajo pueda ser realizado antes que otro. Es decir, en la mayoría de casos, el orden de realización es intercambiable (siempre y cuando esos paquetes de trabajo no tengan ninguna dependencia o, en caso de tenerla, ya se haya realizado).

A continuación, se va a proceder a explicar en detalle cada uno de los módulos que componen el EDT (*Estructura de Descomposición del Trabajo*):

1. **Gestión y planificación**: En este módulo, se realizan tareas que son relativas a la gestión, organización y planificación del proyecto a realizar, por lo que se definirán los objetivos y las tareas a realizar, además de hacer una selección y un estudio de las herramientas a utilizar. Cabe destacar que, en este módulo, la tarea de las reuniones con el director del proyecto se realizará de forma periódica durante el transcurso del mismo.
2. **Formación y aprendizaje**: Este módulo consistirá en el aprendizaje de todas las herramientas y lenguajes que se utilizarán durante el transcurso del proyecto. En este proyecto, se utilizarán diversas herramientas y un lenguaje de programación con el que se dispone de muy poca experiencia (*Python*), por lo que es importante dedicarle tiempo a formarse y aprender a utilizar las herramientas y los lenguajes de programación necesarios.
3. **Captura de requisitos**: Este módulo engloba la captura de requisitos; es decir, la definición de las funcionalidades, la definición de los casos de uso y el diseño de la página web.
4. **Análisis y diseño**: En este módulo se realizarán todas las tareas relacionadas con el análisis y diseño de la aplicación. En este caso, se realizará un diagrama de clases y los diagramas de secuencia necesarios, además de realizar una selección de las distintas posibles librerías que sean necesarias para poder completar el proyecto (por ejemplo, NLTK). Además, también se hará un boceto del diseño de la página web, con el objetivo de disponer de una idea general de cómo estarán distribuidas las funcionalidades.
5. **Implementación y pruebas**: En este módulo se realizará el desarrollo de la aplicación *Python* que calculará los indicadores necesarios, además del desarrollo de la página web y el clasificador que se encargará de determinar el nivel de complejidad del texto. Además, también se realizarán las pruebas y comprobaciones necesarias sobre el programa desarrollado.
6. **Documentación**: Este último módulo será muy activo durante el transcurso del proyecto, puesto que corresponde la realización de toda documentación relativa al proyecto.

Tras detallar cada uno de los módulos, se procede a entrar en detalles sobre cada uno de los paquetes de trabajo (o subtareas):

Módulo 1: Gestión y planificación

1.1.- Reuniones con el director del proyecto
Duración estimada: 20 horas
Descripción: Reunirse periódicamente con el director del proyecto para obtener detalles sobre los objetivos del proyecto, resolver dudas, etc. **Salidas/Entregables:** Avances mostrados al director proyecto y, en caso de haberlas, dudas y cuestiones relativas al desarrollo del proyecto resueltas. **Herramientas necesarias:** —
Precendencias: —

Tabla 1. Reuniones con el director del proyecto

1.2.- Definición de los objetivos
Duración estimada: 4 horas
Descripción: Definir tanto los objetivos principales del proyecto como los personales. **Salidas/Entregables:** Objetivos del proyecto. **Herramientas necesarias:** Navegador web y Microsoft Word
Precendencias: —

Tabla 2. Definición de los objetivos

1.3.- Definición de tareas a realizar
Duración estimada: 12 horas
Descripción: Definir las tareas que se realizarán a lo largo del proyecto. **Salidas/Entregables:** Estructura de Descomposición del Trabajo y listado detallado de las subtareas. **Herramientas necesarias:** Navegador web y Microsoft Word
Precendencias: Subtarea 1.2

Tabla 3. Definición de tareas a realizar

1.4.- Selección de herramientas a utilizar
Duración estimada: 6 horas
Descripción: Informarse sobre las distintas opciones que existen y seleccionar las herramientas oportunas que mejor se adecuen a las necesidades del proyecto. **Salidas/Entregables:** Listado de herramientas a utilizar. **Herramientas necesarias:** Navegador web y Microsoft Word
Precendencias: —

Tabla 4. Selección de herramientas a utilizar

Módulo 2: Formación y aprendizaje

2.1.- Aprendizaje de Python
Duración estimada: 34 horas
Descripción: Aprender un nuevo lenguaje (en este caso, *Python*) requiere recopilar información, comprenderla y aprenderla con el objetivo de entender la sintaxis y los fundamentos de dicho lenguaje. **Salidas/Entregables:** Conocimientos de *Python*. **Herramientas necesarias:** Navegador web y PyCharm IDE
Precendencias: —

Tabla 5. Aprendizaje de Python

2.2.- Aprendizaje de las herramientas a utilizar
Duración estimada: 16 horas
Descripción: Aprender a utilizar las herramientas (IDE, entornos, etc.) que se utilizarán durante el desarrollo del proyecto. **Salidas/Entregables:** Conocimientos de utilización de los entornos y herramientas a utilizar. **Herramientas necesarias:** Navegador web y herramientas
Precendencias: Subtarea 1.4

Tabla 6. Aprendizaje de las herramientas a utilizar

2.3.- Búsqueda y lectura de información
Duración estimada: 42 horas
Descripción: Buscar, leer y comprender la información que sea necesaria para cumplir los objetivos del proyecto. **Salidas/Entregables:** Información acerca de métricas de complejidad textual. **Herramientas necesarias:** Navegador web y Microsoft Word
Precendencias: Subtarea 1.2

Tabla 7. Búsqueda y lectura de información

Módulo 3: Captura de requisitos

3.1.- Definición de las funcionalidades
Duración estimada: 12 horas
Descripción: En base a los objetivos del proyecto, definir qué funcionalidad dispondrá la aplicación a desarrollar. **Salidas/Entregables:** Definición de las funcionalidades. **Herramientas necesarias:** Microsoft Word
Precendencias: Subtarea 1.2

Tabla 8. Definición de las funcionalidades

3.2.- Definición de Casos de Uso
Duración estimada: 6 horas
Descripción: Identificar los Casos de Uso y realizar el diseño del diagrama de los Casos de Uso de la aplicación. **Salidas/Entregables:** Diagrama de Casos de Uso. **Herramientas necesarias:** Visual Paradigm
Precendencias: Subtarea 3.1

Tabla 9. Definición de Casos de Uso

3.3.- Definición del Modelo de Dominio
Duración estimada: 10 horas
Descripción: Identificar las entidades y relaciones de los elementos que conforman la aplicación y, posteriormente, diseñar el diagrama del modelo de dominio de la aplicación. **Salidas/Entregables:** Diagrama de Modelo de Dominio. **Herramientas necesarias:** Visual Paradigm
Precendencias: Subtarea 3.1

Tabla 10. Definición del Modelo de Dominio

Módulo 4: Análisis y diseño

4.1.- Selección de indicadores
Duración estimada: 16 horas
Descripción: Analizar, estudiar y realizar una selección de todos aquellos indicadores que se considera que deberían ser implementados. **Salidas/Entregables:** Listado de indicadores a desarrollar. **Herramientas necesarias:** Navegador web y Microsoft Word
Precendencias: —

Tabla 11. Selección de indicadores.

4.2.- Realización de Diagrama de Clases
Duración estimada: 2 horas
Descripción: Realización de un diagrama de clases que represente cada una de las clases que conformarán la aplicación que se desarrollará en *Python*. **Salidas/Entregables:** Diagrama de Clases. **Herramientas necesarias:** Visual Paradigm
Precendencias: Subtarea 3.3

Tabla 12. Realización de Diagrama de Clases

4.3.- Realización de Diagrama de Secuencia
Duración estimada: 4 horas
Descripción: Realización de el/los diagrama(s) de secuencia para las funcionalidades o métodos más importantes. **Salidas/Entregables:** Diagrama de Secuencia. **Herramientas necesarias:** Visual Paradigm
Precendencias: Subtarea 3.3

Tabla 13. Realización de Diagrama de Secuencia

4.4.- Selección de paquetes y módulos
Duración estimada: 4 horas
Descripción: Estudiar las posibles librerías NLTK de *Python* con el objetivo de seleccionar las que mejor se adapten a las necesidades el proyecto. **Salidas/Entregables:** Listado de librerías a utilizar. **Herramientas necesarias:** Navegador web y Microsoft Word

Precendencias: Subtarea 4.2 y 4.3

Tabla 14. Selección de librerías

4.5.- Diseño interfaz de la aplicación web
Duración estimada: 2 horas
Descripción: Idear un diseño de la interfaz gráfica de la página web que cumpla con los requisitos establecidos. **Salidas/Entregables:** Boceto del diseño de la página web. **Herramientas necesarias:** Herramienta de diseño gráfico.
Precendencias: Subtarea 3.1

Tabla 15. Diseño interfaz de la aplicación web

Módulo 5: Implementación y pruebas

5.1.- Indicadores generales
Duración estimada: 10 horas
Descripción: Implementar los indicadores generales que habrán sido analizados y seleccionados en el proceso de análisis. **Salidas/Entregables:** Código Python. **Herramientas necesarias:** PyCharm IDE y librerías NLTK
Precendencias: Subtarea 4.4

Tabla 16. Desarrollo de los indicadores generales

5.2.- Indicadores de riqueza léxica
Duración estimada: 20 horas
Descripción: Implementar los indicadores de riqueza léxica que habrán sido analizados y seleccionados en el proceso de análisis. **Salidas/Entregables:** Código Python. **Herramientas necesarias:** PyCharm IDE y librerías NLTK
Precendencias: Subtarea 5.1

Tabla 17. Desarrollo de los indicadores de riqueza léxica

5.3.- Indicadores de lecturabilidad
Duración estimada: 20 horas
Descripción: Implementar los indicadores de lecturabilidad que habrán sido analizados y seleccionados en el proceso de análisis. **Salidas/Entregables:** Código Python. **Herramientas necesarias:** PyCharm IDE y librerías NLTK

Precendencias: Subtarea 5.1

Tabla 18. Desarrollo de los indicadores de lecturabilidad

5.4.- Indicadores morfológicos
Duración estimada: 10 horas
Descripción: Implementar los indicadores morfológicos que habrán sido analizados y seleccionados en el proceso de análisis. **Salidas/Entregables:** Código Python. **Herramientas necesarias:** PyCharm IDE y librerías NLTK
Precendencias: Subtarea 5.1

Tabla 19. Desarrollo de los indicadores morfológicos

5.5.- Indicadores de frecuencia de palabra
Duración estimada: 5 horas
Descripción: Implementar los indicadores de frecuencia de palabra (*Word Frequency*) que habrán sido analizados y seleccionados en el proceso de análisis **Salidas/Entregables:** Código Python. **Herramientas necesarias:** PyCharm IDE y librerías NLTK
Precendencias: Subtarea 5.1

Tabla 20. Desarrollo de los indicadores de frecuencia de palabra

5.6.- Indicadores de conocimiento del vocabulario
Duración estimada: 5 horas
Descripción: Implementar, utilizando el diccionario de *Oxford* de palabras clasificadas según el nivel de complejidad, los indicadores del conocimiento del vocabulario, que habrán sido analizados y seleccionados en el proceso de análisis. **Salidas/Entregables:** Código Python. **Herramientas necesarias:** PyCharm IDE y librerías NLTK
Precendencias: Subtarea 5.1

Tabla 21. Desarrollo de los indicadores de conocimiento del vocabulario

5.7.- Indicadores sintácticos
Duración estimada: 10 horas
Descripción: Implementar los indicadores sintácticos que habrán sido analizados y seleccionados en el proceso de análisis. **Salidas/Entregables:** Código Python. **Herramientas necesarias:** PyCharm IDE y librerías NLTK

Precendencias: Subtarea 5.1

Tabla 22. Desarrollo de los indicadores sintácticos

5.8.- Indicadores de cohesión
Duración estimada: 20 horas
Descripción: Implementar los indicadores de cohesión que habrán sido analizados y seleccionados en el proceso de análisis. **Salidas/Entregables:** Código Python. **Herramientas necesarias:** PyCharm IDE y librerías NLTK
Precendencias: Subtarea 5.1

Tabla 23. Desarrollo de los indicadores de cohesión

5.9.- Clasificador
Duración estimada: 15 horas
Descripción: Implementar, en *Python*, un clasificador que sea capaz de determinar la dificultad de un texto en base a los indicadores previamente calculados. **Salidas/Entregables:** Código Python. **Herramientas necesarias:** PyCharm IDE y librerías de Machine Learning.
Precendencias: Subtareas 5.1-5.8

Tabla 24. Desarrollo del clasificador

5.10.- Desarrollo de la aplicación web
Duración estimada: 35 horas
Descripción: Realizar el desarrollo de la página web en base a un boceto de la apariencia del mismo. Esta tarea incluye la configuración del servidor web. **Salidas/Entregables:** Código HTML y PHP. **Herramientas necesarias:** Navegador web, Apache Web Server y Atom.
Precendencias: —

Tabla 25. Desarrollo de la aplicación web

5.11.- Realización de pruebas
Duración estimada: 20 horas
Descripción: Realizar pruebas de manera periódica sobre el código desarrollado y asegurarse de que todas las métricas cumplan con lo establecido. **Salidas/Entregables:** Código depurado. **Herramientas necesarias:** Navegador web y PyCharm IDE
Precendencias: —

Tabla 26. Realización de pruebas

Módulo 6: Documentación

6.1.- Redacción del DOP
Duración estimada: 25 horas
Descripción: Realizar el Documento de Objetivos del Proyecto (DOP), que incluya la definición de los objetivos, alcance, planificación temporal, herramientas, gestión de riesgos y evaluación económica. **Salidas/Entregables:** Documento de Objetivos del Proyecto (DOP). **Herramientas necesarias:** Microsoft Word
Precendencias: Subtarea 1.2

Tabla 27. Redacción del DOP

6.2.- Redacción de la memoria
Duración estimada: 85 horas
Descripción: Realizar la memoria del TFG durante el desarrollo del mismo, que incluya el planteamiento inicial, antecedentes, captura de requisitos, análisis, diseño, desarrollo, pruebas, conclusiones y experiencia personal. **Salidas/Entregables:** Memoria del TFG. **Herramientas necesarias:** Microsoft Word
Precendencias: —

Tabla 28. Redacción de la memoria

A continuación, en la Tabla 29, tras haber definido las tareas de cada uno de los módulos, se procede a listar cada una de ellas en forma de tabla, de manera que servirá a modo de resumen, y se podrá obtener una visión más general de las tareas a realizar, incluyendo el cómputo total de horas estimadas.

Módulo	ID	Tarea	Duración estimada	Duración total
Gestión y planificación	1.1	Reuniones con el director del proyecto	20 h	42 h
	1.2	Definición de los objetivos	4 h	
	1.3	Definición de tareas a realizar	12 h	
	1.4	Selección de herramientas a utilizar	6 h	
Formación y aprendizaje	2.1	Aprendizaje de Python	34 h	92 h
	2.2	Aprendizaje de las herramientas a utilizar	16 h	
	2.3	Búsqueda y lectura de información	42 h	
Captura de requisitos	3.1	Definición de las funcionalidades	12 h	28 h
	3.2	Definición de Casos de Uso	6 h	
	3.3	Definición del Modelo de Dominio	10 h	
Análisis y diseño	4.1	Selección de indicadores	16 h	28 h
	4.2	Realización de Diagrama de Clases	2 h	
	4.3	Realización de Diagrama de Secuencia	4 h	
	4.4	Selección de paquetes y módulos	4 h	
	4.5	Diseño interfaz de la aplicación web	2 h	
Implementación y pruebas	5.1	Indicadores generales	10 h	170 h
	5.2	Indicadores de riqueza léxica	20 h	
	5.3	Indicadores de lecturabilidad	20 h	
	5.4	Indicadores morfológicos	10 h	
	5.5	Indicadores de frecuencia de palabra	5 h	
	5.6	Indicadores de conocimiento del vocabulario	5 h	
	5.7	Indicadores sintácticos	10 h	
	5.8	Indicadores de cohesión	20 h	
	5.9	Clasificador	15 h	
	5.10	Desarrollo de la aplicación web	35 h	
	5.11	Realización de pruebas	20 h	
Documentación	6.1	Redacción del DOP	25 h	110 h
	6.2	Redacción de la memoria	85 h	
				470 h

Tabla 29. Tareas del proyecto

2.3 PLANIFICACIÓN TEMPORAL

Una vez se han definido y detallado cada una de las tareas que se han de completar, se debe especificar cómo se distribuirán a lo largo del tiempo. De esta manera, se puede estimar el momento en el que el proyecto estará finalizado. Para ello, se ha realizado un diagrama de Gantt que puede verse en la Ilustración 2, donde se ilustra la planificación temporal que se desea seguir a lo largo del proyecto.

Se ha establecido una jornada laboral de unas 4 horas al día, de lunes a viernes, de modo que se trabajarán 20 horas semanales. Los días festivos (incluidos fines de semana) se descansará, siempre y cuando se haya cumplido con las horas semanales.

Además, debido a que el mes de mayo es época de exámenes, se hará un parón de, aproximadamente, dos semanas para poder dedicar más tiempo al estudio y preparación de dichos exámenes, aunque se seguirá acudiendo a las reuniones con el director del proyecto. Tras el parón, se reanudará el desarrollo del proyecto.

Cabe destacar que tanto la realización del DOP como la redacción de la memoria son tareas que se realizan a lo largo del transcurso del proyecto, por lo que, en el diagrama de Gantt, con la finalidad de simplificar la representación, se muestran como bloques que duran desde el comienzo del proyecto y se extienden hasta la finalización del mismo. Es decir, con esto se pretende representar que la redacción de toda la documentación se ejecutará en paralelo al desarrollo del proyecto.

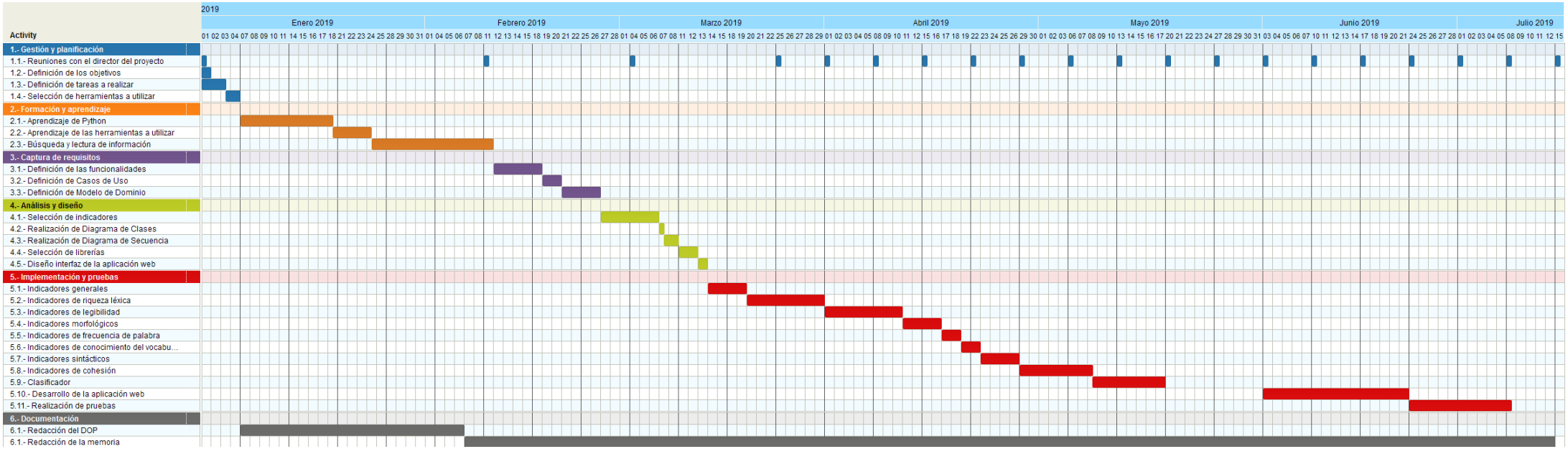


Ilustración 2. Diagrama de Gantt.

2.4 HERRAMIENTAS

En este apartado se exponen todas las herramientas que se utilizarán durante el transcurso del proyecto, explicando para qué sirven y su razón de uso.

- **Windows 10:** Principalmente, se utilizará este sistema operativo desarrollado por Microsoft para redactar la documentación y diseñar los diagramas correspondientes, puesto que se pretende utilizar herramientas que solo están disponibles en este sistema operativo.

- **Ubuntu 18.04:** Para desarrollar tanto la aplicación *Python* como la página web, se trabajará en esta distribución de Linux de código abierto, ya que es más cómodo trabajar con la terminal y, además, la máquina virtual del servidor trabaja con este sistema operativo, por lo que será más sencillo realizar transferencias de archivos del equipo local al servidor remoto.

- **Microsoft Word**: Puesto que es una herramienta fácil de utilizar y se dispone de experiencia con ella, se utilizará este procesador de texto para redactar la documentación del proyecto.

- **Tom's Planner**: Esta aplicación online permite la creación y mantenimiento de diagramas de Gantt de manera sencilla. Se utilizará, por lo tanto, para realizar la planificación temporal del proyecto.

- **Dropbox**: Esta herramienta permite almacenar y compartir archivos, carpetas y documentos en la nube entre distintos usuarios y de manera sencilla. Por esa razón, se utilizará esta aplicación para guardar toda copia de seguridad necesaria, tal y como se especifica en el apartado de la gestión de riesgos.

- **Google Drive**: Además de Dropbox, también se utilizará esta herramienta de almacenamiento de archivos en la nube para guardar todas las copias de seguridad que sean necesarias.

- **Apache Web Server**: Se trata de uno de los servidores web más utilizados. Se ha escogido trabajar con Apache porque durante el grado se ha trabajado con él y, por tanto, se dispone de experiencia.

- **Visual Paradigm**: Se trata de una herramienta que permite realizar diagramas UML, ya sean diagramas de modelo de dominio o diagramas de casos de uso.

Por esa razón, este programa se utilizará para realizar los diagramas UML que sean necesarios.

- **Atom**: Esta herramienta es un editor de código de fuente de código abierto que soporta lenguajes como HTML, PHP o JavaScript. Por tanto, este editor de código se utilizará en el desarrollo de la página web, puesto que habrá que utilizar lenguajes como los anteriormente mencionados, y esta herramienta facilitará el desarrollo de la misma.

- **PyCharm IDE**: Se trata de un IDE (entorno de desarrollo integrado) que está diseñado para desarrollar código *Python* y que dispone de múltiples funcionalidades que son de gran utilidad, así como una terminal integrada y la posibilidad de integrar un entorno virtual. Por esa razón, se utilizará este IDE para desarrollar el código *Python*.

- **Jupyter Notebook**: Esta herramienta se trata de entorno de trabajo interactivo que permite desarrollar código en *Python* de manera dinámica, por lo que será útil a la hora de probar *snippets* de código de manera sencilla. Por lo tanto, se utilizará este entorno como apoyo durante el desarrollo, logrando así mayor eficacia a la hora de programar, puesto que se podrán hacer las pruebas en un entorno separado.

- **GitKraken:** Puesto que el código desarrollado se alojará en GitHub, se utilizará esta herramienta de para realizar el control de versiones del código sobre el repositorio creado. De esta manera, se asegura que no se pierda el código, puesto que éste quedará alojado en dicho repositorio.

- **Mozilla Firefox:** Este será el navegador web principal que será utilizado para la búsqueda de información y recursos necesarios para el cumplimiento de los objetivos del proyecto. Además, se utilizará para testear la aplicación web.

2.5 GESTIÓN DE RIESGOS

En todo proyecto pueden surgir imprevistos y contratiempos que pueden hacer que el desarrollo de éste se vea afectado. Por esa razón, en este apartado se enumeran una serie de riesgos tenidos en cuenta, con el objetivo de minimizar el impacto que estos contratiempos puedan ocasionar sobre el proyecto.

La probabilidad de que suceda algún imprevisto que llegue a afectar al proyecto es variable y difícil de medir. Por ello, se utilizará una tabla orientativa que servirá de apoyo a la hora de identificar los riesgos y definir la probabilidad de que sucedan (Tabla 30).

Probabilidad	Porcentaje
Muy baja	0% – 25%
Baja	25% – 45%
Media	45% – 55%
Alta	55% – 75%
Muy alta	75% – 100%

Tabla 30. Probabilidad de los riesgos

Por otro lado, además de analizar y definir la probabilidad de que una amenaza llegue a producirse, también hay que tener en cuenta qué impacto tendría sobre el proyecto. Por esa razón, se ha realizado una tabla orientativa para estimar cuánto tiempo se retrasaría el proyecto en caso de que el riesgo llegara a producirse (Tabla 31).

Impacto	Retraso
Bajo	Menos de un día
Moderado	1 – 3 días
Alto	Más de 3 días

Tabla 31. Impacto de los riesgos

A continuación, se procede a describir y analizar cada uno de los posibles riesgos. Se definirá, además de la probabilidad y el impacto (usando como base las dos tablas previamente definidas), los planes de prevención y contingencia.

2.5.1 Pérdida de la documentación

Los diagramas y la documentación del proyecto se alojarán en un ordenador de sobremesa, pero puede ocurrir que el disco duro de este ordenador deje de funcionar y se pierda todo el trabajo realizado.

Plan de prevención

Guardar copias de seguridad en al menos dos servicios de alojamiento de archivos en la nube (tales como Dropbox y Google Drive) y en dispositivos de almacenamiento físicos (por ejemplo, un disco duro externo). Realizar dichas copias de seguridad cada día.

Plan de contingencia

Restablecer la última copia de seguridad creada.

Probabilidad
Muy baja.

Impacto
Bajo.

2.5.2 Pérdida del código

El código se alojará en un repositorio de GitHub, pero puede ocurrir que un día este servicio deje de funcionar y se pierda todo el trabajo realizado.

Plan de prevención
Guardar copias de seguridad en al menos dos servicios de alojamiento de archivos en la nube (tales como Dropbox y Google Drive) y en dispositivos de almacenamiento físicos (por ejemplo, un disco duro externo). Realizar dichas copias de seguridad cada día.

Plan de contingencia
Restablecer la última copia de seguridad creada.

Probabilidad
Muy baja.

Impacto
Bajo.

2.5.3 Avería del equipo informático

Debido al transporte, utilización u otras causas, es posible que ocurra que el equipo informático con el que se trabaja deje de funcionar o se averíe.

Plan de prevención
Mantener el equipo informático en unas condiciones de utilización adecuadas, sin exponerlo a posibles caídas ni a un ambiente de trabajo con polvo y/o humedad.

Plan de contingencia
En caso de que el fallo se encuentre en un componente hardware, comprar y reemplazar dicho componente con la mayor brevedad posible.

Si se trata de una avería general del equipo (y que no se pueda arreglar reemplazando componentes del ordenador), utilizar un equipo secundario o trabajar en los ordenadores que se encuentran en la Universidad.

Probabilidad

Baja.

Impacto

Alto.

2.5.4 Fallo o pérdida de la conexión a Internet

La conexión a Internet puede verse interrumpida, lo cual puede impedir que se puedan realizar ciertas actividades, tales como la búsqueda de información y/o consulta de documentación relacionada con el desarrollo del proyecto.

Plan de prevención

Contratar un servicio de conexión a Internet confiable y que disponga de una velocidad aceptable. Además, conviene disponer siempre de un punto de acceso a Internet alternativo (por ejemplo, se puede utilizar el teléfono móvil como punto de conexión a Internet).

Plan de contingencia

Contactar con el proveedor de servicios de Internet correspondiente, con el fin de disponer de una conexión a Internet lo antes posible.

Hasta que se restablezca la conexión a Internet, se puede utilizar un punto de acceso a Internet alternativo, o incluso trabajar en los ordenadores de la Universidad.

Probabilidad

Baja.

Impacto

Moderado.

2.5.5 Fallo o recursos insuficientes en el servidor

El servidor que aloja la aplicación web puede fallar en cualquier momento o dejar de estar operativo. También cabe la posibilidad de que el servidor no sea

capaz de procesar grandes volúmenes de texto (por ejemplo, porque no tiene suficiente memoria RAM).

Plan de prevención

Realizar copias de seguridad de los ficheros que se almacenen en el servidor tanto en Dropbox como en Google Drive. Además, conviene contratar un servidor con lo suficientemente potente (con un mínimo de memoria RAM, capacidad de procesamiento y espacio de disco).

Plan de contingencia

En caso de que el servidor deje de funcionar, contratar otro servicio que ofrezca servidores más fiables y restaurar las copias de seguridad en el nuevo servidor.

Si el problema está en que el servidor no es lo suficientemente potente, contratar otro con más potencia (dentro de la medida de lo posible).

Probabilidad

Media.

Impacto

Moderado.

2.5.6 Lesión o enfermedad

Puede ocurrir que no sea posible trabajar en el proyecto debido a una lesión y/o enfermedad. Esto puede ocurrir debido al exceso o malas costumbres a la hora de trabajar en el proyecto, o bien por causas ajenas.

Plan de prevención

Sentarse correctamente y manteniendo una distancia correcta frente al monitor puede impedir lesiones de espalda. Además, para evitar una lesión de muñeca, es conveniente descansar la mano cada cierto tiempo.

Para evitar enfermedades, es muy importante mantener una correcta alimentación y un buen hábito de sueño.

Plan de contingencia

En caso de que se trate de una enfermedad/lesión que esté impidiendo poder realizar el trabajo, acudir al médico con la mayor brevedad posible y reanudar el trabajo una vez se haya completado la recuperación. Si aún es posible realizar el trabajo aun estando enfermo o lesionado, descansar con mayor frecuencia y no esforzarse en exceso para evitar que la enfermedad/lesión vaya a más.

Probabilidad
Baja.

Impacto
Alto.

2.6 EVALUACIÓN ECONÓMICA

En este apartado, el objetivo es detallar y calcular los diferentes costes asociados a la realización del proyecto, lo cual podría servir en el supuesto de que la aplicación llegara a comercializarse, pese a que ese no es el caso del presente proyecto.

Tras realizar la planificación temporal, conocemos una estimación de la duración del proyecto, por lo que podemos realizar una evaluación económica del mismo. Se tendrán en cuenta diversos aspectos que produzcan costes de proyecto, tales como la utilización del equipo informático necesario o la compra de licencias de software.

Coste de la mano de obra

El trabajo realizado puede clasificarse dentro de la categoría de *Analista programador* y *Diseñador página web*, por lo que para calcular el coste de mano de obra se ha empleado el sueldo mínimo de esta categoría, que viene establecido en el *Convenio colectivo estatal de empresas de consultoría, y estudios de mercados y de la opinión pública.* Según este documento, el sueldo mínimo para este tipo de empleo, a partir del 1 de enero de 2019, es de 20.896,44 € anuales (Ministerio de Empleo y Seguridad Social, 2018). Por tanto, el salario mínimo mensual es de 1.741,37 € pero, dado que se trata de un sueldo correspondiente a un contrato a jornada completa, hay que calcular el sueldo por horas para poder obtener el coste total de la mano de obra.

Asumiendo que la jornada completa corresponde a 8 horas diarias, y teniendo en cuenta que en un mes hay 20 días laborables, se obtiene el salario por horas:

$$Salario\ a\ la\ hora\ = \frac{Salario\ mensual}{Horas\ totales\ al\ mes} = \frac{1741{,}37\ €}{8\ horas\ al\ día \cdot 20\ días} =\ 10{,}88\ {}^{€}/_{hora}$$

Con este dato, y sabiendo que se estima trabajar un total de 470 horas en este proyecto, se obtiene el coste total de la mano de obra:

$$Coste\ de\ la\ mano\ de\ obra\ = 470\ horas\ \cdot\ 10{,}88\ {}^{€}/_{hora} =\ 5.113{,}60\ €$$

En definitiva, el coste de la mano de obra del proyecto asciende a 5113,60 €.

Coste del equipo informático

Se dispone de un ordenador portátil de la marca *Lenovo* (modelo V110-15ISK) valorado en 335 €. Se asume que dicho equipo dispone de una vida útil estimada de 4 años (es decir, 48 meses).

Para calcular el gasto económico que supone este ordenador portátil, se realiza en base a su amortización mensual, cuya fórmula es la siguiente:

$$Amortización\ mensual\ = \frac{Precio}{Vida\ útil\ estimada} = \frac{335\ €}{48\ meses} = \quad 6{,}97\ €$$

Con el supuesto de que la vida útil del ordenador portátil es de 48 meses, se obtiene una amortización mensual de 6,97 €. En base a este dato, se puede proceder a calcular la amortización total, teniendo en cuenta que el proyecto tendrá una duración aproximada de 6 meses:

$$Amortización\ total\ =\ 6{,}97\ € \cdot 6\ meses\ =\ \ 41{,}82\ €$$

Por otro lado, también se dispone de un equipo de sobremesa valorado en 750 €. Se asumirá que este ordenador dispone de una vida útil estimada de 6 años o, lo que es lo mismo, 72 meses. Con estos datos, se procede a calcular la amortización total del equipo de sobremesa de la misma manera que se ha hecho con la del ordenador portátil.

$$Amortización\ mensual\ = \frac{Precio}{Vida\ útil\ estimada} = \frac{750\ €}{72\ meses} = \quad 10{,}42\ €$$

$$Amortización\ total\ =\ 10{,}42\ € \cdot 6\ meses\ =\ \ 62{,}52\ €$$

En definitiva, se obtiene que el gasto derivado de la amortización del ordenador portátil es de 41,82 € y el del equipo de sobremesa de 62,52 €, lo que hace un total de 104,34 €.

Coste del servidor

Se utilizará el servicio de *hosting* que ofrece **Digital Ocean** para albergar la aplicación web del proyecto. En concreto, se ha escogido un servidor que dispone de

3 GB de memoria RAM, una vCPU y 60 GB de almacenamiento SSD, el cual tiene un precio de 15 $ al mes. [1]

Puesto que (en el momento de redactar esta memoria) un dólar estadounidense equivale a 0,8937 € [2], se realiza la correspondiente conversión:

$$Coste\ al\ mes = 15\ \$ \cdot 0{,}8937\ {}^{€}/_{\$} = 13{,}40\ €$$

Una vez realizada esta conversión, se procede a calcular el coste total del servidor teniendo en cuenta, de nuevo, la duración estimada del proyecto:

$$Coste\ total\ del\ servidor = 13{,}40\ € \cdot 6\ meses = 80{,}40\ €$$

Cabe destacar que el coste del servidor no es pagado por el autor del proyecto, sino por la Universidad el País Vasco (UPV).

Coste del software

En este proyecto, se utilizan algunas herramientas que tienen costes de licencia de software. En concreto, el software que requiere la adquisición de licencias para ser utilizado es:

- Microsoft Windows 10
- Microsoft Office
- Visual Paradigm
- PyCharm IDE

Sin embargo, gracias al acuerdo que dispone la Universidad del País Vasco con Microsoft (denominado Microsoft Imagine UPV/EHU), el gasto realizado para la licencia de Microsoft Windows 10 es de 0 €.

Además, JetBrains ofrece licencias de estudiante para varios de sus productos, entre ellos PyCharm, por lo que el gasto para esta licencia también corresponde a 0 €. Lo mismo ocurre con Visual Paradigm, de la cual se dispone de una licencia de estudiante.

En definitiva, el único gasto en software corresponde a la licencia de Microsoft Office. La licencia de este paquete de herramientas de Microsoft corresponde a 69,00 €

[1] Según la página oficial de Digital Ocean: https://www.digitalocean.com/pricing/#Compute
[2] Según el elEconomista.com: https://www.eleconomista.es/cruce/EURUSD

anuales[3], por lo que la amortización de este producto para este proyecto (que se estima que dure 6 meses) sería de 34,50 €. Por lo tanto, el gasto del software corresponde a 34,50 €.

Costes indirectos

Los gastos indirectos son aquellos que no tienen una relación con el desarrollo del proyecto, pero que hay que tener en cuenta a la hora de estimar el coste económico del mismo (por ejemplo, los costes de la luz, la tarifa de Internet, etc.). Para este proyecto, se considerará que dichos gastos suponen el 5 % del coste total del proyecto.

Como la suma de los costes anteriores (mano de obra, equipo informático, servidor y software) asciende a 5.332,84 €, el 5 % de ese valor y, por tanto, los costes indirectos del proyecto son de 266,64 €.

Coste total del proyecto

Finalmente, se realiza la suma de todos los costes con el fin de obtener el coste total del desarrollo del proyecto:

Descripción	Coste
Coste de la mano de obra	5113,60 €
Coste del equipo informático	104,34 €
Coste del servidor	80,40 €
Coste del software	34,50 €
Costes indirectos	266,64 €
Total	5599,48 €

Tabla 32. Coste total del proyecto

[3] Según el listado de precios de la página oficial de Microsoft: https://products.office.com/es-es/buy/office

3 Antecedentes

En este apartado, se expondrá una visión general correspondiente a la situación actual; es decir, todas aquellas herramientas y aplicaciones ya existentes que cuyas funcionalidades sean similares a las de la aplicación a desarrollar.

Actualmente existen múltiples herramientas que están orientadas al análisis y procesamiento de texto y que resulta interesante que sean estudiadas, puesto que pueden servir de inspiración para enfocar el desarrollo de muchas de las funcionalidades del sistema.

3.1 COH-METRIX

Coh-Metrix es una herramienta computacional (ver Ilustración 3) desarrollada por Arthur C. Graesser, Danielle S. McNamara, Max M. Louwerse y Zhiqiang Cai que calcula la coherencia de los textos en base a diversos índices o indicadores. Estos indicadores se clasifican en once grupos (McNamara, Graesser, McCarthy, & Cai, 2014):

1) *Descriptives:* se analizan patrones en el texto como el número de párrafos, palabras o sílabas por palabra.
2) *Text easability*: se analizan características lingüísticas de los textos, tales como la temporalidad, la narrativa y la conectividad.
3) *Cohesion referencial*: se analiza la cantidad de relaciones de cohesión que un ser humano lector podría realizar en base a las proposiciones y frases del texto.
4) *Latent semantic analysis*: se analiza la similitud entre frases y párrafos.
5) *Lexical diversity*: se analizan las relaciones de tipo-*token* con el objetivo de determinar la cohesión del texto.
6) *Conectores*: se analiza la cantidad de conectores en el texto.
7) *Situation model*: se analizan y calculan índices que están relacionado con la representación mental que el lector tiene sobre el texto.
8) *Syntactic complexity*: se analiza las frases y se calcula la densidad de palabras en el texto.
9) *Word Information*: se analizan las palabras del texto para obtener información acerca de la densidad de cada tipo de palabra.
10) *Readability*: se analiza y se calcula la lecturabilidad del texto mediante fórmulas como *Flesch-Kincaid Grade*.

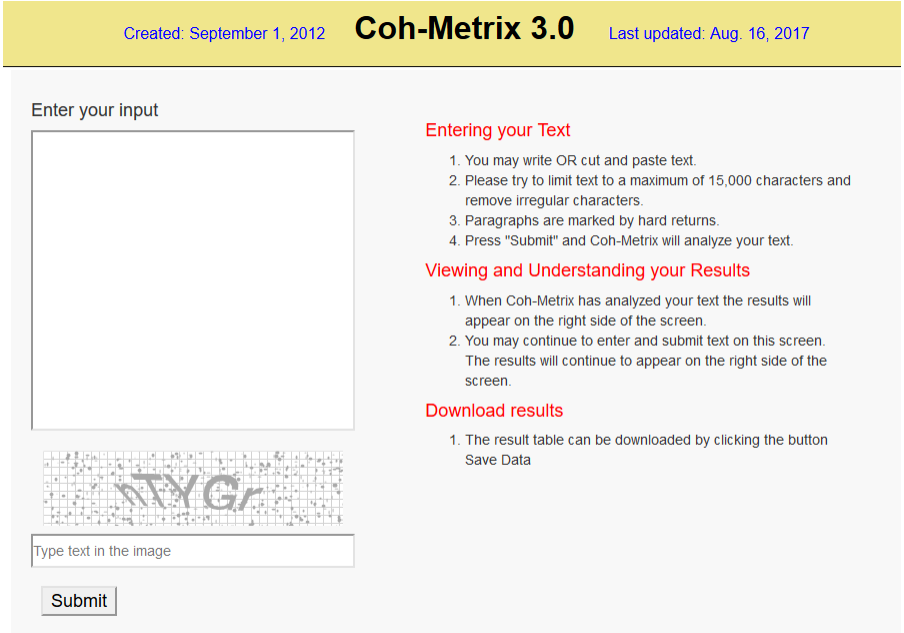
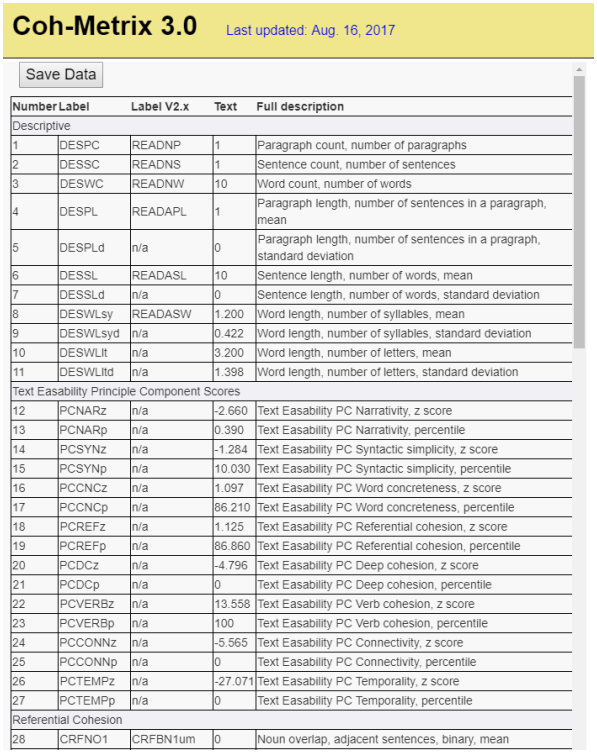


Ilustración 3. Página principal de Coh-Metrix

Coh-Metrix analiza texto sobre más de 50 tipos de relaciones de cohesión y más de 200 medidas de lenguaje, texto y lecturabilidad[4]. Existen módulos que utilizan léxicos, categorizadores, analizadores sintácticos, corpus, y otros componentes que se utilizan ampliamente en la lingüística computacional. *Coh-Metrix* consigue integrar todos estos módulos en una sola aplicación web (Graesser, McNamara, Louwerse, & Cai, 2004).

Tras analizar un texto, los datos son representados en una tabla, especificando la métrica calculada junto con su descripción. Además, se ofrece la posibilidad de guardar estos resultados mediante un botón (denominado *Save Data*) (ver Ilustración 4).

[4] La *lecturabilidad* es la facilidad con la que el lector puede entender un texto, al margen del medio utilizado para emitir el mensaje. Es decir, se refiere a aquellos aspectos gramaticales, sintácticos y léxicos que ayudan a la comprensión de los contenidos del texto.



Coh-Metrix 3.0 Last updated: Aug. 16, 2017

Save Data

Number	Label	Label V2.x	Text	Full description
Descriptive				
1	DESPC	READNP	1	Paragraph count, number of paragraphs
2	DESSC	READNS	1	Sentence count, number of sentences
3	DESWC	READNW	10	Word count, number of words
4	DESPL	READAPL	1	Paragraph length, number of sentences in a paragraph, mean
5	DESPLd	n/a	0	Paragraph length, number of sentences in a pragraph, standard deviation
6	DESSL	READASL	10	Sentence length, number of words, mean
7	DESSLd	n/a	0	Sentence length, number of words, standard deviation
8	DESWLsy	READASW	1.200	Word length, number of syllables, mean
9	DESWLsyd	n/a	0.422	Word length, number of syllables, standard deviation
10	DESWLlt	n/a	3.200	Word length, number of letters, mean
11	DESWLltd	n/a	1.398	Word length, number of letters, standard deviation
Text Easability Principle Component Scores				
12	PCNARz	n/a	-2.660	Text Easability PC Narrativity, z score
13	PCNARp	n/a	0.390	Text Easability PC Narrativity, percentile
14	PCSYNz	n/a	-1.284	Text Easability PC Syntactic simplicity, z score
15	PCSYNp	n/a	10.030	Text Easability PC Syntactic simplicity, percentile
16	PCCNCz	n/a	1.097	Text Easability PC Word concreteness, z score
17	PCCNCp	n/a	86.210	Text Easability PC Word concreteness, percentile
18	PCREFz	n/a	1.125	Text Easability PC Referential cohesion, z score
19	PCREFp	n/a	86.860	Text Easability PC Referential cohesion, percentile
20	PCDCz	n/a	-4.796	Text Easability PC Deep cohesion, z score
21	PCDCp	n/a	0	Text Easability PC Deep cohesion, percentile
22	PCVERBz	n/a	13.558	Text Easability PC Verb cohesion, z score
23	PCVERBp	n/a	100	Text Easability PC Verb cohesion, percentile
24	PCCONNz	n/a	-5.565	Text Easability PC Connectivity, z score
25	PCCONNp	n/a	0	Text Easability PC Connectivity, percentile
26	PCTEMPz	n/a	-27.071	Text Easability PC Temporality, z score
27	PCTEMPp	n/a	0	Text Easability PC Temporality, percentile
Referential Cohesion				
28	CRFNO1	CRFBN1um	0	Noun overlap, adjacent sentences, binary, mean

Ilustración 4. Resultados de Coh-Metrix

La herramienta a desarrollar (*AzterTest*) estará ampliamente inspirada en *Coh-Metrix*, puesto que se pretende que la aplicación sea capaz de analizar textos y calcular múltiples métricas, de manera que sean útiles para evaluar la complejidad de los textos.

Sin embargo, en esta herramienta no es posible analizar más de un texto a la vez y, además, es necesario introducir el contenido del documento en un cuadro de texto. Por ello, con *AzterTest* se pretende mejorar estos aspectos y, por tanto, realizar una versión mejorada de *Coh-Metrix*, en la que sí se pueda analizar más de un documento a la vez y subiendo los propios ficheros (en formatos como TXT o DOC).

3.2 WEB-BASED L2 SYNTACTIC COMPLEXITY ANALYZER

L2 Syntactic Complexity Analyzer (L2SCA) es una herramienta que permite analizar la complejidad sintáctica de textos escritos en inglés, y está orientada a profesores e investigadores que estén acostumbrados a trabajar con multitud de muestras de texto. Fue desarrollada por el Profesor Xiaofei Lu de la Universidad Estatal de Pensilvania (Lu, 2017).

Esta herramienta ofrece dos posibilidades: analizar un solo texto, o analizar varios archivos de texto (con un límite de 30) a la vez. Si se escoge analizar un único texto,

existe la opción de insertar un segundo texto, con el objetivo de comparar la complejidad sintáctica de ambos.

Los resultados se muestran de manera gráfica y en forma de tabla (ver Ilustración 5).

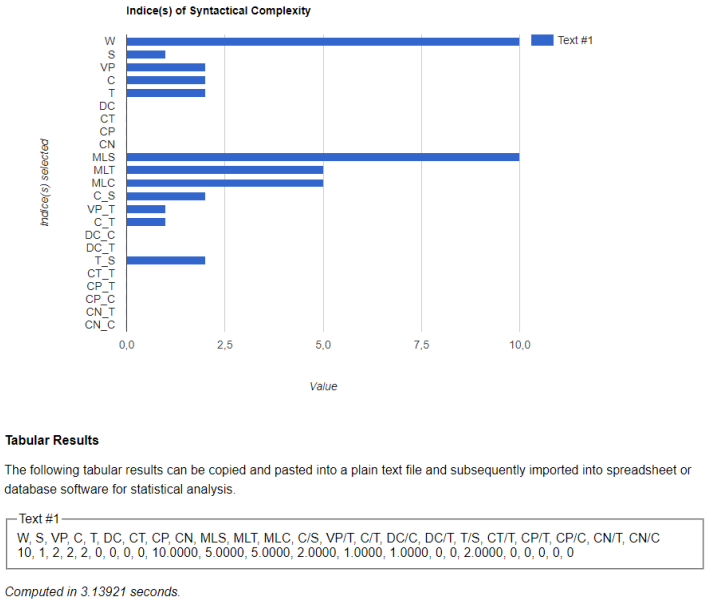


Ilustración 5. Resultados de L2SCA

Se pretende que *AzterTest* funcione de la misma forma; es decir, que sea posible analizar varios ficheros al mismo tiempo y que, después se muestren los resultados en forma de tabla.

Sin embargo, puesto que la forma en la que *L2SCA* representa los datos no es muy agradable a la vista (los datos mostrados en la tabla son confusos y difíciles de leer) y, además, no es posible descargar dichos resultados, se busca una forma más visual de representarlos y que también sea posible descargarlos.

3.3 READABLE

Readable es una herramienta que analiza y puntúa texto en base a fórmulas de lecturabilidad, por lo que le muestra al usuario qué puntuación ha sacado su texto y, por ende, especifica cuán fácil es de leer dicho texto (ver Ilustración 6).

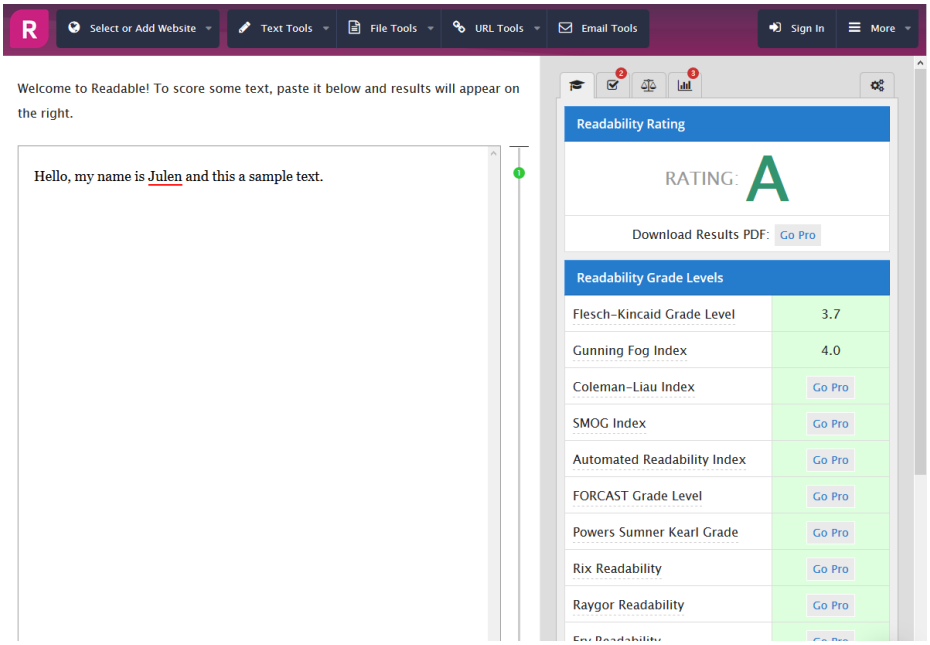


Ilustración 6. Ejemplo de uso para Readable

Además, muestra distintas estadísticas sobre el texto, así como el número de palabras, sentencias y párrafos. Lo hace de manera muy visual y en forma de tabla, por lo que resulta sencillo encontrar un dato específico (Ilustración 6Ilustración 7).

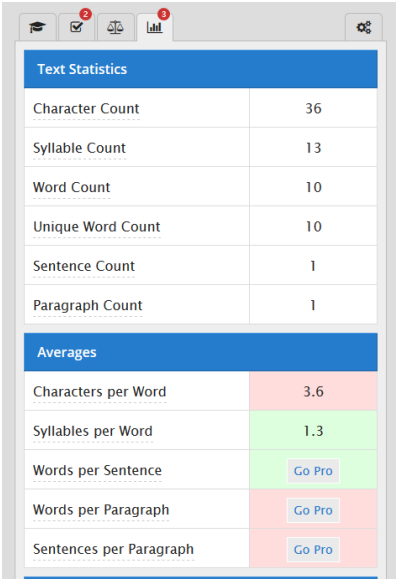


Ilustración 7. Estadísticas de texto en Readable

Visualmente, esta es la herramienta que más llama la atención, por lo que la aplicación web que se va a desarrollar estará ampliamente inspirada en su forma de representar las métricas y los distintos indicadores.

4 Captura de requisitos

En este capítulo se va a mostrar la captura de requisitos de *AzterTest* (nombre con el que se ha bautizado a la aplicación a desarrollar, tras acordarlo con el director del proyecto), en el que se define, entre otros aspectos, qué funcionalidades debe tener la aplicación a desarrollar, qué necesidades tiene que satisfacer y a quién va dirigido.

La captura de requisitos es el paso en el que se trata de entender las necesidades que el usuario final necesita, y se trata de obtener, a partir de estas necesidades, una serie de funcionalidades. Por ello, este paso es muy importante para realizar una buena aplicación.

4.1 REQUISITOS

En este apartado se presentan los requisitos que debe satisfacer la aplicación. Por tanto, se definen las funcionalidades del sistema que debe cumplir este proyecto y, además, se especificará a qué tipo de usuario irá dirigida la aplicación.

4.1.1 Usuario final

La aplicación va dirigida, principalmente a profesores y/o investigadores que tengan que manipular grandes volúmenes de texto, ya que esta ayudará al usuario a conocer cuán complejos son los textos que están manejando y, en el caso de los profesores, pueden saber, por ejemplo, qué grado de complejidad tienen los textos de sus alumnos, o si sus textos son adecuados para el nivel de sus alumnos.

4.1.2 Requerimientos

Cuando el proyecto haya finalizado, la aplicación deberá cumplir con las funcionalidades que se exponen a continuación:

- Permitir analizar uno o más ficheros de texto en una sola ejecución. El sistema debe permitir al usuario seleccionar uno o más ficheros de texto para que la evaluación de éstos se realice de manera simultánea, de manera que el usuario no tenga que seleccionar y analizar los ficheros de uno en uno.

- Permitir analizar documentos en varios formatos para que el usuario no se tenga que preocupar en convertir sus documentos a un formato específico. Por ello, la aplicación deberá permitir analizar ficheros en formato TXT, DOC, DOCX y ODT.

- Mostrar al usuario los resultados del análisis de una forma limpia y ordenada. En caso de que el usuario haya decidido analizar más de un fichero, aparecerá un índice de resultados con el nombre de cada fichero y, si el usuario pincha sobre uno de ellos, le llevará a la tabla de resultados correspondiente a dicho fichero.

- El sistema debe permitir la descarga de un único fichero en formato CSV con todos los resultados del análisis realizado. De esta manera, se consigue que el usuario no tenga que recurrir a volver a analizar los ficheros en caso de que desee volver a consultar los resultados.

- El usuario debe tener claro qué significan cada una de las métricas, por lo que, en una página del sitio web, se debe mostrar una tabla o un listado con todas las métricas, y que cada una de ellas esté explicada de manera apropiada. Además, en esta misma página, conviene explicar qué es y qué hace la aplicación.

- Por último, el sistema debe clasificar cada uno de los ficheros según su complejidad. Esto es, un texto podrá ser fácil, intermedio o difícil en función de las métricas de complejidad calculadas.

4.2 CASOS DE USO

Tras tener claros los requisitos funcionales de la aplicación web que se va a desarrollar, en esta sección se van a identificar y analizar los casos de uso del sitio web. En la Ilustración 8 se pueden observar los casos de uso correspondientes a la aplicación web.

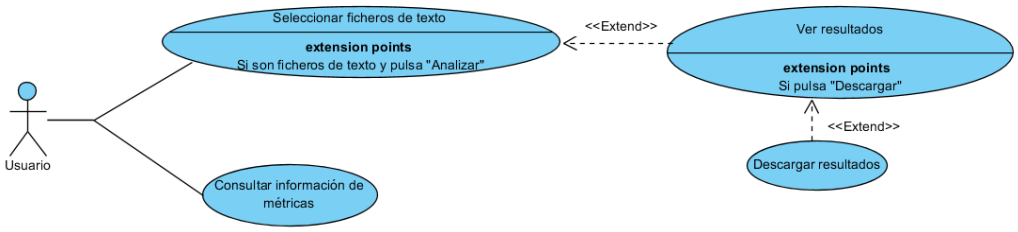


Ilustración 8. Casos de Uso

Como se puede observar en la ilustración anterior, el sistema consta de un solo actor (llamado "Usuario") y los casos de uso que se han identificado son los siguientes:

- **Seleccionar ficheros.** Permite al usuario seleccionar cualquier cantidad de ficheros que se encuentren en su equipo. Si el usuario ha seleccionado únicamente ficheros de texto y pulsa el botón "Analizar", estará disponible el caso de uso que se presenta a continuación:
    - **Ver resultados.** Para cada uno de los ficheros de texto, se procede al análisis y cálculo de las métricas correspondientes. Y, una vez termine el proceso, se mostrarán los resultados. En caso de que el usuario pulse sobre "Descargar", estará disponible el siguiente sub-caso de uso:
        - **Descargar resultados.** El usuario podrá descargar los resultados de los ficheros analizados en formato CSV.
- **Consultar información de métricas.** Permite al usuario acceder a toda la información disponible sobre las métricas que la aplicación web es capaz de analizar. Esta información se mostrará en forma de tabla y estará ordenada según la categoría del indicador.

Cabe destacar que en el Anexo I están disponibles los casos de uso extendidos de la aplicación.

4.3 MODELO DE DOMINIO

En esta sección, se presentará y detallará el diagrama de modelo de dominio que se ha generado para la aplicación, siguiendo y cumpliendo los requerimientos funcionales que se han definido anteriormente.

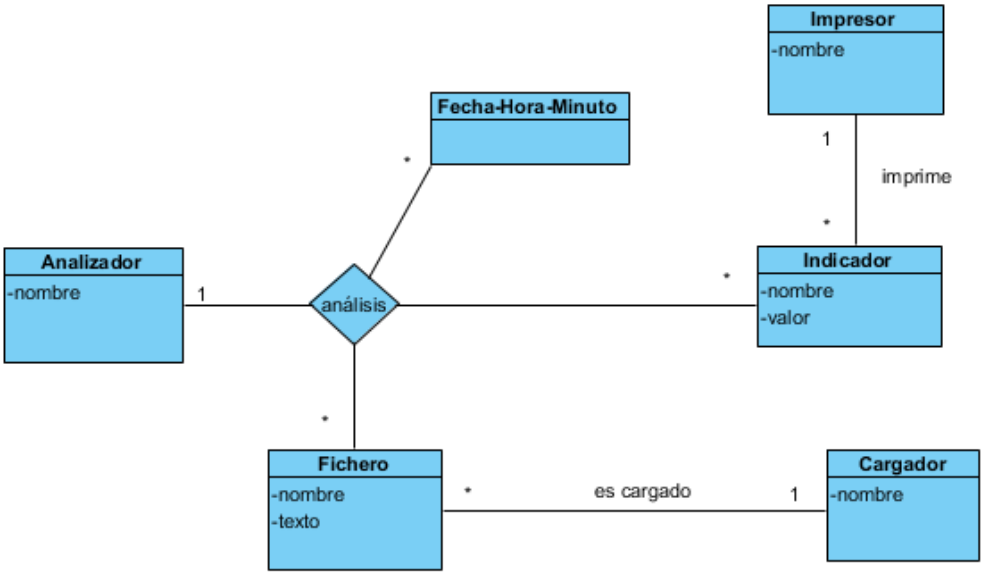


Ilustración 9. Modelo de Dominio.

Como puede observarse en la ilustración anterior (Ilustración 9), existen varias entidades que se van a detallar a continuación.

- **Analizador**: Realiza el análisis y cálculo de los indicadores identificados sobre los ficheros de texto que hayan sido cargados.
- **Fichero**: El fichero de texto contiene el texto correspondiente al contenido de dicho fichero.
- **Cargador**: Se encarga de cargar los ficheros de texto de manera correcta.
- **Indicador**: Contiene el nombre del indicador a analizar, además de su valor (una vez haya sido analizado).
- **Impresor**: Se encarga de recoger todos los indicadores calculados y ordenarlos para imprimirlos por pantalla y generar un fichero con los resultados.
- **Fecha-Hora-Minuto**: Guarda la fecha (incluyendo hora y minuto) en la que se ha realizado un análisis concreto.

Por otro lado, las relaciones que se han identificado son:

- **Es cargado**: Establece un vínculo binario entre la entidad Fichero y la entidad Cargador. Un Fichero puede ser cargado únicamente por un Cargador, mientras que un Cargador puede cargar múltiples Ficheros.
- **Imprime**: Relación binaria entre la entidad Indicador y la entidad Impresor. Establece que un Impresor puede imprimir varios Indicadores, mientras que un Indicador solo puede ser impreso por un Impresor.
- **Análisis**: Relación múltiple entre las entidades Analizador, Fichero, Indicador y Fecha-Hora-Minuto. Un análisis se realiza por un Analizador, sobre varios Ficheros y calculando varios Indicadores. Además, un usuario puede repetir el mismo análisis cuantas veces desee.

5 Análisis y diseño

La aplicación se divide en dos sistemas: la aplicación *Python*, que será la encargada de realizar el análisis de los textos, y la aplicación web, que servirá de interfaz de usuario y será, por tanto, una forma más visual de analizar y clasificar los textos.

En definitiva, el producto final constituye la integración de ambos sistemas, de manera que se dispondrá de la aplicación web, donde el usuario podrá subir y analizar ficheros, y la aplicación *Python*, que se ejecutará en el servidor por la propia aplicación web. La aplicación *Python*, por lo tanto, podría ser utilizada de manera independiente (sin necesidad de utilizar la aplicación web) utilizando la línea de comandos.

5.1 ANÁLISIS Y SELECCIÓN DE INDICADORES

El análisis del texto se realiza mediante unos indicadores que ayudan a identificar y determinar ciertos aspectos importantes sobre la gramática y semántica del mismo. En esta sección, se explicarán de forma detallada todos los indicadores que se consideran importantes, incluyendo la razón de su utilización.

Para escoger qué indicadores implementar, por tanto, se han analizado, principalmente, los indicadores provistos por la herramienta *Coh-Metrix* (ver apartado 3.1) y, tras realizar dicho análisis, se ha acordado con el director del proyecto, Kepa Xabier Bengoetxea, que los indicadores que se deben implementar son los expuestos a continuación.

5.1.1 Indicadores generales

Analizar métricas superficiales puede ser útil para una evaluación inicial de la complejidad de un texto. Por ello, se desarrollan unos indicadores que ofrecen información básica relativa al texto analizado que más adelante serán útiles a la hora de realizar cálculos que sean más complejos. Es decir, son métricas cuyo objetivo es ofrecer datos estadísticos e información básica sobre el texto.

Número de palabras, párrafos y frases

El número de palabras, frases y párrafos son indicadores que ofrecen información básica y que, además, son necesarios para calcular indicadores más complejos.

Número de palabras distintas

El número de palabras distintas puede ayudar a determinar, de manera preliminar, la variedad léxica de un texto. Como se verá más adelante, la riqueza léxica de un texto puede ser indicador de la complejidad del mismo (Capsada Blanch & Torruella Casañas, 2017).

Número de frases en un párrafo

Los párrafos más largos tienden a ser más difíciles de procesar, por lo que resulta interesante calcular esta métrica, puesto que puede ayudar a determinar la lecturabilidad de un texto.

Por tanto, se calcula la media del número de frases en un párrafo de la siguiente manera:

$$Frases\ en\ un\ párrafo_{Media} = \frac{Número\ total\ de\ frases}{Número\ total\ de\ párrafos}$$

Además, también se calcula la desviación típica, puesto que una desviación típica grande, implicaría que el texto tiene una gran variación en términos de longitud de párrafos (McNamara, Graesser, McCarthy, & Cai, 2014).

Número de palabras en una frase

Las frases más largas tienden a demandar más memoria de trabajo y, por lo tanto, son más difíciles de leer (Graesser, McNamara, Louwerse, & Cai, 2004). Por ello, resulta interesante calcular esta métrica con el objetivo de analizar la lecturabilidad del texto.

Para calcularlo, se realiza la media del número total de palabras entre el número total de frases. Además, se realiza el mismo cálculo pero sin tener en cuenta las *stopwords* (una *stopword* es una palabra vacía o sin significado como los artículos, pronombres, preposiciones, etc.).

Número de sílabas en una palabra

Las diversas fórmulas de lecturabilidad utilizan varias formas de determinar el número de palabras complejas. Una de ellas es contar el número medio de sílabas, puesto que las palabras más difíciles tienden a ser polisílabas y, por lo tanto, cuanto más alto sea el porcentaje de polisílabos, más alta será la probabilidad de que el texto contenga palabras complejas (Stahl, 2003).

Número de letras en una palabra

Las palabras más largas tienden a ser menos frecuentes en el inglés, por lo que los lectores disponen de menos conocimiento sobre estas palabras.

Por ello, se pretende realizar la media del número de letras por palabra y, además, realizar el mismo cálculo, pero sin tener en las *stopwords* (una *stopword* es una palabra vacía o sin significado como los artículos, pronombres, preposiciones, etc.). Por otro lado, también se realizará la media de letras por lema.

5.1.2 Indicadores de riqueza léxica

Conocer el vocabulario es una parte vital del conocimiento de cualquier lenguaje, por lo que medir la riqueza léxica (esto es, cuántas palabras diferentes se utilizan) de un texto puede ayudar a determinar cuán complejo es y qué nivel de vocabulario dispone el autor del texto, ya que, por ejemplo, un texto que repite las mismas palabras una y otra vez tiende a ser más sencillo que uno que trata de utilizar un vocabulario variado. En este apartado, se presentarán indicadores como el *Type-Token Ratio*, que tratan de medir el grado de riqueza léxica del que dispone un texto.

Densidad léxica

La densidad léxica es un término comúnmente usado para describir la proporción de palabras de contenido (sustantivos, verbos, adjetivos y adverbios) frente el número total de palabras. Esto ofrece una noción de lo empaquetada que se encuentra la información del texto, puesto que el hecho de que un texto tenga una alta proporción de palabras de contenido significa que éste contiene más información que uno con una alta proporción de palabras funcionales (esto es, preposiciones, pronombres, conjunciones, etc.) (Johansson, 2008). Por esta razón, se calcula dicha proporción de la siguiente manera:

$$Densidad\ léxica = \frac{Total\ de\ palabras\ de\ contenido}{Total\ de\ palabras}$$

Por otro lado, existen variantes de la densidad léxica, como la densidad de sustantivos (el número de sustantivos dividido por el número de *tokens*/palabras en el texto). Por ello, se calcula la densidad para los sustantivos, adjetivos, verbos y adverbios), de la misma manera que se hace con la densidad léxica.

TTR (Type-Token Ratio)

La palabra única en un texto es un tipo (*type*) de palabra y cada instancia de una palabra es un *token*. El *Type-Token Ratio* (TTR) es el cociente que existe entre el número

de tipos (*types*) y el número total de palabras (*tokens*), y su valor está comprendido entre 0 y 1. Por ejemplo, si la palabra *house* aparece en el texto cinco veces, su valor de tipo (*type*) será 1, mientras que el de *token* será 5. La proporción *token*-tipo (TTR) es el número de palabras únicas (*type*) dividido por el número de *tokens* de palabras.

Cuando la relación *type-token* es 1, significa que cada palabra aparece únicamente una sola vez en el texto, lo cual implica que la comprensión debería ser más difícil, puesto que varias palabras únicas deben ser identificadas e integradas en el contexto del discurso.

Por otro lado, una relación *type-token* con un valor bajo, indicaría que las palabras se repiten múltiples veces a lo largo del texto, lo cual implicaría, generalmente, un crecimiento en la facilidad y velocidad de lectura y procesado del texto (Graesser, McNamara, Louwerse, & Cai, 2004).

La relación entre los *types* y los *tokens* de un texto no comprenden una relación lineal (de proporcionalidad) y, como consecuencia, si se calcula la TTR a medida que se aumenta el número de *tokens*, como el número de tipos aumentará con menos frecuencia, el valor de la TTR tenderá a disminuir (Capsada Blanch & Torruella Casañas, 2017).

En definitiva, se puede concluir que la TTR es una medida que presenta limitaciones a la hora de estudiar la riqueza léxica de un texto, puesto que su valor depende de la longitud del texto y, en caso de querer comparar dos textos, no será una medida muy útil a menos que estos tengan una longitud similar o igual.

Teniendo estos aspectos en cuenta, las medidas TTR que se desarrollan son las siguientes:

a. Simple Type-Token Ratio

Como se ha explicado anteriormente, el TTR simple es el cociente que existe entre el número de tipos (*types*) y el número total de palabras (*tokens*), y su valor está comprendido entre 0 y 1:

$$STTR = \frac{V}{N}$$

V → Número de tipos
N → Número de palabras en el texto

b. Content Type-Token Ratio

En este caso, se calcula el TTR de las palabras de contenido. Es decir, es el cociente entre el número de tipos (*types*) de palabras de contenido (esto es, sustantivos, verbos, adjetivos y adverbios) y el número total de palabras de contenido:

$$CTTR = \frac{V_C}{N_c}$$

V_C → Número de tipos de palabras de contenido
N_c → Número de palabras de contenido en el texto

c. Noun Type-Token Ratio

Se calcula el TTR de los sustantivos. Por lo tanto, se trata del cociente entre el número de tipos (*types*) de sustantivos y el número total de sustantivos:

$$NTTR = \frac{V_N}{N_N}$$

V_N→ Número de tipos de sustantivos
N_N → Número de sustantivos

d. Verb Type-Token Ratio

Se calcula el TTR de los verbos. Por lo tanto, se trata del cociente entre el número de tipos (*types*) de verbos y el número total de verbos:

$$VTTR = \frac{V_V}{N_V}$$

V_V→ Número de tipos de verbos
N_V → Número de verbos

e. Adjective Type-Token Ratio

Se calcula el TTR de los adjetivos. Por lo tanto, se trata del cociente entre el número de tipos (*types*) de adjetivos y el número total de adjetivos:

$$AdjTTR = \frac{V_{Adj}}{N_{Adj}}$$

V_{Adj}→ Número de tipos de adjetivos
N_{Adj} → Número de adjetivos

f. Adverb Type-Token Ratio

Se calcula el TTR de los adverbios. Por lo tanto, se trata del cociente entre el número de tipos (*types*) de adverbios y el número total de adverbios:

$$AdvTTR = \frac{V_{Adv}}{N_{Adv}}$$

V_{Adv}→ Número de tipos de adverbios
N_{Adv} → Número de adverbios

g. Type-Token Ratio con lemas

Por último, se realiza el cálculo del Type-Token Ratio, pero utilizando los lemas de las palabras en lugar de las palabras *per se*. Es decir, se calcula el *Simple*, *Content*, *Noun*, *Verb*, *Adjective* y *Adverb* TTR teniendo en cuenta únicamente los lemas.

El lema es la forma que por convenio se acepta como representante de todas las formas flexionadas (es decir, en plural, en femenino, conjugadas, etc.), de una misma palabra. Por ejemplo, el lema de *better* es *good*, el de *walking* es *walk*, etc.

Honoré

Otra alternativa para calcular la riqueza léxica de un texto teniendo en cuenta la longitud del texto es la propuesta por Honoré (1979). Esta métrica se mide a partir del número de *hapax legomena* que contiene un texto. Un *hapax legomena* es cualquier palabra que únicamente aparece una sola vez en el texto. El índice propuesto por Honoré se mide con la siguiente fórmula (Capsada Blanch & Torruella Casañas, 2017):

$$Honoré = 100 \cdot \frac{\log N}{1 - \frac{V_1}{V}}$$

N → Número de palabras en el texto
V1 → Número de *hápax legonema*
V → Número de tipos

Maas

Maas es otra medida destacada que sirve calcular la riqueza léxica de un texto. La particularidad de este índice es que tiene una variación inversa. Es decir, cuanto mayor sea su valor, menor será la riqueza léxica del texto analizado. Esta observación hay que

tenerla en cuenta para evitar posibles confusiones a la hora de interpretar el indicador. La fórmula correspondiente a esta medida, por tanto, es la siguiente (Capsada Blanch & Torruella Casañas, 2017):

$$Maas = \frac{\log N \cdot \log V}{\log^2 \mathrm{V}}$$

N → Número de palabras en el texto
V → Número de tipos

MTLD

El índice MTLD (*Measure of Textual Lexical Diversity*), que fue propuesto por McCarthy en el año 2005, divide el texto en segmentos y calcula la relación *token*-tipo (TTR) de cada uno de ellos. Además, resulta ser una medida estable puesto que no depende mucho de la longitud del texto, por lo que es útil tanto como para textos cortos como para textos de mayor tamaño. El proceso que ha de seguirse para calcular el MTLD de un texto es el siguiente (Capsada Blanch & Torruella Casañas, 2017):

1. Secuencialmente, comenzando desde el principio del texto, se crean segmentos que irán aumentando palabra a palabra.
2. Por cada segmento, se calcula su TTR.
3. Puesto que a medida que los segmentos se alargan, el valor del TTR irá disminuyendo, cuando su valor sea menor a un umbral previamente determinado (establecido en 0,72 a partir de la observación práctica de Capsada Blanch et al.), el segmento se considera como completo y, por tanto, se comienza a crear otro nuevo y se repite la misma operación.

Cuando el proceso haya finalizado, la *Measure of Textual Lexical Diversity* se calcula de la siguiente manera (Capsada Blanch & Torruella Casañas, 2017):

$$MTLD = \frac{N}{n}$$

N → Número de palabras en el texto
n → Número de segmentos

Con el objetivo de reducir los efectos de la aleatoriedad que tienen los segmentos en cuanto a su longitud, el valor definitivo de MTLD se calcula realizando la media de los valores que se obtienen al medir su valor dos veces (Capsada Blanch & Torruella Casañas, 2017):

a) En sentido directo (es decir, en el orden de lectura).
b) En sentido contrario (es decir, en el orden inverso al de lectura).

5.1.3 Indicadores de lecturabilidad

Existen varias fórmulas que, en base a la longitud de las palabras y frases, proveen indicaciones de la lecturabilidad de un texto. Calculando la lecturabilidad de un texto, se puede obtener una idea de su complejidad. A continuación, se presentan las fórmulas de lecturabilidad más utilizadas.

Flesch Reading Ease

Junto con *Flesch-Kincaid Grade Level*, *Flesch Reading Ease* es una de las fórmulas más comunes para el cálculo de la lecturabilidad de un texto. Esta fórmula devuelve un número entre el 0 y el 100 y, cuanto más alto sea el valor, más legible (ergo, más fácil de leer) será el texto. El valor de esta métrica se calcula utilizando la siguiente fórmula (Graesser, McNamara, Louwerse, & Cai, 2004):

$$Flesch\ Reading\ Ease = 206.835 - 1.015 \cdot \mathrm{ASL} - 84.6 \cdot \mathrm{ASW}$$

ASL → Longitud media de las frases del texto
ASW → Media de sílabas por palabra

Flesch-Kincaid Grade Level

Por otro lado, está la fórmula de *Flesch-Kincaid Grade Level*, que convierte el valor de *Flesch Reading Ease* a un grado académico del sistema educativo de Estados Unidos por lo que, en este caso, cuanto más alto sea el número más difícil de leer será el texto (Graesser, McNamara, Louwerse, & Cai, 2004). La fórmula que se utiliza para realizar este cálculo es la siguiente:

$$Flesch\text{–}Kincaid\ Grade\ Level = 0.39 \cdot \mathrm{ASL} + 11.8 \cdot \mathrm{ASW} - 15.49$$

ASL → Longitud media de las frases del texto
ASW → Media de sílabas por palabra

En general, el texto debería tener más de 200 palabras para que tanto *Flesch-Kincaid Grade Level* como *Flesch Reading Ease* puedan devolver valores satisfactorios (Graesser, McNamara, Louwerse, & Cai, 2004).

Dale-Chall readability formula

Otra métrica de medición de la lecturabilidad de un texto se realiza mediante la fórmula de Dale-Chall. Esta fórmula, que utiliza un listado de 3000 palabras consideradas fáciles[5], se calcula de la siguiente manera (Tavernier & Bellot, 2012):

$$Dale - Chall_{Bruto} = 0.1579 \cdot \text{DS} + 0.0496 \cdot \text{ASL}$$

DS → Porcentaje de palabras difíciles (es decir, aquellas que no se encuentren en la lista de 3000 palabras simples)
ASL → Media de sílabas por palabra

Si el porcentaje de palabras difíciles (DS) es menor que 5%, entonces ha de sumarse 3.6365 al valor bruto de la fórmula de Dale-Chall, de manera que la fórmula sería la siguiente:

$$Dale - Chall_{Ajustado} = 0.1579 \cdot \text{DS} + 0.0496 \cdot \text{ASL} + 0.6365$$

Simple Measure Of Gobbledygook (SMOG) grade

El método de SMOG (*Simple Measure Of Gobbledygook*) se utiliza también para determinar el nivel de lecturabilidad de un texto. Esta métrica aproxima la edad necesaria para entender el texto, y la fórmula a utilizar sería la siguiente (Tavernier & Bellot, 2012):

$$SMOG = 1.0430 \cdot \sqrt{NPS \cdot \frac{30}{NS}} + 3.1291$$

NPS → Número de polisílabos (palabras con 3 o más sílabas)
NS → Número de frases

5.1.4 Indicadores semánticos de lecturabilidad

Además de las previamente mostradas fórmulas de lecturabilidad, también es posible determinar la lecturabilidad de un texto mediante el cálculo de sus valores de polisemia e hiperonimia.

Una palabra se considera ambigua cuando ésta contiene múltiples significados. Por ejemplo, la palabra *bank* tiene al menos dos significados:

- Lugar para guardar dinero.
- Tierra junto a un cuerpo de agua.

[5] Listado de palabras de Dale-Chall: https://readable.com/blog/the-dale-chall-word-list/

En cambio, una palabra es abstracta cuando tiene muy pocas características distintivas, por lo que puede llegar a tener distintas representaciones.

Una forma de medir la ambigüedad de una palabra es mediante los valores de polisemia en WordNet, mientras que la abstracción puede ser medida mediante los valores de hiperonimia en WordNet (Graesser, McNamara, Louwerse, & Cai, 2004).

La polisemia se mide contando el número de significados de una palabra. Una palabra con más significados, generalmente, será más ambigua y, en consecuencia, más lenta de procesar (especialmente para lectores con menos habilidades de lectura y conocimiento del lenguaje) (Graesser, McNamara, Louwerse, & Cai, 2004).

La hiperonimia, por otro lado, se calcula contando el número de niveles dentro de la jerarquía. Por ejemplo, la palabra *chair* tiene siete niveles de hiperonimia:

seat → *furniture* → *furnishings* → *instrumentality* → *artifact* → *object* → *entity*

En definitiva, cuantos más niveles de hiperonimia tenga una palabra, tenderá a ser más específica (y, por lo tanto, menos abstracta), por lo que será más fácil de procesar (Graesser, McNamara, Louwerse, & Cai, 2004).

El objetivo es calcular la media de los valores de polisemia e hiperonimia de aquellas palabras que tengan entradas en el lexicón de WordNet. Por lo tanto, se han calculado los siguientes indicadores:

a) Media de los valores de polisemia

Cuanto mayor sea la media de los valores de polisemia, más ambiguas serán las palabras que componen el texto y, por lo tanto, más difícil de procesar.

b) Media de los valores de hiperonimia

De una manera similar, un valor pequeño de la media de los valores de hiperonimia es un refleja un uso general de palabras menos específicas, mientras que un valor más grande significa que se utilizan palabras más específicas (es decir, más difíciles de procesar). Esta métrica se realiza para sustantivos, verbos y una combinación de ambos (sustantivos y verbos).

5.1.5 Indicadores morfológicos

Número de verbos en tiempo pasado, presente y futuro

La temporalidad en un texto hace referencia a la utilización del tiempo. En un texto, el tiempo se representa mediante morfemas de tiempo verbal (por ejemplo, *"-ed"*, *"is"* o *"has"*). Los tiempos verbales, por lo tanto, determinan la consistencia de un texto en cuanto a su temporalidad y, cuanto más consistente sea un texto, más fácil de procesar y entender será (McNamara, Graesser, McCarthy, & Cai, 2014). Por esa razón, se calcula el número de verbos en pasado, presente y futuro.

Número de verbos en modo indicativo, imperativo y subjuntivo

Algunos elementos morfológicos son más frecuentes que otros, por lo que algunos de estos elementos solo aparecen en textos de alto nivel. Por ejemplo, el modo subjuntivo es menos frecuente que el indicativo, y aparece más en textos con una complejidad mayor (Madrazo, 2016). Por esa razón, se calcula el número de verbos en modo indicativo y subjuntivo y, además, se añade el cálculo del modo imperativo.

Número de verbos irregulares en tiempo pasado

Se calcula el número total de verbos irregulares en tiempo pasado porque son más difíciles de procesar y leer (Spache, 1953), por lo que es una medida interesante para determinar la complejidad de un texto.

Número de pronombres

Las palabras funcionales son aquellas que disponen de un significado léxico muy pequeño o de gran ambigüedad, puesto que se utilizan, principalmente, para crear relaciones gramaticales entre palabras (por ejemplo, *if* y *or)*. Una categoría importante de palabra funcional es el pronombre. Los pronombres indican en qué modo narrativo está escrito el texto (por ejemplo, en primera persona) y, además, tienen repercusión en la cohesión y coherencia de un texto, puesto que, si el lector es incapaz de enlazar un pronombre a una referencia (ya sea un objeto, un animal o una persona), es posible que tampoco sea capaz de conectar las ideas del texto. Por ello, la frecuencia de pronombres en el texto tiene correlación con la dificultad de un texto, en el sentido de que los referentes de los pronombres sean o no difíciles de resolver (McNamara, Graesser, McCarthy, & Cai, 2014).

En definitiva, se calcula el número total de pronombres personales, además del número de pronombres en primera persona, primera persona del singular y tercera persona.

5.1.6 Indicadores de frecuencia de palabra (*Word Frequency*)

Existen diversas formas de determinar la cantidad de palabras complejas que hay contenidas en un texto. Una de ellas es contando el número de sílabas por palabra (tal y como se explicó anteriormente), ya que las palabras más difíciles tienden a ser polisílabas, por lo que, si el porcentaje de polisílabos es alto, es probable que exista una alta cantidad de palabras complejas. No obstante, esto no siempre es cierto, puesto que existen palabras como *together* que son más sencillas que palabras monosílabas tales como *din* o *phlegm* (Stahl, 2003).

Por esa razón, como se desea obtener un mejor enfoque para encontrar palabras difíciles, se van a utilizar listas de frecuencias de palabras, por lo que se clasifica cada palabra en función de su frecuencia. Las palabras frecuentes son procesadas y entendidas más rápidamente que las palabras infrecuentes (Crossley & Allen, 2011).

El objetivo será clasificar las palabras difíciles según su frecuencia. Es decir, se relacionará la frecuencia de la palabra (basándose en estimaciones dadas por la librería *wordfreq*, que se presentará más adelante) con la dificultad de ésta.

Para simplificar el valor de la frecuencia de una palabra de manera que sea más legible para el ser humano, se suele utilizar la escala *Zipf*, propuesta por Marc Brysbaert, que es el logaritmo en base 10 del número de veces que aparece la palabra por cada mil millones de palabras. Por ejemplo, si una palabra tiene una frecuencia *Zipf* de 6, significa que aparece 10^6 veces en 10^9 palabras; es decir, una vez por cada 1000 (Speer, s.f.).

Por lo tanto, se contará el número de adjetivos, sustantivos, verbos y adverbios que tengan valores *Zipf* menor o igual a 4, ya que este es el punto de inflexión en el que las palabras tienden a ser más difíciles (Mandera, 2016).

5.1.7 Indicadores de conocimiento del vocabulario

Para determinar el nivel del lenguaje de un texto, se ha escogido utilizar el *Common European Framework of Reference* para clasificar las palabras del texto según su nivel de complejidad. El *Common European Framework of Reference* (CEFR) organiza la habilidad de lenguaje en seis niveles, del A1 al C2, los cuales pueden agruparse en tres categorías generales: básico, intermedio y avanzado. Más concretamente, los niveles A1 y A2 indican habilidades básicas y pre-intermedias, mientras que B1 y B2 indican niveles intermedios. Por otro lado, C1 indica un nivel avanzado y, por último, C2 indica un dominio completo del lenguaje.

Oxford ofrece un diccionario de palabras con niveles del A1 al C1, por lo que el objetivo de estos indicadores es contar el número de palabras para cada uno de los niveles del CEFR.

5.1.8 Indicadores sintácticos

Palabras de contenido

Las palabras de contenido (o palabras léxicas), son aquellas que hacen referencia a objetos de la realidad. Es decir, son aquellas palabras que disponen de un significado. Lo opuesto a este concepto serían las palabras funcionales, las cuales disponen de muy poco o ningún significado, puesto que se utilizan principalmente para crear relaciones gramaticales entre palabras (tal y como se ha explicado anteriormente).

Se calcula, por tanto, el número de palabras de contenido (nombres, adjetivos, verbos y adverbios), puesto que ofrecen información acerca de la sintaxis del texto. Por ello, los indicadores que se calculan son:

a) Número total de palabras de contenido
b) Número total de nombres
c) Número total de adjetivos
d) Número total de verbos
e) Número total de adverbios

Left embeddedness

Se calcula la *left embeddedness* o, dicho de otra manera, el número de palabras delante del verbo principal, ya que las frases que disponen de construcciones sintácticas complejas son, normalmente, estructuralmente densas y, por consiguiente, más difíciles de comprender y procesar (McNamara, Graesser, McCarthy, & Cai, 2014). Con esta métrica (la media del número de palabras delante del verbo principal), por tanto, se consigue determinar cuán sintácticamente y estructuralmente complejo es un texto.

Número de modificadores por sintagma nominal

Se calcula la media de modificadores que existen en el texto por cada uno de los sintagmas nominales (*noun phrase*), puesto que se trata de un indicador que, cuando en un sintagma nominal hay muchos modificadores, se necesita mayor memoria de trabajo, lo cual hace que el texto sea más difícil de leer (McNamara, Graesser, McCarthy, & Cai, 2014). Los modificadores son palabras o frases que modifican el núcleo del sujeto en una oración.

Por lo tanto, además de calcular el número de modificadores, se realiza el cálculo del número de descendientes de estos modificadores. Es decir, por cada sintagma nominal, se calcula el total de descendientes modificadores que parten de los modificadores padre.

Profundidad media del árbol de dependencia

Cuando mayor profundidad tenga el árbol de dependencias de una frase, más compleja será (Madrazo, 2016). Por lo tanto, se realiza el cálculo de la profundidad media de todos los árboles de dependencia que contenga el texto.

Por ejemplo, en la Ilustración 10 puede verse el ejemplo de un árbol de dependencia. La palabra *is* sería el padre (*root*), y este nodo tiene 3 hijos (*hearing, scheduled* y un punto como marca de puntuación). Para el caso de *hearing*, se tienen dos hijos (*A* y *on*), mientras que *scheduled* únicamente tiene uno (*today*). Después, *scheduled* tendría el hijo *today*, y *on* tendría el hijo *issue,* cuyo hijo es *the*.

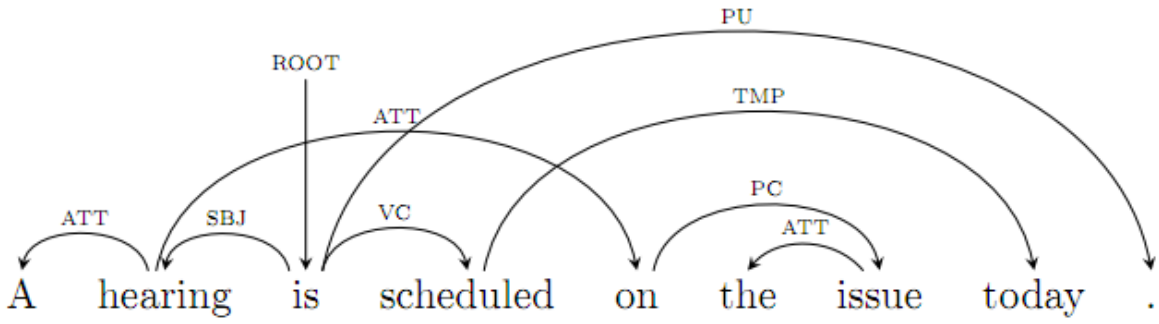


Ilustración 10. Ejemplo de árbol de dependencia.

En la Ilustración 11 se pueden ver las dependecias y la profundidad del árbol de una manera más clara que, en este caso, es cinco.

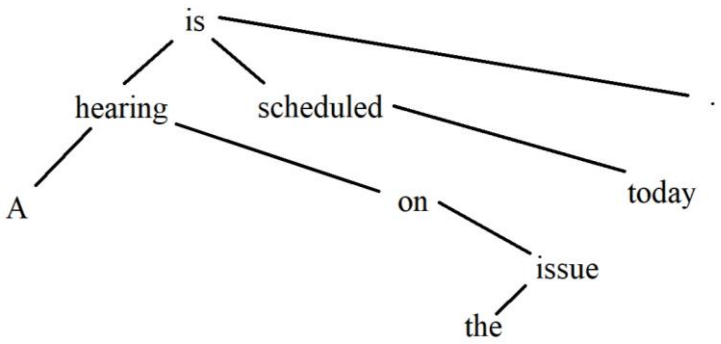


Ilustración 11. Árbol de dependencia.

Número total de oraciones subordinadas y subordinadas relativas

Las oraciones subordinadas y subordinadas relativas son elementos que añaden complejidad al texto. Por eso, existen sistemas de simplificación de la sintaxis del texto que, entre otros aspectos, deben procesar este tipo de oraciones para hacer que el texto sea más fácil de leer (Siddharthan, 2002). Debido a esto, se calculará el número de oraciones subordinadas y subordinadas relativas que existen en el texto.

Media de marcas de puntuación por frase

Según las *Normas europeas para hacer que la información sea fácil de leer y entender* (Freyhoff, y otros, 1998), es recomendable que para que el texto sea sencillo, éste tenga una puntuación simple. Esto quiere decir que cuantas más marcas de puntuación tenga el texto, más complicado será. Por ello, se calcula el número de marcas de puntuación por frase (es decir, se computa una media de marcas de puntuación).

Número de proposiciones

Las proposiciones son la unidad de significado más básica que componen una frase. Cada proposición contiene un predicado (por ejemplo, adjetivos o conectores) y uno o más argumentos (por ejemplo, sustantivos o pronombres) que tienen un rol temático, así como una persona, un objeto, un espacio temporal, o un lugar. Por ejemplo, la frase "*When the committee met on Monday, they discovered the society was bankrupt*" tiene cuatro proposiciones (McNamara, Graesser, McCarthy, & Cai, 2014):

1) Predicado: *meet* (cuyos argumentos son *committee* y *Monday*)
2) Predicado: *discover* (con los argumentos *committee* y la tercera proposición)
3) Predicado: *bankrupt* (cuyo argumento es *society*)
4) Predicado: *when* (cuyos argumentos son las proposiciones 1 y 2)

Se calculará el número total de proposiciones y la media de proposiciones por frase. Estas métricas ayudarán a la hora de evaluar la complejidad del texto, puesto

que las proposiciones hacen que el proceso de comprensión del texto sea más eficiente (McNamara, Graesser, McCarthy, & Cai, 2014).

Media y densidad de sintagmas nominales y sintagmas verbales

La cantidad de sintagmas nominales (*noun phrase*) y verbales (*verb phrase*) en un texto afectan a la complejidad del mismo. Es decir, si un texto tiene una mayor cantidad de sintagmas nominales y verbales, es más probable que dicho texto sea más denso en cuanto a información y, por tanto, más complejo sintácticamente (McNamara, Graesser, McCarthy, & Cai, 2014). Por esa razón, se realiza el cálculo de la media de *noun phrase* y *verb phrase*, además de la densidad (incidencia) de cada uno de ellos.

Número de verbos en voz pasiva

Los verbos pueden representarse tanto en voz activa como en voz pasiva. Cuando se usa la voz pasiva, el elemento que realiza la acción es el sujeto de la oración, y el elemento que recibe la acción es el objeto. La mayoría de frases son activas (por ejemplo, "*John needs money*"). Por otro lado, la voz pasiva se utiliza cuando el elemento que recibe la acción es el sujeto de la oración y el elemento que la realiza se puede incluir, de manera opcional, al final de la oración (mediante el agente "*by*"). Por ejemplo, la frase "*The house was bought by John*" está construida en voz pasiva.

Las frases en voz pasiva son más difíciles de procesar que las que están en voz activa (McNamara, Graesser, McCarthy, & Cai, 2014), por lo que se realizará el cálculo de los siguientes indicadores:

1) Número de verbos en voz pasiva
2) Número de verbos en voz pasiva sin agente ("*by*")

Número de verbos en infinitivo y gerundio

También se calcula el número de verbos en infinitivo y en gerundio, puesto que ofrecen información acerca de la conjugación de los verbos (McNamara, Graesser, McCarthy, & Cai, 2014). Los verbos en infinitivo son aquellos que no está conjugados (por ejemplo, *be*, *make* o *write*), mientras que los verbos en gerundio son aquellos que terminan en *–ing*.

Número de palabras negativas

El uso de la negación (es decir, principalmente, la palabra not) está relacionado con la dificultad de procesamiento del texto (McNamara, Graesser, McCarthy, & Cai, 2014), por lo que resulta interesante realizar el cálculo del número de palabras negativas que existen en el texto.

5.1.9 Indicadores de cohesión

Superposición de palabras

La cohesión referencial hace referencia a la superposición de palabras de contenido entre las distintas frases y, si las palabras o conceptos en una frase no se superponen (es decir, coinciden) con otras frases del texto, significa que existen brechas o diferencias en la cohesión. Estas brechas o diferencias, dependiendo de las habilidades del lector, pueden tener efectos adversos en la comprensión y tiempo de lectura (McNamara, Graesser, McCarthy, & Cai, 2014).

Por ello, se desarrollan indicadores que determinen la superposición de palabras, con el objetivo de informar acerca de la cohesión referencial del texto. Se distinguen cuatro tipos de cohesión referencial: superposición de sustantivos, superposición de argumentos, superposición de raíces y superposición de palabras de contenido. Cada uno de los cuatro tipos se calcularán tanto de manera local (entre frases adyacentes) como global (todos los posibles pares de frases en un párrafo). A continuación, se procede a explicar en detalle cada una de las cuatro mediciones que se pretenden realizar:

- **Superposición de sustantivos**. En primer lugar, la superposición de sustantivos entre frases adyacentes (superposición local) representa el valor medio de frases en un texto que tienen coincidencias de sustantivos entre una frase y la anterior. De todas las medidas de cohesión referencial, esta es la más estricta, ya que los sustantivos tienen que coincidir exactamente, tanto en forma como en número gramatical (singular o plural). Por ejemplo, no habría superposición entre la *cell* y *cells*, pero sí entre *cell* y *cell*. Por otro lado, la superposición de sustantivos global determina la superposición de sustantivos de cada frase con el resto de frases (McNamara, Graesser, McCarthy, & Cai, 2014).
- **Superposición de argumentos**. Esta medida (local y global) es similar a la anterior, pero, en este caso, se tienen en cuenta tanto sustantivos como pronombres. Existe superposición de argumentos cuando hay coincidencia entre un sustantivo en una frase y el mismo sustantivo (en plural o singular) en la otra frase. También ocurre cuando hay coincidencias de pronombres personales entre las dos frases (por ejemplo, *he* y *he*). Por lo tanto, es evidente que se trata de una métrica de superposición menos estricta (McNamara, Graesser, McCarthy, & Cai, 2014).
- **Superposición de raíces**. Se realiza la superposición local y global mediante el cálculo de las coincidencias que existen entre un sustantivo en una frase y una palabra de contenido en la frase anterior, compartiendo

un lema común (por ejemplo, *tree* y *treed*, o *price* y *priced*) (McNamara, Graesser, McCarthy, & Cai, 2014).

- **Superposición de palabras de contenido.** Este indicador considera la proporción de las palabras de contenido que coinciden entre pares de frases. Por ejemplo, si un par de frases tiene pocas palabras y dos de esas palabras coinciden, la proporción es mayor que en el caso del par de frases tuviera más palabras y dos palabras coincidieran. Esta métrica se realiza tanto de manera local como global y, además, se incluye la desviación estándar. Esta métrica puede ser especialmente útil cuando las longitudes de las frases en un texto son la principal preocupación (McNamara, Graesser, McCarthy, & Cai, 2014).

Similitud semántica

La similitud semántica se utiliza para determinar la cohesión y coherencia de un texto. Esto es, un texto que disponga un alto valor de similitud semántica refleja que las frases y palabras de dicho texto están estrechamente relacionadas en cuanto a significado. Por ejemplo, la palabra *martillo* estará altamente relacionada con palabras del mismo contexto funcional, tales como *clavo*, *sierra* o *construcción*. Estas palabras, aunque no son sinónimos ni hiperónimos de *martillo*, comparten el mismo contexto, lo cual es señal de que el texto está correctamente cohesionado (McNamara, Graesser, McCarthy, & Cai, 2014).

Por lo tanto, mediante este indicador se pretende analizar la coherencia del texto mediante el cálculo de la similitud semántica, lo cual será la media de los valores (que oscilarán entre 0 y 1) de similitud entre textos adyacentes y todos los textos (es decir, la similitud de cada texto respecto al resto de textos). Cabe destacar que cuando se habla de textos, se hace referencia a los párrafos y frases de un mismo documento de texto, y no a dos ficheros de texto distintos e independientes.

Conectores

Los conectores son palabras que conectan dos o más frases, por lo que hay que tenerlos en cuenta a la hora de evaluar la cohesión de un texto, puesto que son un claro reflejo de que las ideas de un texto están correctamente relacionadas. Por ello, se calcula el número de conectores para las siguientes categorías, que son las que se consideran más importantes (McNamara, Graesser, McCarthy, & Cai, 2014):

a) Conectores causales (por ejemplo, *because*)
b) Conectores lógicos (por ejemplo, *or*)
c) Conectores adversativos (por ejemplo, *although*)
d) Conectores temporales (por ejemplo, *before*)

e) Conectores condicionales (por ejemplo, *also*)

5.1.10 Incidencia por cada 1000 palabras

Por último, es importante explicar qué es la incidencia y por qué es tan relevante calcular la incidencia por cada 1000 palabras de algunas de las métricas presentadas anteriormente.

A la hora de comparar algunas métricas para distintos textos, puesto que cada uno de ellos tendrá una cantidad de palabras distinta, es necesaria una manera de encontrar una proporción entre los indicadores de dichos textos. Por ejemplo, no tendría sentido comparar el número de adjetivos de un texto de 700 palabras con uno de 300, puesto que uno de esos textos tiene más palabras que el otro y, por tanto, más cabida para el número de adjetivos. Por ello, se calcula la incidencia por cada 1000 palabras de ese indicador. De esta manera, se consigue igualar de manera proporcional el número de adjetivos. Para entenderlo mejor, si el texto de 700 palabras tuviera 53 adjetivos y el texto de 300 palabras tuviera 23, se tendría que realizar la incidencia por cada mil palabras, de manera que:

$$Incidencia\ Núm.adj._{Texto\ 700\ palabras} = \frac{53\ adjetivos\ \cdot 1000\ palabras}{700\ palabras} = 75{,}71$$

$$Incidencia\ Núm.adj._{Texto\ 300\ palabras} = \frac{23\ adjetivos\ \cdot 1000\ palabras}{300\ palabras} = 76{,}67$$

Como se puede observar, de esta manera ambos textos quedan igualados, y se puede realizar la comparación del número de adjetivos de manera justa (en lugar de comparar el 53 y el 23, se compara el 75.71 con el 76.67 y se concluye que ambos textos tienen una incidencia de adjetivos similar).

Esto también se podría realizar utilizando porcentajes (lo cual sería la incidencia sobre 100 palabras) pero, generalmente, en el análisis de complejidad y lecturabilidad de textos se realiza la incidencia por cada 1000 palabras porque ofrece los resultados de manera más clara (McNamara, Graesser, McCarthy, & Cai, 2014).

5.2 DISEÑO DE LA APLICACIÓN *PYTHON*

La aplicación *Python* será ejecutada en el servidor cuando el usuario desee analizar ficheros de texto, pero también podrá ser una aplicación independiente que se ejecute

mediante la línea de comandos, pero, en caso de que se utilice así, no dispondrá de interfaz de usuario y los resultados no serán representados de manera clara y visual.

Teniendo esto en cuenta, se ha realizado un diagrama de clases donde se muestran las clases y métodos necesarios para la correcta implementación.

5.2.1 Diagrama de clases

Tras determinar los indicadores a desarrollar, se ha creado el diagrama de clases de la aplicación *Python*, en el que se han definido las siguientes clases:

- *Analyzer*: el objetivo de esta clase es el de realizar todo el proceso de análisis y cálculo de los indicadores que se desean desarrollar. Por lo tanto, en esta clase están todos los métodos necesarios para realizar dichos cálculos, que serán ejecutados por el método *analyze()*. Este método será el encargado de procesar todo el texto y ejecutar los métodos previamente definidos para que, cuando finalice, devuelva todos los indicadores que hayan sido calculados.
  Algunos de los métodos correspondientes al cómputo de los indicadores hacen uso de la clase *FileLoader*, puesto que ésta tendrá cargados los ficheros de texto correspondientes al listado de conectores, verbos irregulares, palabras por niveles (del A1 al C1) y palabras simples (listado de *Dale-Chall*).

- *Printer*: esta clase se encargará de imprimir por pantalla los resultados generados por la clase *Analyzer* y, además, generará los ficheros .CSV correspondientes a los resultados de cada uno de los ficheros analizados.

- *FileLoader*: esta clase se encargará de cargar los ficheros de texto y, por tanto, extraer el texto que esté en ellos, para que posteriormente la clase *Analyzer* pueda analizarlo. Además, también se cargarán los listados de conectores, verbos irregulares, palabras por niveles y palabras simples.

En la Ilustración 12 puede verse el diagrama de clases completo.

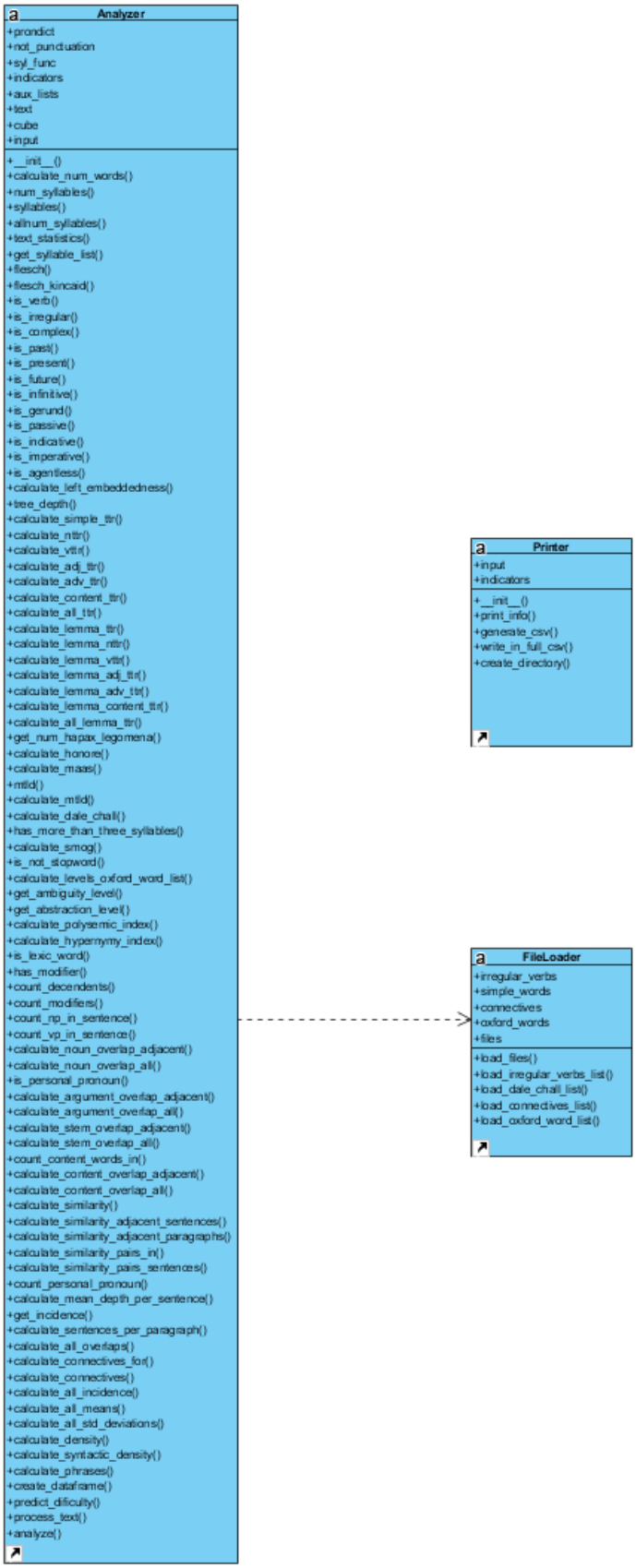


Ilustración 12. Diagrama de clases.

5.3 DISEÑO DE LA APLICACIÓN WEB

Tal y como se ha explicado en apartados anteriores, *AzterTest* puede dividirse en dos sistemas: la aplicación *Python* y la aplicación web, que será la encargada de ejecutarla. Esta aplicación web utiliza *Apache Web Server*. *Apache* es un software gratuito de código abierto que ofrece la posibilidad de crear un servidor web, por lo que hace posible servir contenido en la web.

Apache no es un servidor físico, sino una aplicación que se ejecuta en segundo plano en un servidor. Su principal tarea es establecer una conexión entre el servidor y los navegadores que deseen visitar el sitio web. El servidor y el cliente (los navegadores), se comunican mediante el protocolo HTTP, por lo que, cuando el cliente desee acceder a un recurso del servidor (por ejemplo, la página principal), éste le enviará una solicitud al servidor, y *Apache* se encargará de devolverle una respuesta al cliente.

Los recursos de la aplicación web se encontrarán en el directorio de *Apache*, dentro del servidor, y tendrá los ficheros HTML y PHP que solicitará el cliente. Además, dentro del mismo directorio, estará la aplicación *Python*, la cual será ejecutada por el código PHP en la parte del servidor y, cuando finalice, se devolverá la respuesta al cliente. Por último, la presentación de los ficheros HTML se define en un documento CSS.

Por otro lado, se han diseñado varios bocetos para cada una de las páginas del sitio web, en lo referente a lo que se desea que sea en cuanto a apariencia y funcionalidades, con la finalidad de tratar de acercarse lo máximo posible a estos bocetos cuando se esté desarrollando la aplicación web.

En la Ilustración 13 puede verse el boceto página principal del sitio web. Dispondrá de dos botones en la parte superior derecha: uno en el que estará la funcionalidad principal de la aplicación ("*Analizar*") y otro en el que se podrá consultar información relativa a la aplicación y los indicadores ("*Más info*"). Además, en la parte central del sitio web, aparecerá la opción de "*Seleccionar*", donde se podrá seleccionar cualquier cantidad de ficheros. Por último, el botón "*Analizar*" ejecutará el proceso de análisis de los ficheros seleccionados mediante el botón "*Seleccionar*". Si el usuario no selecciona ningún fichero, este botón no hará nada (sino que mostrará un mensaje de aviso).

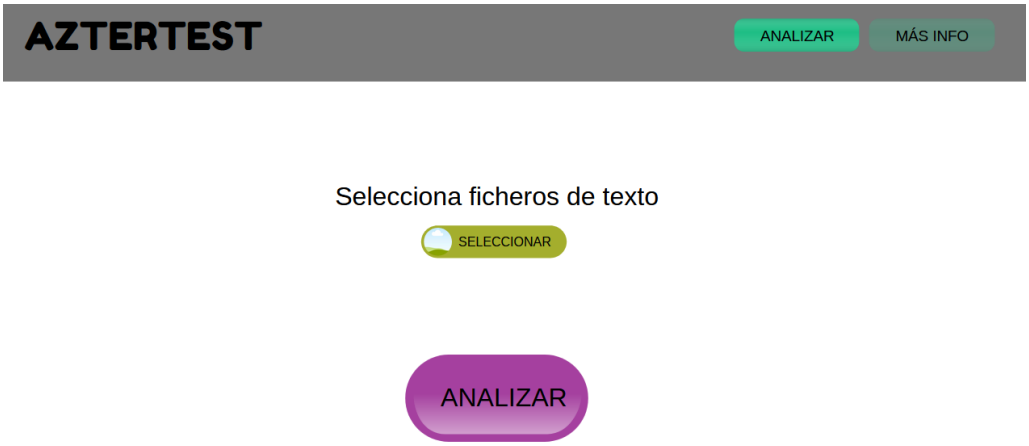


Ilustración 13. Página princpal de AzterTest.

Tal y como se puede ver en la Ilustración 14, cuando el análisis se haya completado, aparecerá la posibilidad de descargar los resultados ("*Descargar resultados*") y un índice con el nombre de todos los ficheros ("*Ir a resultados del fichero X*") que llevará al usuario a la tabla de resultados del fichero que seleccione.

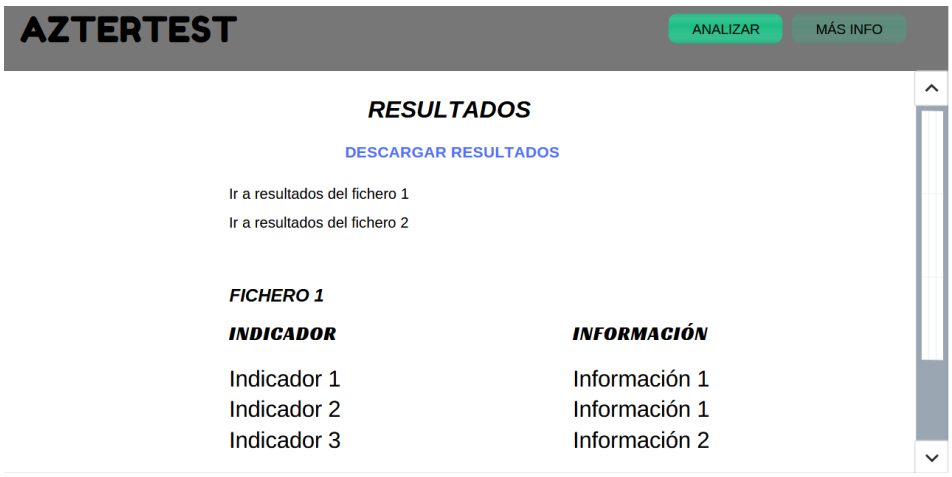


Ilustración 14. Resultados del análisis.

Por último, si el usuario hace clic en "*Más info*", será redireccionado a la página informativa correspondiente, donde podrá ver información acerca de la aplicación y los indicadores que ésta calcula (Ilustración 15).

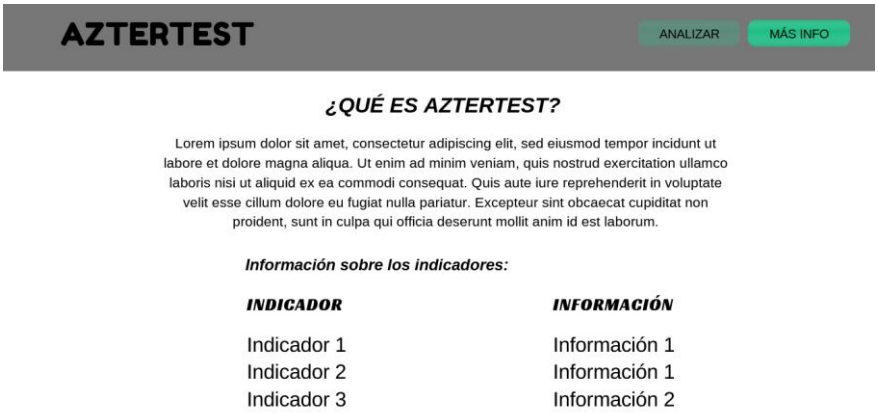


Ilustración 15. Página informativa.

6 Desarrollo

En este apartado, se explicará en detalle qué proceso se ha seguido y cómo se han desarrollado las distintas partes de la aplicación: el programa desarrollado en *Python* y la aplicación web que hará uso de dicho programa.

6.1 LIBRERÍAS DE PROCESAMIENTO DE TEXTO

Con el objetivo de analizar y obtener la información necesaria de los textos, se utilizarán distintas herramientas y recursos. En este apartado, se presentarán las herramientas y librerías más importantes que se utilizarán para desarrollar la aplicación.

6.1.1 Librería NLTK

La librería NLTK (*Natural language toolkit*) de *Python* es la más popular para el procesamiento del lenguaje natural (NLP) y, por ende, tiene una gran comunidad detrás, lo cual puede servir de ayuda en caso de que surjan problemas durante su utilización. Además, es una librería muy sencilla y fácil de utilizar. Todas estas razones la hacen la mejor opción para realizar el procesamiento de texto.

Tokenizador

A grandes rasgos, un tokenizador (*tokenizer* en inglés) divide una cadena de caracteres (*string*) en partes más pequeñas (*substrings*). Estas pequeñas partes se denominan *tokens*. Por lo tanto, *tokenizar* un texto resulta muy útil a la hora de realizar una limpieza del mismo y clasificar los distintos elementos que lo conforman.

La librería NLTK es capaz de *tokenizar* tanto palabras como frases. Para ello, se hace uso de los siguientes métodos:

1) **word_tokenize()** se utiliza para dividir una frase en palabras, lo cual se puede utilizar para detectar, por ejemplo, signos de puntuación. Por tanto, al introducir un texto, este método devolverá una lista de *tokens*, en la que cada *token* será una palabra (o signo de puntuación).
   Para obtener una mejor visión de este método, en la Ilustración 16 puede verse un ejemplo de uso.

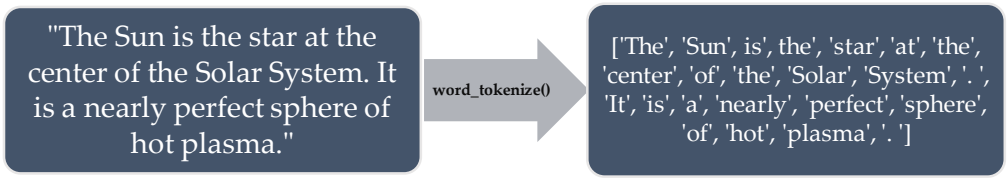


Ilustración 16. Ejemplo de uso para word_tokenize()

2) **sent_tokenize()** se utiliza para dividir un texto en frases. Esto se puede utilizar, por ejemplo, para detectar el número de frases en un texto, o para calcular la media del número de palabras por frase. Un ejemplo de uso puede verse en la Ilustración 17.

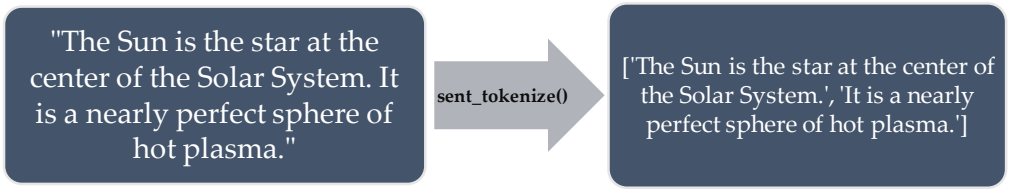


Ilustración 17. Ejemplo de uso para sent_tokenize()

CMUdict

CMUdict (*Carnegie Mellon University Pronouncing Dictionary*) es un diccionario de pronunciación para el inglés norteamericano que contiene más de 134000 palabras y sus pronunciaciones, partiendo las palabras en fonemas. Este diccionario se puede utilizar para generar representaciones de reconocimiento y síntesis de voz (Carnegie Mellon University, s.f.) pero, en el caso de este proyecto, se utilizará para contar el número de sílabas en un texto.

WordNet

WordNet es un lexicón en línea que está inspirado en teorías psicolingüísticas actuales que se basan en las representaciones humanas del léxico. Los sustantivos, verbos, adjetivos y adverbios del inglés están organizados en campo semánticos. Muchos de los conjuntos de palabras son funcionalmente sinónimos, puesto que tienen el mismo o un significado muy similar (Graesser, McNamara, Louwerse, & Cai, 2004). En este proyecto, se utilizará este lexicón para hallar los hiperónimos y los significados de las palabras.

6.1.2 Procesador NLP

Actualmente, los procesadores de *Natural Language Processing* (NLP) son capaces de *tokenizar* (o, dicho de otra manera, segmentar) el texto, de manera que no es necesario utilizar un *tokenizador* externo, aunque, como se ha visto anteriormente, este puede ser útil para ciertas tareas. También disponen de *lematizadores*, que extraen la raíz (*lema*) de las palabras, y etiquetadores de *Parts of Speech* (POS) que, tal y como su nombre indica, etiquetan las palabras.

El procesador NLP que se va a utilizar en este proyecto es *NLP-Cube*. Este procesador, escrito en *Python* y basado por completo en redes neuronales construidas en *DyNET*, toma un fichero de texto plano como entrada y genera un fichero en formato CoNLL-U. Este formato es el que utiliza *Universal Dependencies* (UD), el cual es un proyecto que desarrolla un *treebank* (*corpus* lingüístico en el que cada frase ha sido *parseada*, es decir, anotada con su estructura sintáctica) lingüísticamente consistente para varios lenguajes (Boros, Dumitrescu, & Burtica, 2018).

A continuación, se procede a presentar la estructura que dispone el fichero CoNLL-U generado por *NLP-Cube*, en el que, por cada *token* generado, se generan los campos mostrados en la Tabla 33.

Campo	Descripción
ID	Índice de la palabra o *token* (decir *token* es más correcto puesto que no todos los elementos de la frase son palabras), comenzando en 1 por cada nueva frase.
FORM	La forma de la palabra o marca de puntuación. Por ejemplo, la forma de la palabra *house* sería *house*.
LEMMA	Lema de la palabra. Por ejemplo, el lema de la palabra *is* sería *be*.
UPOS	Etiqueta universal de la categoría gramatical del *token*. [6]
XPOS	Etiqueta específica (para el idioma) de la categoría gramatical. Se muestra un guion bajo en caso de no estar disponible.
FEATS	Listado de las características morfológicas. [7]
HEAD	La cabeza (*head*) de la actual palabra/*token*, que puede ser el valor de un ID o cero.
DEPREL	Relación de dependencia universal con la cabeza (HEAD) de la frase [8]. Si HEAD = 0, el valor de este campo será *root*.
DEPS	Grafo de dependencias mejorado en forma de una lista de pares cabeza-relación (HEAD-DEPREL)
MISC	Cualquier otra anotación.

[6] Es posible consultar estas etiquetas de UPOS en: https://universaldependencies.org/u/pos/index.html
[7] Es posible consultar el listado de estas características en: https://universaldependencies.org/u/feat/index.html
[8] Es posible consultar las relaciones de dependencia en. https://universaldependencies.org/u/dep/index.html

Tabla 33. Campos del formato CoNLL-U.

Como se puede observar, una de las partes más interesantes es el etiquetador de POS que utiliza, puesto que etiqueta las palabras en función de su POS (*Part-of-Speech*), de manera que es posible conocer la categoría gramatical de las palabras. Además, también es posible conocer las características morfológicas y las relaciones de dependencia de las palabras.

Se va a probar la herramienta con un texto de ejemplo con el objetivo de observar y analizar los resultados teniendo en cuenta todo lo explicado anteriormente. El texto de prueba es "*This is a test*" y el resultado obtenido puede verse en la Ilustración 18.

```
1       This    this    PRON    DT      Number=Sing|PronType=Dem        4       nsubj   _
2       is      be      AUX     VBZ     Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin   4       cop     _
3       a       a       DET     DT      Definite=Ind|PronType=Art       4       det     _
4       test    test    NOUN    NN      Number=Sing     0       root    SpaceAfter=No
5       .       .       PUNCT   .       _       4       punct   SpaceAfter=No
```

Ilustración 18. Resultado de la prueba.

Para ver mejor el resultado, se ha insertado en una tabla (Tabla 34).

ID	FORM	LEMMA	UPOS	XPOS	FEATS	HEAD	DEPREL	DEPS
1	This	this	PRON	DT	Number=Sing|PronType=Dem	4	nsubj	_
2	is	be	AUX	VBZ	Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin	4	cop	_
3	a	a	DET	DT	Definite=Ind|PronType=Art	4	det	_
4	test	test	NOUN	NN	Number=Sing	0	root	SpaceAfter=No
5	.	.	PUNCT	.	_	4	punct	SpaceAfter=No

Tabla 34. Tabla del resultado de la prueba.

Como se puede observar, gracias a esta herramienta es posible obtener gran cantidad de información sobre las frases y las palabras que las componen. Por ejemplo, para la prueba que se ha realizado se ha obtenido que:

- *This* es un pronombre (PRON en UPOS) demostrativo (PronType=Dem en FEATS) y está en singular (Number=Sing en FEATS). Además, es un sujeto nominal (nsubj en DEPREL). Su lema (LEMMA) es "this".
- *is* es un verbo auxiliar (AUX en UPOS) que está en modo indicativo (Mood=Ind en FEATS), en singular (Number=Sing en FEATS), en tercera persona del presente (Person=3 y Tense=Pres en FEATS) y es un verbo finito (VerbForm=Fin en FEATS). Además, es un verbo copulativo (cop en DEPREL). Su lema (LEMMA) es "be".
- *a* es un determinante (DET en UPOS) indefinido (Definite=Ind en FEATS). Más concretamente, un artículo (PronType=Art en FEATS).

- *test* es un sustantivo (NOUN en UPOS) que está en singular (Number=Sing en FEATS). Además, es la raíz de la frase (0 en HEAD y root en DEPREL).
- Por último, el punto (.) es una marca de puntuación (PUNCT en UPOS).

6.1.3 Wordfreq

Wordfreq es una librería de *Python* que, en base a varias fuentes de datos, ofrece estimaciones de la frecuencia de las palabras en diversos idiomas. Esta frecuencia se obtiene utilizando el método *word_frequency()* y devuelve un número decimal que se encuentra entre el cero y el uno (Speer, s.f.).

Con el objetivo de obtener la frecuencia de una palabra en una escala más amigable para el ser humano, se utiliza la *zipf_frequency()*, que es una variación de *word_frequency()* que utiliza la escala *Zipf*. Dicha escala es el logaritmo en base 10 del número de veces que aparece la palabra por cada mil millones de palabras. Por ejemplo, si una palabra tiene una frecuencia *Zipf* de 6, significa que aparece 10^6 veces en 10^9 palabras; es decir, una vez por cada 1000 (Speer, s.f.).

6.2 PREPARACIÓN DEL ENTORNO Y LIBRERÍAS

En esta sección, se explica cómo se ha creado el entorno virtual de *Python* con el que se trabaja, además de explicar cómo se han instalado las librerías que son necesarias para implementar la aplicación.

i. Creación del entorno virtual de *Python*

A partir de la versión 3.6 de *Python*, la opción más recomendable para instalar un entorno virtual es haciéndolo mediante el módulo `venv`. Por tanto, se comienza instalando el paquete `python3-venv` mediante el siguiente comando:

```
sudo apt install python3-venv
```

Una vez finalizada la instalación del paquete, en primer lugar, hay que situarse en el directorio en el cual se desee crear el entorno virtual. Dentro de dicho directorio, únicamente hay que ejecutar el siguiente comando para crear el entorno:

```
python3 -m venv aztertest-env
```

Este comando crea un directorio llamado `aztertest-env` (para este caso) que contiene los archivos binarios de *Python,* el gestor de paquetes *Pip* y la librería estándar de *Python*, además de otros ficheros adicionales.

Ahora se dispone de un entorno virtual de *Python* pero, para poder utilizarlo, hay que activarlo. Para ello, dentro del mismo directorio del entorno, habrá que ejecutar el script *activate* que se encuentra en el directorio */bin.* Por tanto, simplemente habrá que ejecutar el siguiente comando:

```
source bin/activate
```

Tras ejecutarlo, en la terminal se podrá ver indicado que el entorno virtual `aztertest-env` ya está activado y listo para ser utilizado:

```
(aztertest-env) $
```

ii. Instalación de las librerías necesarias

Una vez se disponga de un entorno virtual de *Python,* se puede proceder a la instalación de las librerías necesarias para el desarrollo de la aplicación. Antes de eso, habrá que asegurarse de tener el administrador de paquetes *Pip* de *Python 3.* Para ello, se utiliza el siguiente comando:

```
sudo apt install python3-pip
```

En primer lugar, hay que asegurarse de que, en la terminal de Ubuntu desde la que se vaya a trabajar, se encuentra activado el entorno virtual de *Python* y, en caso de no estarlo, activarlo mediante la ejecución del script *activate,* tal y como se especifica en la sección anterior.

Después, para evitar que surjan posibles problemas durante la instalación de los paquetes, habrá que asegurarse de que *Pip* está actualizado. Para comprobarlo, se ejecuta el siguiente comando:

```
pip3 install --upgrade pip
```

Una vez se haya actualizado *Pip*, se comienzan a instalar las siguientes librerías:

- *NumPY*: esta librería diseñada para realizar computación científica y, en este proyecto, será utilizada principalmente para obtener la media y la

desviación típica de *arrays* numéricos (aunque también es un módulo necesario para trabajar con otras librerías como *Skikit-learn*). Para instalar este módulo, se utiliza el siguiente comando:

```
pip3 install numpy
```

- *NTLK*: tal y como se ha explicado anteriormente, se utiliza esta librería de procesamiento del lenguaje natural. Para instalarla, es necesario introducir el siguiente comando:

```
pip3 install nltk
```

- *Argparse*: se trata de un módulo de análisis de línea de comandos que ofrece la posibilidad de crear argumentos para el programa de manera sencilla. La instalación se realiza de la siguiente manera:

```
pip3 install argparse
```

- *TensorFlow*: es una librería de código abierto para aprendizaje automático y, en este proyecto, se utilizará (junto con el módulo *TensorFlow Hub*) para calcular la similitud entre dos textos mediante el *Universal Sentence Encoder*. Los comandos a introducir son:

```
pip3 install tensorflow
pip3 install tensorflow_hub
```

- *Pandas*: se trata de una librería de código abierto que provee la posibilidad de manipular y analizar grandes cantidades de datos. Se utilizará, principalmente, para manipular los datos que serán utilizados para crear el clasificador. Para instalarlo, se debe introducir el siguiente comando:

```
pip3 install pandas
```

- *Scikit-learn*: es un módulo que provee múltiples herramientas de *machine learning* (Pedregosa, y otros, 2011). Por ello, será utilizado para crear el clasificador. El comando para instalarlo es:

```
pip3 install scikit-learn
```

- *NLP-Cube*: esta librería de procesamiento de texto, que ha sido presentada anteriormente, será utilizada durante la mayor parte del proyecto. Para instalarla, será necesario introducir el siguiente comando:

```
pip3 install nlpcube
```

- *WordFreq*: tal y como se ha explicado en el apartado anterior, se utilizará esta librería para hallar las palabras más raras (en el sentido de su infrecuencia) del texto. Para instalarlo, el comando a introducir es:

```
pip3 install wordfreq
```

- *Textract*: para extraer texto plano de ficheros con formato .odt, .docx y .doc, se hará uso de esta librería, puesto que permite realizar esa misma función de manera simple y efectiva. Para instalarla, se debe introducir el siguiente comando:

```
pip3 install textract
```

6.3 IMPLEMENTACIÓN DE LOS INDICADORES

El análisis se realiza mediante unos indicadores que nos ayudan a identificar y determinar ciertos aspectos importantes sobre la gramática del texto. Estos indicadores se desarrollan en el lenguaje de programación *Python* ya que, como se ha explicado anteriormente, existen multitud de librerías y herramientas que facilitan el procesado del texto y, por ende, la evaluación y el análisis del mismo.

Por lo tanto, en esta sección se explicará de forma detallada cómo se ha implementado cada uno de esos indicadores, incluyendo los problemas que se han encontrado durante el desarrollo y cómo se solucionaron.

6.3.1 Implementación de los indicadores generales

En este apartado, se mostrará cómo han sido implementados los indicadores generales (es decir, aquellos que muestran información superficial del texto).

Número de palabras

Para calcular el número de palabras, se utiliza la función *filter()*. Esta función, que toma como parámetro una función condicional y un iterable, es capaz de devolver una nueva colección con los elementos filtrados que cumplan la condición establecida en dicha función condicional. En este caso, para establecer la función condicional se ha utilizado una función *lambda*, puesto que es una forma sencilla de crear funciones anónimas en una sola línea de código.

Consideramos palabras a todos aquellos *tokens* que no sean signos de puntuación (es decir, que la longitud del *token* no sea uno) o que estén compuestos por caracteres alfabéticos. Esta condición se tiene en cuenta en la función *lambda,* en la que se comprueba si una cadena de caracteres esté compuesta únicamente por caracteres alfabéticos, utilizando el método *isalpha()* de *Python*, y se comprueba la longitud del *token* mediante la función *len()* de *Python* (Ilustración 19).

```
not_punctuation = lambda w: not (len(w) == 1 and (not w.isalpha()))
```

Ilustración 19. Función lambda.

Por último, se filtran los *tokens* (que se obtienen mediante el uso del *tokenizador* y, concretamente, la función *word_tokenize()*) utilizando la función *filter()* previamente mencionada (Ilustración 20).

```
filterwords = filter(not_punctuation, word_tokenize(self.text))
```

Ilustración 20. Función filter().

Por último, se cuenta cada elemento de la colección *filterwords* retornada por la función *filter()* y, de esta manera, se obtiene el indicador *num_words* (es decir, el número de palabras).

Número de palabras distintas

El número de palabras distintas se calcula recorriendo la lista de *tokens* (también obtenida mediante *word_tokenize()*) y realizando la misma comprobación que con el indicador del *Número de palabras*, pero sin tener en cuenta las palabras repetidas. Para ignorar las palabras repetidas, se van introduciendo las palabras en una lista y, si una palabra ya se encuentra en esa lista, no será introducida (de manera que en esa lista únicamente habrá palabras distintas). Por lo tanto, el indicador de palabras distintas se calcula obteniendo la longitud de la lista de palabras distintas.

Número de palabras y signos de puntuación

Este indicador se calcula obteniendo el número total de *tokens* puesto que, como hay que tener en cuenta también los signos de puntuación, no es necesario realizar ningún tipo de filtrado.

Número de párrafos

Un párrafo termina con el carácter de nueva línea (que se representa con *\n*) ya que indica el final de una línea de texto y da paso a la siguiente. Teniendo esto en cuenta, es posible obtener el número de párrafos.

Por lo tanto, según se realiza la lectura del fichero de texto, se reemplaza el carácter *\n* por el carácter @ para que sea más sencilla su segmentación y, por último, se obtiene el número de líneas (que, en este caso, serán párrafos) mediante la función s*plit()*, que divide un *String* en una lista según el valor pasado como parámetro (Ilustración 21).

```
with open(self.input, encoding='utf-8') as f:
    text2 = f.read().replace('\n', '@')
lines = text2.split('@')
```

Ilustración 21. Obtención de párrafos.

Por otro lado, es necesario realizar una pequeña limpieza, ya que es posible que existan líneas (párrafos) vacíos. Por ello, se utiliza la función *strip()*, que elimina los caracteres del inicio y del final de un *String*, para comprobar que el párrafo no esté vacío y, una vez comprobado que no lo está, se introduce en una nueva lista llamada *paragraphs* (Ilustración 21).

```
for line in lines:
    if not line.strip() == '':
        paragraphs.append(line)
```

Ilustración 22. Limpieza de párrafos.

Por último, se obtiene el indicador mediante la obtención de la longitud de la lista *paragraphs*, utilizando el método *len()* de *Python*.

Número de frases

El número total de frases se puede calcular mediante el uso de un *tokenizador*. En este caso, se utiliza el *tokenizador* de *NLP-Cube* para dividir el texto en frases. Este *tokenizador* utiliza un algoritmo no supervisado para construir un modelo para abreviaturas, colocaciones y palabras que empiezan frases. Este modelo es entrenado con una larga colección de texto plano (NLTK, s.f.).

Por ello, tan solo hay que obtener la longitud del listado de *sequences* (o frases) que devuelve *NLP-Cube* mediante su método *cube()* (Ilustración 23).

```
sequences = self.cube(text)
```

Ilustración 23. Obtención de frases mediante NLP-Cube.

Número de frases en un párrafo

Tal y como se ha visto anteriormente, la media del número de frases en un párrafo se calcula de la siguiente manera:

$$Frases\ en\ un\ párrafo_{Media} = \frac{Número\ total\ de\ frases}{Número\ total\ de\ párrafos}$$

En primer lugar, se recorre cada párrafo y, en una lista, se introduce la longitud de dicho párrafo (la longitud equivale al número de frases en dicho párrafo), tal y como puede verse en la Ilustración 24.

```
self.aux_lists['sentences_per_paragraph'].append(len(sequences))
```

Ilustración 24. Frases por párrafo.

Mediante esa lista, es posible obtener tanto la media como la desviación típica de número de frases en un párrafo. Para ello, se utilizan los métodos *mean()* y *std()* (las cuales reciben como parámetro una lista de valores numéricos) ofrecidos por la librería *NumPy* de *Python.*

Número de palabras en una frase

Para calcular este indicador, se realiza la media del número total de palabras entre el número total de frases y el proceso a seguir para realizar dicho cálculo es el mismo que para el indicador anterior: introducir en una lista el número de palabras en cada frase (Ilustración 25) y utilizar la librería *NumPy* para obtener la media y la desviación típica.

```
self.aux_lists['sentences_length_list'].append(num_words_in_sentences)
```

Ilustración 25. Lista de número de palabras en cada frase.

Número de palabras en una frase (sin tener en cuenta stopwords)

Se realiza el mismo proceso que con el indicador *Número de palabras en una frase* pero, en este caso, el número de palabras en una frase pasa por un filtro que comprueba si la palabra es una *stopword* (una *stopword* es una palabra vacía o sin significado como los artículos, pronombres, preposiciones, etc.), tal y como puede verse en la Ilustración 26.

```
if self.is_not_stopword(entry.word):
    num_words_in_sentence_without_stopwords += 1
```

Ilustración 26. Comprobación de stopwords en una frase.

El corpus de NLTK dispone de un listado de *stopwords*, el cual es utilizado en la función *is_not_stopword()* para determinar si la palabra pasada como parámetro es una *stopword* o no (Ilustración 27).

```
def is_not_stopword(self, word):
    stop_words = stopwords.words('english')
    return word.lower() not in stop_words
```

Ilustración 27. Método is_not_stopword().

Número de sílabas en una palabra

De nuevo, para obtener la media y desviación típica del número de sílabas en una palabra, se crea una lista de sílabas por palabra y se utiliza la librería *NumPy* para realizar el correspondiente cálculo.

Para obtener el número de sílabas que tiene una palabra, se utiliza el diccionario de pronunciación *CMUDict* pero, como puede ocurrir que una palabra no se encuentre en dicho diccionario, se utiliza un algoritmo menos eficiente. Por lo tanto, si una palabra no se encuentra en el diccionario *CMUDict*, se ejecutará ese otro algoritmo, tal y como puede ver en la Ilustración 28.

```
def allnum_syllables(self, word):
    try:
        return self.num_syllables(word)
    except KeyError:
        # if word not found in cmudict
        return self.syllables(word)
```

Ilustración 28. Cálculo del número de sílabas en una palabra.

Para entender la función *num_syllables()* hay que tener claros algunos conceptos de *CMUDict*:

- El diccionario divide las palabras en fonemas (que son más cortos que las sílabas). Por ejemplo, la palabra *cat* se divide en tres fonemas: K - AE - T.
- Las vocales tienen un marcador de estrés (*stress marker*) que es 0, 1 o 2 dependiendo de la pronunciación de la palabra (en el AE de *cat*, sería AE1).

Por lo tanto, en esta función se cuenta el número de *stress markers* para las vocales para, finalmente, obtener el número de sílabas (Ilustración 29).

```
def num_syllables(self, word):
    list = []
    max = 0
    for x in self.prondict[word.lower()]:
        tmp_list = []
        tmp_max = 0
        for y in x:
            if y[-1].isdigit():
                tmp_max += 1
                tmp_list.append(y)
        list.append(tmp_list)
        if tmp_max > max:
            max = tmp_max
    return (max)
```

Ilustración 29. Número de sílabas utilizando CMUDict.

Por otro lado, la función *syllables()* utiliza un algoritmo menos eficiente y que utiliza identificadores silábicos comunes para contar el número de sílabas (Ilustración 30).

```
if word[0] in vowels:
    count += 1
for index in range(1, len(word)):
    if word[index] in vowels and word[index - 1] not in vowels:
        count += 1
if word.endswith('e'):
    count -= 1
if word.endswith('le') or word.endswith('a'):
    count += 1
if count == 0:
    count += 1
if "ooo" in word or "mm" in word:
    count = 1
if word == 'll':
    count = 0
if (word.startswith('x') and len(word) >= 2) and word[1].isdigit():
    count = 0
if word == 'lmfao':
    count = 5
if len(word) < 2 and word not in ['a', 'i', 'y', 'o']:
    count = 0
return count
```

Ilustración 30. Número de sílabas utilizando otro algoritmo.

Número de letras en una palabra

El número de letras de una palabra se calcula obteniendo la longitud del *String* que compone dicha palabra utilizando la función *len()* de *Python*. Después, para poder

obtener la media y la desviación típica, se introducen los valores de la longitud de las palabras en una lista (Ilustración 31) y se calculan mediante las funciones *mean()* y *std()* de la librería *NumPy*.

```
self.aux_lists['words_length_list'].append(len(entry.word))
```

Ilustración 31. Lista de longitud de palabras.

Número de letras en una palabra (sin tener en cuenta stopwords)

Este indicador se calcula de igual manera que el anterior, pero en este caso no se tienen en cuenta las *stopwords*, por lo que se utiliza la función *is_not_stopword()* que se utiliza para calcular el indicador de *Número de palabras en una frase (sin tener en cuenta stopwords)* (Ilustración 32).

```
if self.is_not_stopword(entry.word):
    self.aux_lists['words_length_no_stopwords_list'].append(len(entry.word))
```

Ilustración 32. Lista de longitud de palabras sin stopwords.

Número de letras en un lema

Para calcular este indicador, se sigue el mismo proceso que en los anteriores indicadores, pero, a la hora de obtener la longitud, se utiliza el lema de la palabra en vez de la palabra en sí.

Como se ha explicado en el apartado 6.1, NLP-Cube es capaz de devolver el lema de las palabras, por lo que para obtenerlo simplemente es necesario hacer *entry.lemma*, siendo *entry* el *token* o la palabra en cuestión (Ilustración 33).

```
self.aux_lists['lemmas_length_list'].append(len(entry.lemma))
```

Ilustración 33. Lista de longitud del lema de las palabras.

6.3.2 Implementación de los indicadores de riqueza léxica

En este apartado, se mostrará cómo han sido implementados los indicadores de riqueza o diversidad léxica.

Densidad léxica

Como se ha visto anteriormente, la densidad léxica se calcula de la siguiente manera (Johansson, 2008):

$$Densidad\ léxica = \frac{Total\ de\ palabras\ de\ contenido}{Total\ de\ palabras}$$

Concretamente, se calculan los siguientes indicadores, tal y como se puede observar en la Ilustración 34:

- Densidad léxica
- Densidad de sustantivos
- Densidad de verbos
- Densidad de adjetivos
- Densidad de adverbios

```
def calculate_density(self):
    i = self.indicators
    i['lexical_density'] = round(i['num_lexic_words'] / i['num_words'], 4)
    i['noun_density'] = round(i['num_noun'] / i['num_words'], 4)
    i['verb_density'] = round(i['num_verb'] / i['num_words'], 4)
    i['adj_density'] = round(i['num_adj'] / i['num_words'], 4)
    i['adv_density'] = round(i['num_adv'] / i['num_words'], 4)
```

Ilustración 34. Calcular densidad.

Para obtener el total de palabras léxicas o de contenido, se ha utilizado el campo UPOS que genera *NLP-Cube*, de manera que:

- Si el valor del UPOS es "NOUN", el *token*/palabra es un sustantivo.
- Si el valor del UPOS es "ADJ", el *token*/palabra es un adjetivo.
- Si el valor del UPOS es "ADV", el *token*/palabra es un adverbio.
- Si el valor del UPOS es "VERB", o es "AUX" y además su padre (HEAD) no es "VERB", el *token*/palabra es un verbo. Esta comprobación se realiza mediante el método *is_verb()* (Ilustración 35).

```
def is_verb(self, word, frase):
    return word.upos == 'VERB' or (word.upos == 'AUX' and frase[word.head - 1].upos != 'VERB')
```

Ilustración 35. Método is_verb

TTR (Type-Token Ratio)

El TTR, tal y como se ha explicado en el apartado 5.1, tiene en cuenta los tipos (palabras únicas) y las instancias de cada palabra, por lo que es necesario realizar una lista de palabras distintas para cada uno de los TTR.

Los TTR que se han calculado son:

a. Simple Type-Token Ratio
b. Content Type-Token Ratio
c. Noun Type-Token Ratio
d. Verb Type-Token Ratio
e. Adjective Type-Token Ratio
f. Adverb Type-Token Ratio
g. Type-Token Ratio con lemas

Por lo tanto, se han realizado los siguientes listados de palabras distintas y para comprobar si una palabra es sustantivo, verbo, adjetivo o adverbio, se ha utilizado el campo UPOS, tal y como se hace para obtener la densidad léxica.

Honoré

Tal y como se ha mostrado en el apartado 5.1, el índice propuesto por Honoré se mide con la siguiente fórmula (Capsada Blanch & Torruella Casañas, 2017):

$$Honoré = 100 \cdot \frac{\log N}{1 - \frac{V_1}{V}}$$

N → Número de palabras en el texto
V1 → Número de *hápax legonema*
V → Número de tipos

Como se puede observar, esta métrica se calcula a partir del número de *hapax legomena* que contiene un texto. Un *hapax legomena* es cualquier palabra que únicamente aparece una sola vez en el texto. Para poder conocer cuántos *hapax legonema* tiene un texto, se utiliza un diccionario de *Python* en el que cada tipo de palabra (*type*) le corresponde un valor en el que se anota cuántas veces aparece dicha palabra en el texto. Después, se cuentan las palabras que tienen una sola aparición en el texto (Ilustración 36).

```
def get_num_hapax_legomena(self):
    num_hapax_legonema = 0
    for word, frecuencia in self.words_freq.items():
        if frecuencia == 1:
            num_hapax_legonema += 1
    return num_hapax_legonema
```

Ilustración 36. Método para obtener el hapax legonema.

Tras obtener el *hapax legonema*, tan solo hay que utilizar la fórmula previamente establecida (Ilustración 37).

```
def calculate_honore(self):
    n = self.indicators['num_words']
    v = len(self.aux_lists['different_forms'])
    v1 = self.get_num_hapax_legomena()
    self.indicators['honore'] = round(100 * ((np.log10(n)) / (1 - (v1 / v))), 4)
```

Ilustración 37. Calcular Honoré.

Maas

Anteriormente se ha presentado la fórmula correspondiente a la métrica de Maas, que es (Capsada Blanch & Torruella Casañas, 2017):

$$Maas = \frac{\log N \cdot \log V}{\log^2 \mathrm{V}}$$

N → Número de palabras en el texto
V → Número de tipos

Disponemos del número de palabras en el texto y el número de tipos (palabras distintas), por lo que tan solo hay aplicar la fórmula (Ilustración 38).

```
def calculate_maas(self):
    n = self.indicators['num_words']
    v = len(self.aux_lists['different_forms'])
    self.indicators['maas'] = round((np.log10(n) - np.log10(v)) / (np.log10(v) ** 2), 4)
```

Ilustración 38. Calcular Maas.

Como se puede observar, para calcular el logaritmo se utiliza el método *log10()* de la librería *NumPY*.

MTLD

En el apartado 5.1, se presentó el proceso que ha de seguirse para calcular el MTLD de un texto. En este apartado, se va a seguir el mismo proceso acompañándolo de su implementación.

Secuencialmente, se crean segmentos (aumentando el número de palabras y añadiendo la palabra a la lista de palabras distintas en caso de no estarlo) y se calcula su TTR (Ilustración 39) y, si se cumple la condición de que el TTR sea igual o menor a

0.72, se aumenta el valor del número de fragmentos y se resetean el resto de variables, incluida la lista de palabras distintas (Ilustración 40).

```
word_count += 1
if word not in dif_words:
    dif_words.append(word)
ttr = self.calculate_simple_ttr(dif_words, word_count)
```

Ilustración 39. Cálculo del TTR en un segmento.

```
fragments += 1
word_count = 0
dif_words.clear()
ttr = 1.0
```

Ilustración 40. Aumento de fragmentos y reseteo de variables.

Al llegar al final del texto, si se da el caso de que queda un segmento sin llegar a alcanzar el umbral, este segmento no se desprecia, sino que se obtiene un número residual menor que uno. Este valor residual es la proporción de la cantidad que le falta a la TTR de este segmento para llegar a uno. Este valor se suma al número de segmentos completos (Capsada Blanch & Torruella Casañas, 2017). Esto puede verse en la Ilustración 41.

```
residual = (1.0 - ttr) / (1.0 - ttr_threshold)
fragments += residual
```

Ilustración 41. Cálculo del valor residual.

El proceso termina realizando el cálculo del MTLD de la siguiente manera:

$$MTLD = \frac{N}{n}$$

N → Número de palabras en el texto
n → Número de segmentos

Esto es en caso de que el número de segmentos sea mayor que cero, tal y como puede verse en la Ilustración 42.

```
if fragments != 0:
    return len(filtered_words) / fragments
else:
    return 0
```

Ilustración 42. Cálculo del MTLD.

Para finalizar, como se explicó anteriormente, el MTLD se calcula tanto en el orden de lectura como en el orden inverso, de manera que el MTLD se calcula tal y como puede verse en la Ilustración 43.

```
self.indicators['mtld'] = round((self.mtld(filtered_words) + self.mtld(filtered_words[::-1])) / 2, 4)
```

Ilustración 43. Cálculo del MTLD final.

6.3.3 Implementación de los indicadores de lecturabilidad

Existen varias fórmulas que, en base a la longitud de las palabras y frases, proveen indicaciones de la lecturabilidad de un texto. Calculando la lecturabilidad de un texto, se puede obtener una idea de su complejidad. A continuación, se presentan las fórmulas de lecturabilidad más utilizadas:

Flesch Reading Ease

El valor de esta métrica se calcula utilizando la fórmula mostrada en el apartado 5.1 (Graesser, McNamara, Louwerse, & Cai, 2004):

$$Flesch\ Reading\ Ease = 206.835 - 1.015 \cdot \text{ASL} - 84.6 \cdot \text{ASW}$$

ASL → Longitud media de las frases del texto
ASW → Media de sílabas por palabra

Puesto que ya se disponía de la media de sílabas y de longitud de las frases (previamente calculadas), para calcular este indicador simplemente se ha aplicado dicha fórmula en el método **`def flesch(self)`**.

Flesch-Kincaid Grade Level

Por otro lado, está la fórmula de *Flesch-Kincaid Grade Level*, cuya fórmula se ha mostrado anteriormente y es la siguiente:

$$Flesch\text{–}Kincaid\ Grade\ Level = 0.39 \cdot \text{ASL} + 11.8 \cdot \text{ASW} - 15.49$$

ASL → Longitud media de las frases del texto
ASW → Media de sílabas por palabra

De la misma manera, para esta métrica únicamente ha sido necesario aplicar dicha fórmula en el método **`def flesch_kincaid(self)`**.

Dale-Chall readability formula

Para calcular este indicador, es necesario obtener el número de palabras complejas. Para determinar qué palabras son complejas, se utiliza una lista de palabras comunes[9], de manera que, para contar el número de palabras consideradas complejas, se comprueba si la palabra está o no en dicha lista mediante el método *is_complex()* (Ilustración 44). Se consideran palabras complejas a todas aquellas que no sean simples.

```
def is_complex(self, word):
    return False if word.word.lower() in FileLoader.simple_words or word.lemma.lower() in FileLoader.simple_words else True
```

Ilustración 44. Método is_complex()

Después, tras haber calculado el número de palabras complejas, se procede a calcular el valor de Dale-Chall mediante la correspondiente fórmula, tal y como se puede ver en la Ilustración 45.

```
def calculate_dale_chall(self):
    ts = self.indicators['num_sentences']
    tc = self.indicators['num_complex_words']
    tw = self.indicators['num_words']
    percentage = (tc/tw) * 100
    if percentage >= 5.0:
        self.indicators['dale_chall'] = round(0.1579 * percentage + 0.0496 * (tw / ts) + 3.6365, 4)
    else:
        self.indicators['dale_chall'] = round(0.1579 * percentage + 0.0496 * (tw / ts), 4)
```

Ilustración 45. Fórmula de Dale-Chall.

Simple Measure Of (SMOG) grade

Para realizar el cálculo de esta métrica, es necesario calcular el número de palabras con más de tres sílabas. Para ello, se hace uso del método *has_more_than_three_syllables()* (Ilustración 46).

[9] Listado obtenido de: https://readable.com/blog/the-dale-chall-word-list/

```
def has_more_than_three_syllables(self, word):
    num_syl = 0
    try:
        num_syl = self.num_syllables(word)
    except KeyError:
        # if word not found in cmudict
        num_syl = self.syllables(word)
    return True if num_syl > 3 else False
```

Ilustración 46. Método has_more_than_three_syllables()

Tras haber calculado el número de palabras con más de tres sílabas, se puede aplicar la fórmula de SMOG, tal y como se puede observar en la Ilustración 47.

```
def calculate_smog(self):
    ts = self.indicators['num_sentences']
    tps = self.indicators['num_words_more_3_syl']
    self.indicators['smog'] = round(1.0430*math.sqrt(30*tps/ts)+3.1291, 4)
```

Ilustración 47. Fórmula de SMOG.

6.3.4 Implementación de los indicadores semánticos de lecturabilidad

En este apartado, se mostrará cómo han sido implementados los indicadores que comprueban la lecturabilidad de un texto según la semántica del mismo.

Media de los valores de polisemia

Utilizando WordNet, es posible encontrar los distintos significados de una palabra. Para ello, se utiliza el método *synsets()* que proporciona NLTK. Este método, cuyo argumento es la propia palabra, devuelve un listado de los distintos significados que esa palabra puede tener. Por lo tanto, como para calcular este indicador tan solo interesa conocer el número de significados, se utilizará la función *len()* de *Python*, para obtener la longitud de la lista de significado y, por tanto, el número de significados de la palabra.

Por cada palabra, se introducirá en un listado que se ha denominado *ambiguity_list* la cantidad de significados que esa palabra posee. Después, cuando se haya terminado de recorrer todas las palabras, se calculará la media de los valores que se encuentren en dicha lista mediante el método *mean()* de *NumPy* (Ilustración 48).

```
def calculate_polysemic_index(self, ambiguity_list):
    i = self.indicators
    i['polysemic_index'] = round(float(np.mean(ambiguity_list)), 4)
```

Ilustración 48. Método calculate_polysemic_index()

Media de los valores de hiperonimia

De la misma manera, se utiliza WordNet para encontrar el número de hiperónimos de cada palabra. Para ello, se utiliza el método *hypernym_paths()*, que devuelve un listado con todos los hiperónimos de una palabra. Por lo tanto, para cada palabra, se guarda en un listado el número de hiperónimos (obtenido mediante la función *len()* de *Python*) de cada palabra y, utilizando el método *mean()* de *NumPy*, se obtiene la media de los valores de hiperonimia para verbos, sustantivos y una combinación de ambos (Ilustración 49).

```
def calculate_hypernymy_index(self, ambiguity_content_words_list, FLAG = 'VN'):
    i = self.indicators
    if FLAG == 'VN':
        i['hypernymy_index'] = round(float(np.mean(ambiguity_content_words_list)), 4)
    elif FLAG == 'V':
        i['hypernymy_verbs_index'] = round(float(np.mean(ambiguity_content_words_list)), 4)
    elif FLAG == 'N':
        i['hypernymy_nouns_index'] = round(float(np.mean(ambiguity_content_words_list)), 4)
```

Ilustración 49. Método calculate_hypernymy_index()

6.3.5 Implementación de los indicadores morfológicos

Para implementar estos indicadores, es necesario acceder a las características morfológicas de cada entrada (*token*) del texto. Dichas características, tal y como se explicó en el apartado 6.1, se extraen gracias a la utilización de NLP-Cube.

Número de verbos en tiempo pasado

Para saber si un verbo está en tiempo pasado, el valor *Tense=Past* debe estar entre las características morfológicas de la palabra, por lo que se ha desarrollado el método *is_past()* que comprueba si la palabra pasada como parámetro está en tiempo pasado o no (Ilustración 50). En caso de estarlo, se aumenta en uno el número de verbos en tiempo pasado.

```
def is_past(self, word):
    atributos = word.attrs.split('|')
    return True if 'Tense=Past' in atributos else False
```

Ilustración 50. Método is_past()

Número de verbos en tiempo presente

Para saber si un verbo está en tiempo presente, el valor *Tense=Pres* debe estar entre las características morfológicas de la palabra, por lo que se ha desarrollado el método *is_present()* que comprueba si la palabra pasada como parámetro está en tiempo presente o no (Ilustración 51). En caso de estarlo, se aumenta en uno el número de verbos en tiempo presente.

```
def is_present(self, word):
    atributos = word.attrs.split('|')
    return True if 'Tense=Pres' in atributos else False
```

Ilustración 51. Método is_present()

Número de verbos en tiempo futuro

En este caso, pese a que la primera intención fue la de comprobar que el valor *Tense=Fut* (el cual correspondería al tiempo futuro), se tuvo que desarrollar otra forma de realizar el cálculo puesto que, tras realizar pruebas preliminares, se comprobó que dicho valor (*Tense=Fut*) nunca llegaba a darse.

La otra forma que se ha desarrollado es la de comprobar los verbos en futuro de manera "manual". Es decir, comprobar que existen los verbos "will" y "shall" seguido de una forma básica de un verbo (por ejemplo, "will be" o "shall go").

Para ello, se mira si el lema de la palabra pasada como parámetro es "will" o "shall" y si el XPOS de la siguiente palabra es VB (es decir, una forma verbal básica) (Ilustración 52).

```
def is_future(self, word, frase):
    return word.upos == 'AUX' and word.lemma in ['will', 'shall'] and frase[word.head - 1].xpos == 'VB'
```

Ilustración 52. Método is_future()

Número de verbos en modo indicativo

Para saber si un verbo está en modo indicativo, el valor *Mood=Ind* debe estar entre las características morfológicas de la palabra, por lo que se ha desarrollado el método

is_indicative() que comprueba si la palabra pasada como parámetro está en indicativo o no (Ilustración 53). En caso de estarlo, se aumenta en uno el número de verbos en modo indicativo.

```
def is_indicative(self, word):
    atributos = word.attrs.split('|')
    return True if 'Mood=Ind' in atributos else False
```

Ilustración 53. Método is_indicative()

Número de verbos en modo imperativo

Para saber si un verbo está en modo imperativo, el valor *Mood=Imp* debe estar entre las características morfológicas de la palabra, por lo que se ha desarrollado el método *is_imperative()* que comprueba si la palabra pasada como parámetro está en modo imperativo o no (Ilustración 54). En caso de estarlo, se aumenta en uno el número de verbos en modo imperativo.

```
def is_imperative(self, word):
    atributos = word.attrs.split('|')
    return True if 'Mood=Imp' in atributos else False
```

Ilustración 54. Método is_imperative()

Número de verbos en modo subjuntivo

Para saber si un verbo está en modo subjuntivo, el valor *Mood=Sub* debe estar entre las características morfológicas de la palabra, por lo que se ha desarrollado el método *is_subjunctive()* que comprueba si la palabra pasada como parámetro está en modo subjuntivo o no (Ilustración 55). En caso de estarlo, se aumenta en uno el número de verbos en modo subjuntivo.

```
def is_subjunctive(self, word):
    atributos = word.attrs.split('|')
    return True if 'Mood=Sub' in atributos else False
```

Ilustración 55. Método is_subjunctive()

Número de verbos en voz pasiva

Para saber si un verbo es pasivo, el valor *Voice=Pass* debe estar entre las características morfológicas de la palabra, por lo que se ha desarrollado el método *is_passive()* que comprueba si el verbo pasado como parámetro es pasivo o no

(Ilustración 56). En caso de serlo, se aumenta en uno el número de verbos en voz pasiva.

```
def is_passive(self, word):
    atributos = word.attrs.split('|')
    return True if 'Voice=Pass' in atributos else False
```

Ilustración 56. Método is_passive()

Número de verbos en voz pasiva sin agente

Para saber si un verbo es pasivo y además no tiene agente, se comprueba que el valor Voice=Pass esté entre las características morfológicas de la palabra (utilizando el método *is_passive()* definido anteriormente) y, además, se pasa por un filtro en el que se comprueba si la siguiente palabra es "by" (siempre y cuando el siguiente índice esté dentro del rango de la lista), tal y como puede verse en la Ilustración 57. En caso de no serlo, será un verbo en voz pasiva sin agente (*agentless*), por lo que se aumentará en uno el número de verbos en voz pasiva sin agente.

```
if word.index < len(frase):
    siguiente_word = frase[word.index].word.lower()
    if siguiente_word == 'by':
        return False
    else:
        return True
```

Ilustración 57. Comprobación de agente en verbo pasivo.

Número de verbos en infinitivo

Para saber si un verbo está en infinitivo, el valor *VerbForm=Inf* debe estar entre las características morfológicas de la palabra, por lo que se ha desarrollado el método *is_infinitive()* que comprueba si la palabra pasada como parámetro está en infinitivo o no (Ilustración 58). En caso de estarlo, se aumenta en uno el número de verbos en infinitivo.

```
def is_infinitive(self, word):
    atributos = word.attrs.split('|')
    return True if 'VerbForm=Inf' in atributos else False
```

Ilustración 58. Método is_infinitive()

Número de verbos en gerundio

En cambio, para saber si un verbo está en gerundio, el valor del *XPOS* de la palabra debe ser "VBG", puesto que esta etiqueta indica que se trata de un verbo en gerundio. Por ello, se ha desarrollado el método *is_gerund()* que devuelve True en caso de que el *XPOS* de la palabra sea igual a "VBG" (Ilustración 59). En caso de serlo, se aumenta en uno el número de verbos en gerundio.

```
def is_gerund(self, word):
    return word.xpos == 'VBG'
```

Ilustración 59. Método is_gerund()

Número de verbos irregulares en tiempo pasado

Para realizar la comprobación de los verbos irregulares en tiempo pasado, en primer lugar es necesario comprobar que sea un verbo irregular y, después, comprobar que ese verbo esté en pasado mediante el uso del método *is_past()* que se ha mostrado anteriormente.

Se utiliza una lista de verbos irregulares[10] para realizar la comprobación de que la palabra pasada como parámetro es irregular y se compara cada uno de los verbos de la lista con el lema (de esta manera, obtenemos la base del verbo) de dicha palabra (Ilustración 60).

```
def is_irregular(self, word):
    return True if word.lemma in FileLoader.irregular_verbs else False
```

Ilustración 60. Método is_irregular()

Como se puede observar, el listado de los verbos irregulares se encuentra cargado en la clase *FileLoader*.

Número de palabras negativas

Comprobar que una palabra es negativa es tan sencillo como obtener el lema del *token*/palabra y compararlo con la palabra "not". Si son iguales, significa que es una palabra negativa, por lo que se aumenta en uno el número de palabras negativas.

Número de pronombres personales

En primer lugar, se comprueba que la palabra sea un pronombre mirando que su UPOS sea igual a "PRON". Después, para saber si una palabra es un pronombre

[10] Lista obtenida de: https://github.com/Bryan-Legend/babel-lang/blob/master/Babel.EnglishEmitter/Resources/Irregular%20Verbs.txt

personal, el valor *PronType=Prs* debe estar entre las características morfológicas de la palabra, por lo que se ha desarrollado el método *is_personal_pronoun()* que comprueba si la palabra pasada como parámetro es un pronombre personal o no (Ilustración 61). En caso de serlo, se aumenta en uno el número de pronombres personales.

```
def is_personal_pronoun(self, word):
    atributos = word.attrs.split('|')
    return "PronType=Prs" in atributos
```

Ilustración 61. Método is_personal_pronoun()

Número de pronombres en primera persona

En primer lugar, se comprueba que la palabra sea un pronombre mirando que su UPOS sea igual a "PRON". Después, para saber si una palabra es un pronombre en primera persona, primero se mira si es un pronombre personal mediante el método *is_personal_pronoun()* y, por último, se comprueba que el valor *Person=1* esté dentro de sus características morfológicas. En caso de estarlo, se aumenta en uno el número de pronombres personales en primera persona.

Número de pronombres en primera persona del singular

En primer lugar, se comprueba que la palabra sea un pronombre mirando que su UPOS sea igual a "PRON". Después, para saber si una palabra es un pronombre en primera persona, primero se mira si es un pronombre personal mediante el método *is_personal_pronoun()* y, por último, se comprueba que los valores *Person=1* y *Number=Sing* estén de sus características morfológicas. En caso de estarlo, se aumenta en uno el número de pronombres personales en primera persona del singular.

Número de pronombres en tercera persona

Para calcular este último indicador morfológico, se comprueba que la palabra sea un pronombre mirando que su UPOS sea igual a "PRON". Después, para saber si una palabra es un pronombre en primera persona, primero se mira si es un pronombre personal mediante el método *is_personal_pronoun()* y, por último, se comprueba que el valor *Person=3* esté dentro de sus características morfológicas. En caso de estarlo, se aumenta en uno el número de pronombres personales en tercera persona.

6.3.6 Implementación de los indicadores de frecuencia de palabra (*Word Frequency*)

Para estimar el número de palabras de contenido (adjetivos, sustantivos, verbos y adverbios difíciles), se utiliza el método *zipf_frequency()* de la librería *wordfreq* (presentado anteriormente), que utiliza la escala *Zipf*.

Como se mencionó en el apartado 5.1, se cuenta el número de adjetivos, sustantivos, verbos y adverbios que tengan valores *Zipf* menor o igual a 3, ya que este es el punto de inflexión en el que las palabras tienden a ser más difíciles (Mandera, 2016). Por lo tanto, por cada palabra, se extrae el valor *Zipf* (Ilustración 62) y, si el valor es menor o igual a 3 y, además, la palabra es de contenido (sustantivo, adjetivo, adverbio o verbo), se incrementa en uno el contador de palabras raras.

```
wordfrequency = zipf_frequency(entry.word, 'en')
```

Ilustración 62. Valor Zipf de una palabra.

6.3.7 Implementación de los indicadores sintácticos

En este apartado, se expone cómo han sido implementados los indicadores sintácticos.

Número total de palabras de contenido

Para calcular el número total de palabras de contenido, se ha utilizado el método *is_lexic_word()*, que determina si una palabra pasada como parámetro es léxica (de contenido) o no. Para ello, se comprueba si cumple alguna de estas condiciones:

- Si el valor del UPOS es "NOUN".
- Si el valor del UPOS es "ADJ".
- Si el valor del UPOS es "ADV".
- Si el valor del UPOS es "VERB", o es "AUX" y además su padre (HEAD) no es "VERB". Esta comprobación se realiza mediante el método *is_verb()* visto anteriormente (Ilustración 35).

Número total de nombres

En caso de que el valor del UPOS de la palabra/*token* sea "NOUN", se incrementa en uno el número total de nombres/sustantivos que hay en el texto.

Número total de adjetivos

En caso de que el valor del UPOS de la palabra/*token* sea "ADJ", se incrementa en uno el número total de adjetivos que hay en el texto.

Número total de verbos

Se incrementa en uno el número total de verbos siempre y cuando el valor del UPOS del *token* sea "VERB", o sea "AUX" y además su padre (HEAD) no sea "VERB". Esta comprobación se realiza mediante el método *is_verb()* visto anteriormente (Ilustración 35).

La razón por la que también se comprueba que sea "AUX" y, además, su padre no sea "VERB" es porque existen los verbos compuestos y, si se tuviera en cuenta únicamente la etiqueta "AUX", se estaría contando cada palabra que compone el verbo compuesto. Por ejemplo, en la Ilustración 63 puede verse un verbo compuesto. En este caso, únicamente se contaría la palabra "written" como verbo, puesto que se cumple la condición de que la etiqueta del UPOS sea "VERB", y el resto de verbos ("have" y "been") no cumplen la condición de que el UPOS sea igual a "AUX" y su padre no sea "VERB" (puesto que el padre es "written", que es un verbo).

```
3       have    have    AUX     VBP     Mood=Ind|Tense=Pres|VerbForm=Fin        5       aux     _
4       been    be      AUX     VBN     Tense=Past|VerbForm=Part        5       aux:pass _
5       written write   VERB    VBN     Tense=Past|VerbForm=Part|Voice=Pass     0       root    _
```

Ilustración 63. Ejemplo verbo compuesto.

Esto se hace así porque existen verbos etiquetados como "AUX" (auxiliares) que sean verbos simples (por lo que no tienen ningún padre) por lo que, en ese caso, se contaría como verbo.

Número total de adverbios

En caso de que el valor del UPOS de la palabra/*token* sea "ADV", se incrementa en uno el número total de adverbios que hay en el texto.

Número total de oraciones subordinadas y subordinadas relativas

Para comprobar si una oración es subordinada, se comprueban las relaciones de dependencia de las palabras. Para ello, se ha creado una lista con todas las posibles etiquetas que puede tener una palabra que tenga una relación de dependencia de subordinación (Ilustración 64).

```
subordinadas_labels = ['csubj', 'csubj:pass', 'ccomp', 'xcomp', 'advcl', 'acl', 'acl:relcl']
```

Ilustración 64. Listado de etiquetas de oraciones subordinadas.

Por lo tanto, si una palabra dispone de alguna de las relaciones de dependencia que se encuentran en dicho listado, se considera una oración subordinada, por lo que se incrementa en uno la cantidad total de oraciones subordinadas.

Por otro lado, será una oración subordinada relativa si la etiqueta corresponde a `acl:relcl`, por lo que, además, se incrementará en uno el número de oraciones subordinadas relativas.

Media de marcas de puntuación por frase

Por cada frase del texto, se comprueban todos los *tokens* que la componen y, en caso de que el valor del UPOS del *token* sea "PUNCT", se incrementa en uno el número de marcas de puntuación en dicha frase. Al llegar al final de la frase, se introduce el valor del número de marcas de puntuación en una lista.

Al finalizar, tras haber comprobado todas las frases, se tendrá una lista de valores numéricos con el número de marcas de puntuación en cada frase, por lo que para obtener la media simplemente se utiliza el método *mean()* de NumPY.

Número de proposiciones

Si una palabra se encuentra etiquetada con alguna de las siguientes etiquetas de relación de dependencia, se considera que hay una proposición, por lo que se incrementa en uno el número total de proposiciones:

```
'conj', 'csubj', 'csubj:pass', 'advcl', 'acl', 'acl:relcl'
```

Media de proposiciones por frase

Para calcular la media de proposiciones por frase, se divide el número total de proposiciones entre el total de frases.

6.3.8 Implementación de los indicadores de cohesión

En este apartado, se presenta cómo han sido implementados los indicadores de cohesión. Es decir, los indicadores de superposición de palabras, similitud semántica y los conectores.

Superposición de palabras

La superposición de palabras se calcula tanto como para las frases adyacentes, como para los pares posibles (es decir, todas las frases con todas). Para encontrar las frases adyacentes, se crea una lista de *map*s. Cada elemento del *map* será el par de sentencias adyacentes, obtenido mediante el método *zip()* de *Python*, que divide devuelve una tupla por cada elemento pasado como parámetro ().

```
adjacents = list(map(list, zip(paragraph, paragraph[1:])))
```

Ilustración 65. Frases adyacentes.

Por otro lado, para calcular la superposición para todos los posibles pares, por cada frase de un párrafo, se recorre el resto de frases de dicho párrafo.

De esta manera, se pueden calcular los siguientes indicadores:

- **Superposición de sustantivos**: se crea una lista con todos los sustantivos que dos frases tengan en común (para comprobar si una palabra es sustantivo, se utiliza el campo UPOS, tal y como se puede observar en la Ilustración 66). Si existe al menos un sustantivo en común entre ambas frases, se considera que hay superposición y, por tanto, se incrementa el número de frases con superposición de sustantivos.

```
if entry1.upos == 'NOUN':
```

Ilustración 66. Es sustantivo.

- **Superposición de argumentos**: existe superposición de argumentos cuando hay coincidencia entre un sustantivo en una frase y el mismo sustantivo (en plural o singular) en la otra frase. También ocurre cuando hay coincidencias de pronombres personales entre las dos frases. Para comprobarlo, se utiliza el método *is_personal_pronoun()* previamente definido y se comprueba que la etiqueta UPOS corresponda a "NOUN", tal y como se puede observar en la Ilustración 67.

```
if self.is_personal_pronoun(entry1) or entry1.upos == 'NOUN':
```

Ilustración 67. Comprobación superposición de argumentos.

- **Superposición de raíces**: hay superposición de raíces cuando existe al menos una coincidencia entre un sustantivo en una frase y una palabra de contenido en la frase anterior, compartiendo un lema común. Para comprobarlo se utiliza el método *is_lexic_word()* previamente definido y, para comprobar el sustantivo, se comprueba que la etiqueta UPOS sea igual a "NOUN" (Ilustración 68).

```
for entry1 in x[0]:
    if self.is_lexic_word(entry1, x[0]):
        sentence1.append(entry1.lemma.lower())
for entry2 in x[1]:
    if entry2.upos == 'NOUN':
        sentence2.append(entry2.lemma.lower())
```

Ilustración 68. Comprobación superposición de raíces.

- **Superposición de palabras de contenido**: se comprueba que coincida al menos una palabra de contenido en cada par de frases. Para comprobar esa coincidencia, se utiliza el método *is_lexic_word()* previamente definido (Ilustración 69).

```
if self.is_lexic_word(entry1, x[0]):
```

Ilustración 69. Comprobación de superposición de palabras de contenido.

Similitud semántica

La primera opción para calcular la similitud existente entre dos frases o párrafos, fue la de utilizar *FastText*, que es una librería de aprendizaje de *word embeddings* y clasificación de texto creada por el laboratorio de Investigación de Inteligencia Artificial de Facebook (FAIR).

Esta librería utiliza *word embeddings*, que son representaciones vectoriales de palabras, para representar la semántica léxica. La idea tras este enfoque es la de codificar cada una de las palabras como vectores multidimensionales de números reales, de manera que las palabras similares tendrán una representación vectorial similar. La similitud entre dos palabras es normalmente evaluada mediante la similitud coseno[11] (Merril & Baum).

FastText parte las palabras en varios n-gramas (sub-palabras). Por ejemplo, los tri-gramas para la palabra *apple* son *app, ppl* y *ple.* El vector de *word embedding* para la palabra *apple* será la suma de todos esos n-gramas. Tras entrenar la red neuronal de *FastText*, se obtuvieron *word embeddings* para todos los n-gramas dados en el conjunto de entrenamiento. De esta manera, las palabras raras también estarán representadas,

[11] La fórmula es $\text{similarity} = \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$ y devuelve el ángulo entre dos vectores. Los vectores que apunten a la misma dirección tendrán una similitud coseno de 1, mientras que los que sean perpendiculares tendrán una similitud de 0. Por lo tanto, en el caso de los *Word embeddings*, los vectores altamente similares tendrán una similitud coseno cercano a 1, mientras las palabras que no tengan relación entre ellas tendrán un valor coseno cercano a 0 (Merril & Baum).

puesto que es muy probable que algunos de sus n-gramas también aparezcan en otras palabras (Merril & Baum).

El problema de utilizar *FastText* era que tan solo era posible obtener la similitud existente entre dos palabras y el objetivo real era obtener la similitud entre frases y párrafos. Una posible primera solución que se barajó fue la de comparar la media de todas las similitudes de las palabras de una frase, pero esta solución no da buenos resultados (Le & Mikolov).

Por ello, la segunda solución fue utilizar un modelo que estuviese entrenado y optimizado para que tuviera como entrada texto más grande (y no únicamente palabras), así como frases o párrafos. Para ello, se decidió utilizar un *Universal Sentence Encoder*.

Los *Universal Sentence Encoders*, cuyo modelo está modelo está entrenado en una gran variedad de fuentes de datos, codifican texto en vectores multidimensionales y están diseñados para realizar el mismo objetivo que *FastText*: los *embeddings* que producen pueden ser usados para variedad de aplicaciones, así como clasificación de texto, similitud, etc. La entrada del modelo es texto de longitud variable (en inglés) y la salida es un vector de 512 dimensiones. Además, este método es más eficiente que el de *FastText* (Cer, Yang, & Kong, 2018).

Concretamente, se ha utilizado el *Universal Sentence Encoder* de Google, y se ha implementado de la siguiente manera:

- Primero, se carga el módulo del *Universl Sentence Encoder* de TensorFlow (Ilustración 70).

```
module_url = "https://tfhub.dev/google/universal-sentence-encoder-large/2"
embed = hub.Module(module_url)
```

Ilustración 70. Carga de Universal Sentence Encoder.

- Se prepara el entorno para realizar los *embeddings* (Ilustración 71).

```
with tf.Session() as session:
    session.run(tf.global_variables_initializer())
    session.run(tf.tables_initializer())
```

Ilustración 71. Preparación del entorno de TensorFlow.

- Por cada frase, se crea su *embedding* y se introduce en una lista de *embeddings* de frases (Ilustración 72).

```
sentences_embeddings = session.run(similarity_sentences_encodings, feed_dict={similarity_input_placeholder: sent_tokenize(line)})
self.aux_lists['sentences_in_paragraph_token_list'].append(sentences_embeddings)
```

Ilustración 72. Creación de embeddings de frases.

- De la misma manera, por cada párrafo, se crea su *embedding* y se introduce en una lista de *embeddings* de párrafos (Ilustración 73).

```
self.aux_lists['paragraph_token_list'] = session.run(similarity_sentences_encodings, feed_dict={similarity_input_placeholder: paragraphs})
```

Ilustración 73. Creación de embeddings de párrafos.

La similitud se calcula realizando el producto escalar entre los vectores correspondientes a los *embeddings*. Para ello, se utiliza el método *calculate_similarity()* (Ilustración 74), que toma como parámetro dos vectores (correspondientes a los *embeddings*, ya sean frases o párrafos) y realiza el producto escalar mediante el método *inner()* de la librería *NumPY*.

```
def calculate_similarity(self, sentence1, sentence2):
    return np.inner(sentence1, sentence2)
```

Ilustración 74. Método calculate_similarity()

Por lo tanto, teniendo los *embeddings* de las frases y los párrafos, es posible calcular los siguientes indicadores:

a) Similitud semántica entre frases adyacentes

Se recorre la lista de *embeddings* de las frases adyacentes y se calcula la similitud mediante el método *calculate_similarity()*. Para ello, se utiliza el método *zip()*. Mediante este método, se crean tuplas de todas aquellas frases que sean adyacentes, tal y como se puede observar en la Ilustración 75.

```
def calculate_similarity_adjacent_sentences(self):
    i = self.indicators
    adjacent_similarity_list = []
    for sentence in self.aux_lists['sentences_in_paragraph_token_list']:
        if len(sentence) > 1:
            for x, y in zip(range(0, len(sentence) - 1), range(1, len(sentence))):
                adjacent_similarity_list.append(self.calculate_similarity(sentence[x], sentence[y]))
        else:
            adjacent_similarity_list.append(0)
    if len(adjacent_similarity_list) > 0:
        i['similarity_adjacent_mean'] = round(float(np.mean(adjacent_similarity_list)), 4)
        i['similarity_adjacent_std'] = round(float(np.std(adjacent_similarity_list)), 4)
```

Ilustración 75. Método calculate_similarity_adjacent_sentences()

b) Similitud semántica entre párrafos adyacentes

De igual forma, se recorre la lista de *embeddings* de los párrafos adyacentes y se calcula la similitud mediante el método *calculate_similarity()*. Para recorrer los párrafos adyacentes, se utiliza el mismo método que para recorrer las frases adyacentes. Es decir, mediante el método *zip()*, se crean tuplas de todos aquellos párrafos que sean adyacentes, tal y como se puede observar en la Ilustración 76.

```
def calculate_similarity_adjacent_paragraphs(self):
    i = self.indicators
    adjacent_similarity_par_list = []
    if len(self.aux_lists['paragraph_token_list']) > 1:
        for x, y in zip(range(0, len(self.aux_lists['paragraph_token_list']) - 1), range(1, len(self.aux_lists['paragraph_
            adjacent_similarity_par_list.append(self.calculate_similarity(self.aux_lists['paragraph_token_list'][x], self.
        if len(adjacent_similarity_par_list) > 0:
            i['similarity_adjacent_par_mean'] = round(float(np.mean(adjacent_similarity_par_list)), 4)
            i['similarity_adjacent_par_std'] = round(float(np.std(adjacent_similarity_par_list)), 4)
```

Ilustración 76. Método calculate_similarity_adjacent_paragraphs()

c) Similitud semántica entre todos los posibles pares de frases en un párrafo

Como se puede observar en la Ilustración 77, se recorre la lista de *embeddings* de los párrafos y se calcula la similitud entre todos los posibles pares de frases de un párrafo mediante el método *calculate_similarity_pairs_in()*.

```
def calculate_similarity_pairs_sentences(self):
    i = self.indicators
    similarity_pairs_list = []
    for paragraph in self.aux_lists['sentences_in_paragraph_token_list']:
        similarity_pairs_list.append(self.calculate_similarity_pairs_in(paragraph))
    i['similarity_pairs_par_mean'] = round(float(np.mean(similarity_pairs_list)), 4)
    i['similarity_pairs_par_std'] = round(float(np.std(similarity_pairs_list)), 4)
```

Ilustración 77. Método calculate_similarity_pairs_sentences()

En la Ilustración 78 se puede ver el funcionamiento del método *calculate_similarity_pairs_in()*. En primer lugar, se comienza recorriendo el párrafo y, por cada elemento del mismo, se crea una sublista que consta de los elementos que están a la derecha del elemento actual (sin contar el propio elemento). Por ejemplo, si se está recorriendo la siguiente lista:

```
[“This”, “is”, “an”, “example”, “.”]
```

Para el índice 0 (correspondiente al elemento *This*), la sublista que se crearía sería:

```
[“is”, “an”, “example”, “.”]
```

Para el índice 1, la sublista sería:

```
[“an”, “example”, “.”]
```

Y el proceso sigue hasta llegar al final de la lista.

Por lo tanto, teniendo este concepto claro, se puede calcular la similitud (mediante el método *calculate_similarity()* visto en la Ilustración 74) que tiene el elemento actual de la lista con cada uno de los elementos de la sublista. Con esto se consigue obtener la similitud de cada par posible par de frases en el párrafo.

```
def calculate_similarity_pairs_in(self, paragraph):
    list_similarities_mean = []
    for index in range(len(paragraph)):
        similarity_tmp = paragraph[index+1:]
        x = paragraph[index]
        for index2 in range(len(similarity_tmp)):
            y = similarity_tmp[index2]
            list_similarities_mean.append(self.calculate_similarity(x, y))
    if len(list_similarities_mean) > 1:
        return round(float(np.mean(list_similarities_mean)), 4)
    else:
        return 0.0
```

Ilustración 78. Método calculate_similarity_pairs_in()

Pese a que esta solución funciona y se obtienen los resultados esperados, se tuvo que limitar el número de ficheros que pueden ser analizados puesto que el cálculo de este indicador consume muchos recursos y, por lo tanto, hace inviable el uso del programa cuando el número de ficheros es mayor que cinco.

Para solucionar este inconveniente, puesto que contratar un servidor con mayor potencia no era posible, se decidió que lo mejor era hacer que este indicador fuera opcional y, por lo tanto, si el usuario desease calcularlo, estaría limitado a hacerlo analizando un máximo de 5 ficheros al mismo tiempo. Si, por el contrario, decidiera no calcularlo, el usuario no estaría limitado a cinco ficheros, sino que podría analizar cualquier cantidad de manera simultánea.

Conectores

Para realizar el cálculo del número de conectores según su categoría (causales, lógicos, adversativos, etc.), se hace uso de un listado[12] en formato de texto (.TXT) que tiene la siguiente estructura:

- La categoría empieza por el carácter "//". Por ejemplo, `//causal` hace referencia a la categoría de conectores causales.
- Tras la categoría, comienza el listado de conectores de dicha categoría, cada uno en una nueva línea. Por ejemplo:

```
because
since
as
inasmuch as
now that
as long as
```

- Se puede dar el caso de que algunos conectores puedan tener otras palabras al principio, en medio o al final del conector. En ese caso, se utiliza el carácter "*" para representarlo. Por ejemplo, para `such*that` podría darse el caso de que haya otras palabras entre `such` y `that`.

Teniendo la estructura del fichero de la lista de conectores clara, se puede proceder a explicar el proceso que se sigue para calcular el número de conectores según la categoría.

En primer lugar, se utiliza el método *load_connectives_list()* la clase *FileLoader* para leer el fichero de texto y crear un diccionario de *Python* en el que la clave es la categoría del conector (si empieza por el carácter "//", será la categoría) y el valor será una lista compuesta por los conectores correspondientes a esa categoría (Ilustración 79).

[12] Obtenido de: https://www.sparklebox.co.uk/literacy/vocabulary/word-lists/connectives/#.XPu3b8RR2Uk

```
def load_connectives_list():
    f = open('data/connectives.txt', 'r')
    lineas = f.readlines()
    categoria = ''
    for linea in lineas:
        if linea.startswith("//"):
            categoria = linea.replace('//', '').replace('\n', '').lower()
        else:
            FileLoader.connectives[categoria].append(linea.replace('\n', ''))
    f.close()
```

Ilustración 79. Método load_connectives_list()

Una vez cargado el diccionario con las categorías y sus correspondientes conectores, se utiliza el método *calculate_connectives_for()* para calcular la cantidad de conectores existentes en el texto para la categoría pasada como parámetro. En este método, se crean dos listas (de esta manera, se pueden utilizar dos expresiones regulares distintas para cada lista):

- Una con aquellos conectores que tengan el carácter "*".
- Otra con aquellos conectores que no lo tengan.

La primera lista se parte en una sublista en base al carácter "*". Después, se comprueba si existe cada elemento de la sublista utilizando la expresión regular "`\b%s\b[^.!?()]+\b%s\b`", que comprueba lo siguiente:

- `\b%s\b`: existe la palabra `%s` (siendo `%s` el primer elemento de la sublista). Por ejemplo, `%s` podría ser `such` y la expresión quedaría en `\bsuch\b`.
- `[^.!?()]+`: no existe (`^`), una o más veces (`+`), los caracteres `.!?()` que indican el final de una frase. Con esto se consigue que solo se busquen apariciones de las palabras siempre y cuando estén en la misma frase.
- `\b%s\b`: existe la palabra `%s` (siendo `%s` el segundo elemento de la sublista). Por ejemplo, `%s` podría ser `that` y la expresión quedaría en `\bthat\b`.

La segunda lista comprueba si existe el conector mediante la siguiente expresión regular: `\b%s\b`. Esta expresión simplemente trata de encontrar la palabra `%s`. Por ejemplo, `%s` podría ser `as long as` y la expresión quedaría en `\bas long as\b`.

Para contar el número de apariciones que cumplan la expresión regular, simplemente hay que obtener la longitud de la lista devuelta por el método *findall()* (Ilustración 80). Este método del módulo *re*, el cual sirve para trabajar con expresiones regulares, trata de encontrar todas las posibles apariciones que cumplan con la expresión regular pasada como parámetro.

```
def calculate_connectives_for(self, text, connective):
    list = FileLoader.connectives.get(connective)
    list_a = []
    list_b = []
    num_a = 0
    num_b = 0
    for x in list:
        if "*" in x:
            list_a.append(x)
        else:
            list_b.append(x)
    for a in list_a:
        split = a.split('*')
        matches_a = re.findall(r'\b%s\b[^.!?()]+\b%s\b' % (split[0], split[1]), text)
        num_a += len(matches_a)
    for b in list_b:
        matches_b = re.findall(r'\b%s\b' % b, text)
        num_b += len(matches_b)
    return num_a + num_b
```

Ilustración 80. Método que calcula el número de conectores.

6.4 CLASIFICADOR

Para clasificar un texto según su complejidad, se ha desarrollado un clasificador supervisado. En el aprendizaje supervisado, se parte de datos previamente etiquetados (*labeled data*), y se trata de encontrar una función que sea capaz de predecir etiquetas de salida utilizando las variables de entrada. Es decir, en el caso de este proyecto, las variables de entrada serán los distintos indicadores previamente calculados y, mediante un algoritmo de clasificación supervisada, se tratará de hallar aquellos atributos (o indicadores) que sean los más importantes (es decir, aquellos que influyan más sobre la etiqueta o clase).

6.4.1 Conjunto de datos

En primer lugar, el *corpus* o conjunto de datos utilizado para el entrenamiento y aprendizaje del clasificador es el compilado en *OneStopEnglish*, que es un sitio web en el que se proveen distintos recursos para aprender inglés. Una de las funcionalidades de este sitio es una sección de noticias semanales, que contiene artículos del periódico *The Guardian* y son reescritos por profesores para adecuarse a los tres niveles de estudiantes de inglés como segundo lenguaje (básico, intermedio y avanzado), de manera que cada uno de los artículos está escrito en tres niveles distintos. El artículo reescrito para el nivel avanzado sería el que más se acerca al artículo original, aunque no es exactamente el mismo (Vajjala & Lucic, 2018).

Por lo tanto, se utilizará este *corpus* para entrenar el modelo, puesto que se puede utilizar libremente[13], siempre y cuando se cite a los autores (Vajjala & Lucic, 2018). Tal y como se puede observar en la Ilustración 81, cada uno de los artículos está distribuido en tres carpetas, cada una con los artículos que pertenecen al correspondiente nivel: *Adv-Txt*, *Int-Txt* y *Ele-Txt*.

Ilustración 81. Distribución del corpus de OneStopEnglish.

Para entrenar y evaluar el modelo, se parte el conjunto de datos en dos: el conjunto de entrenamiento y el conjunto de pruebas. Esta partición se realiza de manera manual, cogiendo el 20% de cada una de las carpetas para la parte de test, de manera que se tendrá el 80% del conjunto de datos para el entrenamiento y el 20% para las pruebas. Como cada una de las carpetas contiene 189 artículos y el 20% de 189 es aproximadamente 37, se organizarán las carpetas de la siguiente manera, tal y como puede verse en la Ilustración 82:

- Para el conjunto de entrenamiento: 152 artículos para cada uno de los niveles, haciendo un total de 456 artículos.
- Para el conjunto de prueba: 37 artículos para cada uno de los niveles, haciendo un total de 111 artículos.

[13] Se ha obtenido de https://github.com/nishkalavallabhi/OneStopEnglishCorpus/tree/master/Texts-SeparatedByReadingLevel

Ilustración 82. Partición de training y test.

Tras partir el conjunto de datos, se procede a ejecutar la aplicación *AzterTest* para cada uno de los artículos, consiguiendo así un fichero CSV con todos los resultados que se utilizarán para el entrenamiento y evaluación del clasificador. Con la intención de facilitar esta tarea, se ha creado un argumento opcional (*-a* o *–all*) que se utiliza para crear un único fichero CSV con los resultados del análisis de los ficheros (Ilustración 83).

```
optional.add_argument('-a', '--all', action='store_true', help="Generate a CSV file with all the results")
```

Ilustración 83. Argumento opcional.

De esta manera, si se introduce este argumento a la hora de ejecutar la aplicación mediante la línea de comandos, se ejecutará un método que automatiza la tarea de crear un fichero CSV con todos los resultados de los análisis realizados por *AzterTest* y esto facilitará la posterior carga de los conjuntos de datos.

6.4.2 Carga y organización del conjunto de datos

Para cargar y organizar el conjunto de datos obtenido anteriormente, se va a utilizar la librería *Pandas*, que es una herramienta de manipulación de datos de alto nivel que utiliza una estructura de datos llamada *DataFrame*.

Para realizar la carga de los ficheros, se utiliza el método *read_csv* de *Pandas* (Ilustración 84). De esta manera, se obtienen dos *DataFrames* para el conjunto de entrenamiento y el conjunto de pruebas, por lo que se vuelven fácilmente manipulables.

```
dataset_train = pd.read_csv('dataset_train.csv')
dataset_test = pd.read_csv('dataset_test.csv')
```

Ilustración 84. Método read_csv()

Después, se establece la clase (que, en este caso, es *level*) a predecir. Para ello, tanto para el conjunto de entrenamiento como para el de pruebas, se establecen las variables *X_train* y *X_test*, que tendrán los valores de los *features* o atributos, y las variables *y_train* y *y_test*, que tendrán los valores de las distintas clases. En la Ilustración 85 puede verse cómo se establecen estas variables para el conjunto de entrenamiento (*X_train* y *y_train*).

```
feature_names = dataset_train.columns.tolist()
feature_names.remove("level")

X_train = dataset_train[feature_names]

y_train = dataset_train['level']
```

Ilustración 85. Establecer clase.

6.4.3 Selección de atributos

Mediante el proceso de selección de atributos se busca el subconjunto de atributos que sea más relevante para la creación del modelo predictivo que se va a construir. Esto supone, principalmente, las siguientes ventajas (Asaithambi, 2018):

- Se consigue reducir el *overfitting*, es decir, que el modelo se ajuste demasiado a los datos de entrenamiento, de manera que no se consiguen los mejores resultados.
- Se reduce el tiempo de entrenamiento ya que, al disponer de menos datos que procesar, los algoritmos pueden aprender más rápidamente.

En *Python*, para realizar la selección de atributos, se utiliza el método *SelectKBest()* de la librería *SKLearn*. En concreto, se va a utilizar la prueba de Chi cuadrado (χ^2) para encontrar los mejores atributos y, en este caso, se establece que se desea conservar los 30 mejores (*k=30*), tal y como se puede observar en la Ilustración 86. Una vez se hayan encontrado los 30 mejores atributos, se procede a transformar los conjuntos de entrenamiento y de pruebas.

```
selector=SelectKBest(score_func=chi2,k=30)

X_train_new = selector.fit_transform(X_train,y_train)
X_test_new = selector.transform(X_test)
```

Ilustración 86. Selección de atributos.

Cabe destacar que, pese a que en este caso se ha decidido conservar los 30 mejores atributos, en el capítulo 7 se han realizado pruebas con distintos valores para la K, con el objetivo de encontrar el valor más óptimo.

6.4.4 Selección de algoritmos y optimización de hiperparámetros

Antes de proceder a evaluar y entrenar el modelo con el conjunto de entrenamiento, es importante, en primer lugar, seleccionar los algoritmos que mejor se adapten a las necesidades y, en segundo lugar, encontrar los parámetros más óptimos para dichos algoritmos de aprendizaje automático. Estos parámetros son llamados hiperparámetros, y sirven para controlar el proceso de aprendizaje de dichos algoritmos (Agrawal, 2019).

Para seleccionar los algoritmos de aprendizaje, se hace uso del *cheat-sheet* que proporciona *SKLearn*[14]. Este *cheat-sheet* sirve de gran ayuda para encontrar los algoritmos de aprendizaje que mejor se adapten a los datos. Como se puede observar en la Ilustración 87, el *cheat-sheet* de *Scikit-Learn* indica que los algoritmos que mejor se adaptan a los conjuntos de datos del proyecto son:

- *Linear SVC*
- *K-Neighbors Classifier*
- *Ensemble Classifiers* (*Random Forest Classifier*, *Ada Boost Classifier* y *Gradient Boosting Classifier*)

[14] Este *cheat-sheet* puede encontrarse en: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

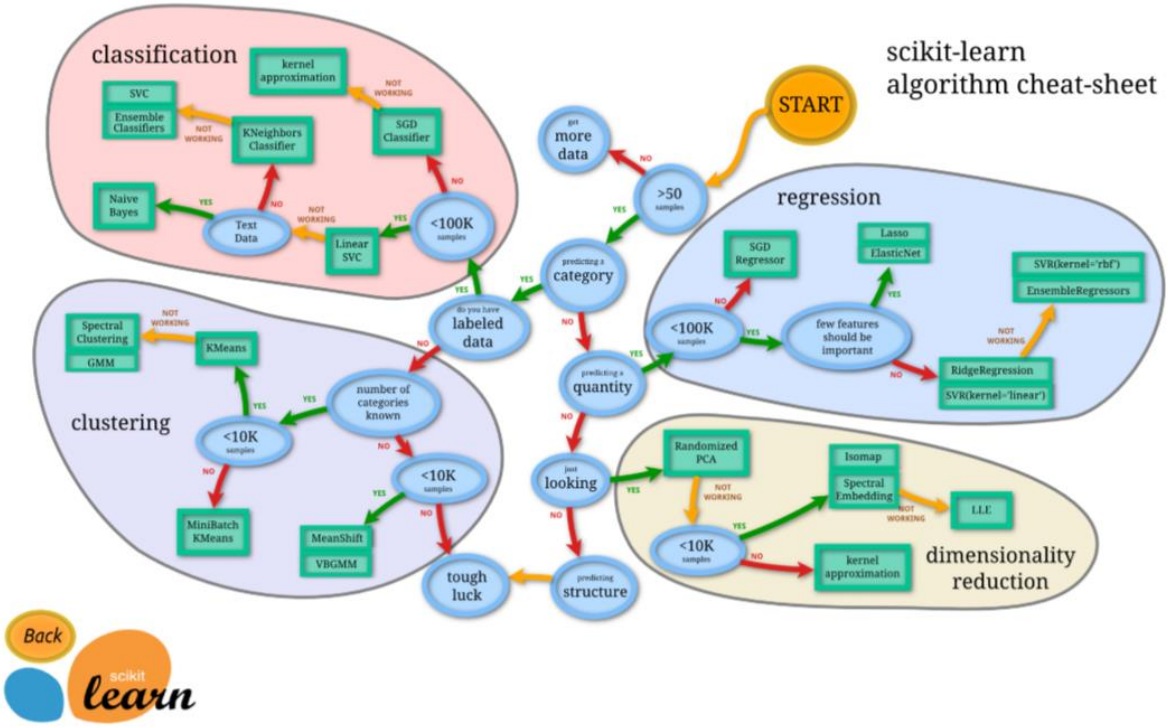


Ilustración 87. Scikit-Learn Algorithm Cheat-sheet.

Por lo tanto, para dichos algoritmos, se establecen los hiperparámetros que cada uno de los algoritmos procesará en el momento del entrenamiento (Ilustración 88) y, el método *GridSearchCV()* de *SKLearn* será el encargado de encontrar los parámetros más óptimos que mejores resultados ofrezca. Concretamente, *GridSearchCV()* utiliza la técnica de *K-Fold Cross Validation* (con *K*=10) para validar los resultados y encontrar el mejor (Ilustración 89). El *10-Fold Cross Validation* divide el conjunto de entrenamiento en 10 fragmentos y realiza 10 iteraciones. En cada iteración, nueve de los diez fragmentos se utilizan como conjunto de entrenamiento, mientras que el fragmento restante sirve como conjunto de pruebas.

```
params_grid = [{
                'estimator':[KNeighborsClassifier()],
                'estimator__n_neighbors': [3,4,5,6,7,8,9,10,20],
                'estimator__weights': ['uniform', 'distance'],
                'estimator__algorithm': ['ball_tree', 'kd_tree', 'brute'],
                'estimator__metric': ['euclidean', 'manhattan', 'minkowski'],
                },
                {
                'estimator': [LinearSVC()],
                'estimator__C': [0.025, 0.5, 1, 10, 100, 1000],
                'estimator__max_iter': [2000, 3000, 4000, 5000, 6000, 7000, 8000]
                },
                {
                'estimator': [RandomForestClassifier()],
                'estimator__criterion': ['gini', 'entropy'],
                'estimator__n_estimators': [10, 15, 20, 25, 30, 100, 500, 1000],
                'estimator__max_features': ['auto', 'sqrt', 'log2'],
                },
                {
                'estimator': [AdaBoostClassifier()],
                'estimator__n_estimators': [10, 15, 20, 25, 30, 100, 500, 1000],
                'estimator__learning_rate': [.001, 0.01, .1],
                },
                {
                'estimator': [GradientBoostingClassifier()],
                "estimator__learning_rate": [0.15, 0.1, 0.05, 0.01, 0.005, 0.001],
                "estimator__max_depth": [3, 5, 8],
                "estimator__max_features": ["log2", "sqrt"],
                "estimator__n_estimators": [100, 250, 500, 750, 1000, 1250, 1500, 1750],
                }
              ]
```

Ilustración 88. Algoritmos e hiperparámetros.

```
clf = GridSearchCV(pipe, params_grid, n_jobs=-2, cv=skfold,  verbose=10)
```

Ilustración 89. Método GridSearchCV()

En el capítulo 7 se realizan pruebas para hallar el mejor algoritmo utilizando el método *GridSearchCV()* y, además, se trata de hacer una selección de atributos, tal y como se ha explicado anteriormente.

6.4.5 Evaluación

Una vez el método *GridSearchCV()* encuentra el algoritmo de aprendizaje que mejores resultados proporcione, se procede a entrenar el modelo (Ilustración 90) para evaluarlo utilizando el conjunto de pruebas.

```
clf.fit(X_train_new, y_train)
```

Ilustración 90. Entrenamiento del modelo.

Después, se realiza la predicción sobre el conjunto de pruebas mediante el método *predict()* (Ilustración 91) y se imprime tanto la matriz de confusión como la *accuracy* de la predicción realizada (Ilustración 92).

```
y_pred = clf.predict(X_test_new)
```

Ilustración 91. Realizar predicción.

```
unique_label = np.unique(y_test)
print("\nConfusion matrix for Test Set:")
print(pd.DataFrame(confusion_matrix(y_test, y_pred, labels=unique_label),
                   index=['Real:{:}'.format(x) for x in unique_label],
                   columns=['Predicted:{:}'.format(x) for x in unique_label]))

print("\nAccuracy: "+ str(accuracy_score(y_test, y_pred) * 100))
```

Ilustración 92. Imprimir matriz de confusión y precisión.

6.4.6 Guardado y carga del modelo

Para guardar el modelo en un fichero, de manera que más adelante éste pueda ser cargado y utilizado para predecir la complejidad de un texto, se utiliza el método *dump()*. Además de guardar el modelo, también se guarda el selector de atributos para que se pueda realizar la misma selección de atributos sobre los datos a predecir (Ilustración 93).

```
joblib.dump(clf, 'classifier.pkl')
pickle.dump(selector, open("selector.pickle", "wb"))
```

Ilustración 93. Guardar clasificador y selector de atributos.

Por último, para poder realizar la predicción de la dificultad teniendo los resultados de *AzterTest*, se carga el modelo y el selector mediante el método *load()*, se aplica el selector de atributos a los datos mediante el método *transform()* y se realiza la predicción utilizando el método *predict()* (Ilustración 94).

```
def predict_dificulty(self, data):
    feature_names = data.columns.tolist()
    X_test = data[feature_names]

    # Para cargarlo, simplemente hacer lo siguiente:
    clf = joblib.load('classifiers/classifier_aztertest.pkl')

    with open("classifiers/selectorAztertest.pickle", "rb") as f:
        selector = pickle.load(f)

    X_test_new = selector.transform(X_test)
    return clf.predict(X_test_new)
```

Ilustración 94. Predecir dificultad.

6.5 APLICACIÓN WEB

Con el objetivo de que la aplicación *Python* pueda ser utilizada de una manera más visual y sin tener que recurrir a la terminal para hacer uso de ella, se ha desarrollado una aplicación web en la que el usuario puede subir uno o más ficheros para poder analizarlos de una manera cómoda y sencilla.

En primer lugar, tal y como se ha explicado anteriormente, la aplicación web se aloja en un servidor remoto. Para acceder a él es necesario utilizar SSH (también conocido como *Secure Shell*). SSH es un protocolo que permite el acceso remoto de una máquina a otra de manera segura. El comando a utilizar para acceder al servidor, por tanto, es el siguiente:

```
ssh [usuario]@[ip]
```

La etiqueta `[usuario]` ha de sustituirse por el nombre de usuario con el que se desea iniciar sesión, mientras que `[ip]` corresponde a la IP de la máquina remota.

Puesto que, por la razón que se explicará más adelante, en el servidor ha de ejecutarse la aplicación *Python* previamente desarrollada, es necesario crear el entorno virtual de *Python* e instalar en él las librerías necesarias, tal y como se hizo en el apartado 6.2. Una vez se dispone del entorno y de las librerías, se puede comenzar a trabajar con la aplicación web.

Con el fin de facilitar el desarrollo, la aplicación web se desarrolla de manera local y, cuando el desarrollo de ésta haya terminado, se procede a subir los ficheros de la aplicación al servidor. Para subir los ficheros al servidor, se utiliza el comando `scp` (*Secure Copy*), que permite copiar ficheros desde una máquina a otra de manera segura (utiliza SSH para transferir los datos). Por ejemplo, para subir el fichero *index.php* al

directorio */var/www/html* del servidor, se utilizaría el siguiente comando (sustituyendo tanto `[usuario]` como `[ip]` por sus correspondientes valores):

```
scp /var/www/html/index.php [usuario]@[ip]:/var/www/html/
```

En primer lugar, para poder crear aplicaciones web es necesario un servidor web. En este proyecto, se trabaja con Apache Web Server, tal y como se explicó en el apartado 2.4. Por lo tanto, tanto en el servidor como en local es imprescindible instalar dicho servidor. Además, puesto que también se trabaja con PHP, es necesario instalar el módulo PHP de Apache. Para ello, se ejecutan los siguientes comandos:

```
sudo apt-get install apache2
sudo apt install php libapache2-mod-php
sudo systemctl restart apache2
```

Los ficheros de la aplicación web se encuentran, por tanto, en la carpeta `/var/www/html`. Esta carpeta dispone de la estructura que puede observarse en la Ilustración 95.

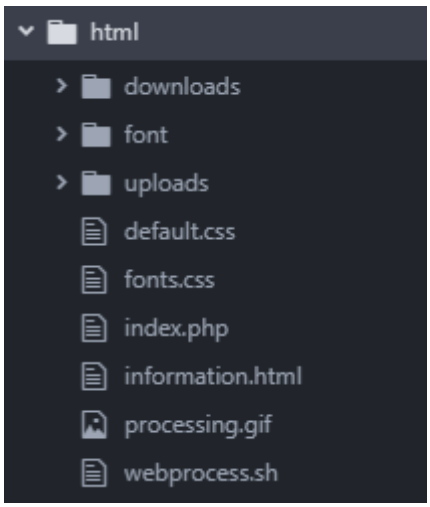


Ilustración 95. Estructura de la aplicación web.

Por lo tanto, el sitio web dispone de los ficheros *index.php* e *information.html* y, para definir los estilos e implementar la interfaz de usuario de la aplicación web se ha utilizado el lenguaje de diseño CSS (*Cascading Style Sheets*). Los estilos del sitio web se definen en el fichero *default.css* y éste se carga en los ficheros *index.php* e *information.html*, tal y como se puede observar en la Ilustración 96.

```
<link href="default.css" rel="stylesheet" type="text/css" media="all" />
```

Ilustración 96. Carga del fichero CSS.

Además, se utiliza una fuente especial, que se define en la carpeta */font* y el fichero *fonts.css*. Por otro lado, se encuentran las carpetas */downloads* y */uploads*, donde se alojarán los ficheros de los resultados y los documentos de texto que el usuario desee analizar, respectivamente.

En el fichero *index.php*, el cual corresponde a la página principal de la aplicación (Ilustración 97), se implementa la funcionalidad de subir documentos de texto para que estos sean analizados por la aplicación *Python*. Para ello, se implementa un formulario con un capo de tipo *file* para que el usuario pueda seleccionar ficheros y, una vez haya seleccionado todos, pulse el botón "Analyze" para que el proceso de evaluación de los textos comience (Ilustración 98).

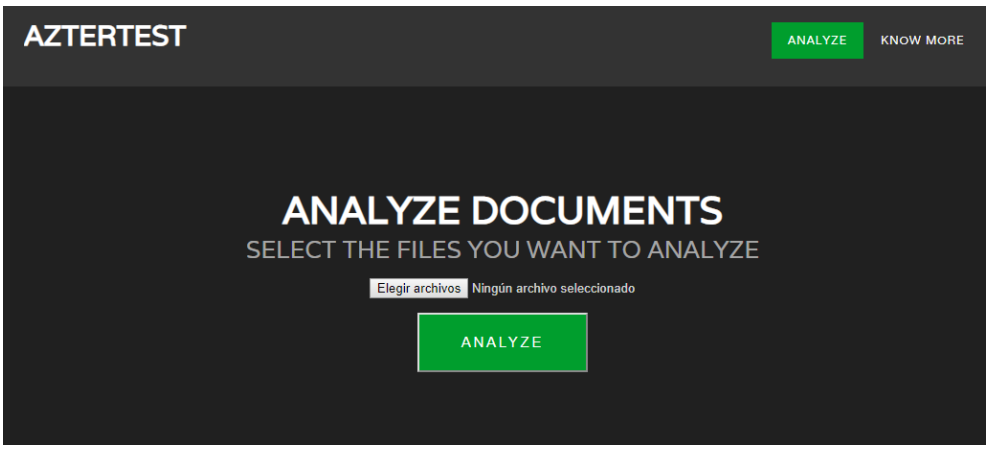


Ilustración 97. Página principal de la aplicación.

```
<form id="upload" enctype="multipart/form-data" method="post" onsubmit="return checkextension(); ">
  <li><input id="infile" name="infile[]" type="file" value='Seleccionar' multiple></li>
  <li><input type="submit" id="submit" name="submit" value="ANALYZE" class="button"></li>
</form>
```

Ilustración 98. Implementación del formulario.

Al pulsar el botón, tal y como indica el campo "onsubmit" del formulario, se ejecuta el método *checkextension()*. Este método, desarrollado en *JavaScript*, comprueba, en primer lugar, que se haya seleccionado al menos un fichero y, en caso de que así sea, que dichos ficheros estén en formato .txt, .odt, .doc o .docx. Tras realizar esa comprobación, se mostrará un mensaje y una imagen (*processing.gif*) que indicarán que los ficheros están siendo analizados (Ilustración 99).

```
function checkextension() {
  var file = document.querySelector("#infile");
  if (document.getElementById("infile").files.length == 0) {
     alert("You have to upload at least one file!");
     return false;
   }  else {
     var index = 0;
     while (index < document.getElementById("infile").files.length){
       if (/\.(txt|docx|doc|odt)$/i.test(file.files[index].name) === false ) {
        alert("Invalid format. Only .txt, .odt, .doc or .docx files allowed.");
         return false;
     }
     index++;
    }
  }
  document.getElementById("mensajeResultados").innerHTML="Analyzing... This can take a few minutes.\n\nPlease, wait.";
  document.getElementById("resultados").innerHTML = "";
  processing();
}
```

Ilustración 99. Método checkextension()

Si tanto la cantidad de ficheros como el formato de los mismos es correcto, se ejecutará el código PHP que se encarga de subir los ficheros, ejecutar la aplicación *Python* para que ésta los analice y mostrar los resultados en pantalla (además de dar la opción de descargarlos en formato .csv) una vez la aplicación haya terminado.

Por lo tanto, cuando envíe el formulario con los ficheros haciendo clic en el botón "Analyze" previamente mostrado, se ejecutará el siguiente código PHP:

- En primer lugar, se suben los ficheros al servidor. Para ello, se crea una carpeta (cuyo nombre será aleatorio) dentro de la carpeta */uploads* y se hace uso del método *move_uploaded_file()* de PHP para mover cada uno de los ficheros seleccionados a la carpeta recién creada (Ilustración 100).

```
$name = md5(rand() * time());
$uploadDir = "uploads/".$name;
if (!is_dir($uploadDir)) {
    mkdir($uploadDir, 0777, true);
}
for ($i = 0; $i < count($_FILES['infile']['name']); $i++) {
  echo " <a href='#".$i."'>Results of ".$_FILES['infile']['name'][$i]."</a><br>";
  $moved = move_uploaded_file($_FILES['infile']['tmp_name'][$i], $uploadDir."/" . $_FILES['infile']['name'][$i]);
  if( $moved ) {
    echo "<br>";
  } else {
    echo "The files could not be loaded. <br>";
  }
}
```

Ilustración 100. Subir ficheros al servidor.

- Después, mediante un script de *Bash* (*webprocess.sh*), ejecutado mediante el método *exec()* de PHP, se ejecuta el programa *Python* que analiza los ficheros y, además, en la carpeta */downloads*, crea un fichero .zip con los resultados. El método *exec()* toma como parámetro el comando completo del script de *Bash*. Por lo tanto, se le pasa el directorio del script seguido del directorio de los ficheros a analizar y el fichero .zip (Ilustración 101).

```
$zip= 'downloads/Aztertest_'.$name.'.zip';
$cmd = $binPath." ../".$uploadDir."/* ../../".$zip." ../".$uploadDir;
exec($cmd." 2>&1", $output, $return);
```

Ilustración 101. Ejecución del script de Bash.

- Tal y como se puede observar en la Ilustración 102, este script de *Bash* realiza las siguientes acciones:
    1. Activar el entorno virtual de *Python*.
    2. Ejecutar el programa *Python* previamente desarrollado. Este programa toma como parámetro el directorio donde se encuentran los ficheros a analizar.
    3. Crear un fichero .zip con los resultados obtenidos del análisis realizado por la aplicación *Python*.

```
#!/bin/bash
cd /var/www/html/python3envmetrix
source bin/activate
python3 ./main.py -f $1
cd $3
zip -q -x ./*.txt -r $2 ./*
#Para salir
deactivate
cd ..
```

Ilustración 102. Script de Bash.

- Por último, tras analizar los ficheros, se muestra un enlace para la descarga del fichero .zip de los resultados creado por el script de *Bash* previamente mostrado y se crea una tabla para que el usuario pueda ver los resultados directamente en la página web de una manera visualmente agradable (Ilustración 103).

```php
echo "<script>$('#mensajeResultados').html('<a href=".$zip.">Download results</a>');</script>";
$counter = 0;
foreach (new DirectoryIterator("./".$uploadDir.'/results') as $fileInfo) {
    if($fileInfo->isDot()) continue;
    echo "<table>";
    echo "<thead>
    <tr>
    <th colspan='3' id='".$counter."'>File: ".$_FILES['infile']['name'][$counter]."</th>
    </tr>
    </thead>";
    $f = fopen("./".$uploadDir.'/results'.'/'.$fileInfo->getFilename(), "r");
    while (($line = fgetcsv($f, 0, ":")) !== false) {
            echo "<tr>";
            foreach ($line as $cell) {
              if (count($line) == 1){
                    echo "<td colspan='3' id='titulo'>". htmlspecialchars($cell)."</tr>";
              }else{
                    echo "<td>" . htmlspecialchars($cell) . "</td>";
              }
            }

            echo "</tr>\n";
    }
    fclose($f);
    echo "\n</table><ul class='actions'>
      <li><a href='#' class='button'>Go to the top</a></li>
    </ul>";
    $counter++;
}
```

Ilustración 103. Mostrar resultados.

Por otro lado, el contenido del fichero *information.html* se muestra al pulsar sobre el botón "Know more" de la parte superior derecha del sitio web (este botón puede verse en la Ilustración 97). La finalidad de esta página es meramente informativa, por lo que simplemente muestra información acerca de la aplicación y los indicadores que genera, tal y como puede verse en la Ilustración 104.

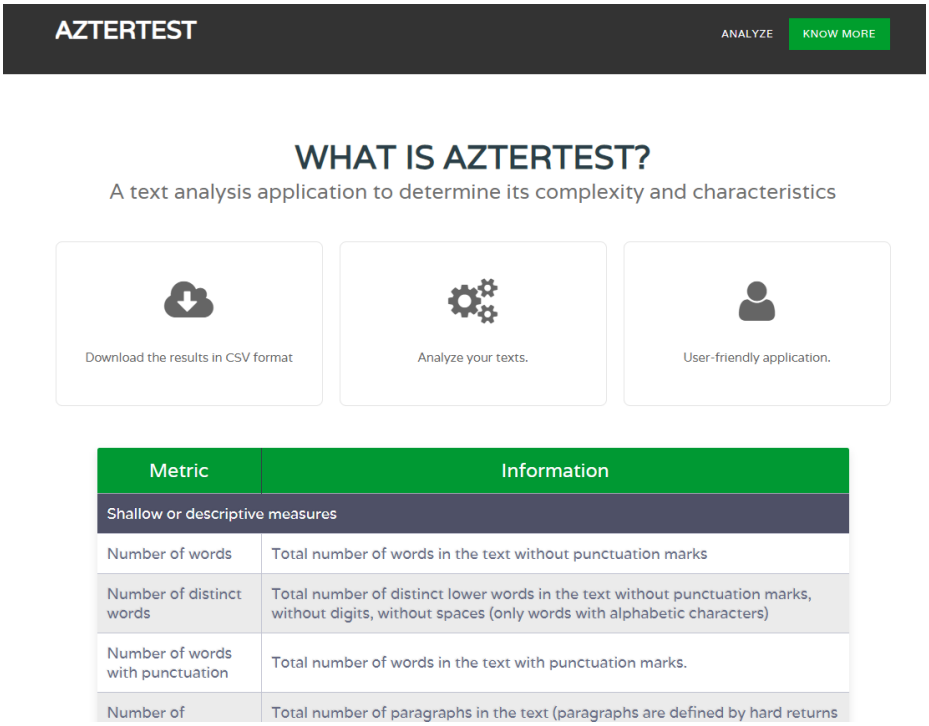


Ilustración 104. Página "Know more".

7 Verificación y evaluación

En este capítulo, se identifican y analizan las pruebas que se han realizado para comprobar el correcto funcionamiento de las diferentes partes que componen este proyecto. Para ello, se mostrará el contexto en el que estas fueron realizadas y los resultados obtenidos de las mismas. En concreto, se diferenciarán las pruebas realizadas sobre la aplicación *Python* y la aplicación web, además de verificar que los distintos indicadores estén correctamente calculados. Por otro lado, también se realizarán pruebas con el clasificador desarrollado, con el fin de obtener el resultado más óptimo posible.

7.1 PRUEBAS DE LA APLICACIÓN *PYTHON*

A continuación, se procede a realizar una serie de pruebas para comprobar el correcto funcionamiento de la aplicación *Python*.

ID	Prueba realizada	Resultado esperado	Resultado obtenido	¿Correcto?
1	Ejecutar el programa especificando un fichero de texto en formato ODT	El fichero se analiza y se muestran los resultados, además de generar un fichero CSV con los mismos.	El fichero se analiza y se muestran los resultados, además de generar un fichero CSV con los mismos.	SÍ.
2	Ejecutar el programa especificando un fichero de texto en formato DOC	El fichero se analiza y se muestran los resultados, además de generar un fichero CSV con los mismos.	El fichero se analiza y se muestran los resultados, además de generar un fichero CSV con los mismos.	SÍ.
3	Ejecutar el programa especificando un fichero de texto en formato DOCX	El fichero se analiza y se muestran los resultados, además de generar un fichero CSV con los mismos.	El fichero se analiza y se muestran los resultados, además de generar un fichero CSV con los mismos.	SÍ.
4	Ejecutar el programa especificando un fichero de texto en formato TXT	El fichero se analiza y se muestran los resultados, además de generar un fichero CSV con los mismos.	El fichero se analiza y se muestran los resultados, además de generar un fichero CSV con los mismos.	SÍ.
5	Ejecutar el programa especificando varios ficheros en diversos formatos (ODT,	Los ficheros se analizan y se muestran los resultados, además de	Los ficheros se analizan y se muestran los resultados, además de	SÍ

	DOCX, DOC o TXT). Menos de 5 ficheros.	generar ficheros CSV con los mismos.	generar ficheros CSV con los mismos.	
6	Ejecutar el programa especificando varios ficheros en diversos formatos (TXT, DOCX, DOC o ODT). Más de 5 ficheros.	Los ficheros se analizan y se muestran los resultados, además de generar ficheros CSV con los mismos.	El programa termina porque consume demasiados recursos. Para solucionarlo, se ha decidido que el cálculo de la similitud semántica sea opcional, puesto que este era el problema que causaba un excesivo uso de memoria RAM, haciendo que los 8GB de memoria contratados resultaran insuficientes.	NO.
6.1			Los ficheros se analizan y se muestran los resultados, además de generar ficheros CSV con los mismos.	SÍ.
7	Ejecutar el programa especificando uno o más ficheros en un formato distinto al permitido (los formatos permitidos son TXT, DOCX, DOC o ODT).	Se muestra un error de formato erróneo.	Se muestra un error de formato erróneo.	SÍ.
8	Ejecutar el programa sin especificar un fichero.	Se muestra un error de argumento erróneo.	Se muestra un error de argumento erróneo.	SÍ.
9	Clasificar los ficheros de texto según su complejidad.	Los ficheros de texto son clasificados según su complejidad.	El programa termina porque aparece un error de carga de modelo. El directorio era incorrecto.	NO.
			Los ficheros de texto son clasificados según su complejidad.	SÍ.

7.2 VERIFICACIÓN DE LOS INDICADORES

Para probar los indicadores, se han comparado con otras herramientas que ofrecen resultados similares y, en caso de que algún indicador no pudiera ser comparado porque éste no es calculado en ninguna otra herramienta, se ha validado de manera manual o utilizando otros métodos.

Por lo tanto, aquellos indicadores que son calculados por otras herramientas similares, pueden ser comparados. Para ello, se ha introducido un texto en inglés en cada una de las herramientas, y se han comparado los resultados con los obtenidos por la aplicación *AzterTest.*

A continuación, se presentan todos los indicadores que han sido verificados y validados.

Tal y como se puede observar en la Tabla 35, la mayoría de indicadores generales pueden ser comparados con las herramientas *Coh-Metrix* y *Textalyser*, y se obtienen resultados similares. Sin embargo, estas herramientas no realizan el cálculo para los siguientes indicadores:

- *Number of words with punctuation (total)*
- *Number of paragraphs (incidence per 1000 words)*
- *Number of sentences (incidence per 1000 words)*
- *Standard deviation of length of paragraphs*
- *Number of words (length) of sentences without stopwords (mean)*
- *Number of words (length) of sentences without stopwords (standard deviation)*
- *Mean number of letters (length) in words without stopwords*
- *Standard deviation of the mean number of letters in words without stopwords*
- *Mean number of letters (length) in lemmas*
- *Standard deviation of the mean number of letters in lemmas*

Por esa razón, se ha realizado la comprobación de estos indicadores de manera manual. Es decir, se ha introducido un texto de menor tamaño, y se ha realizado el cálculo de los indicadores comprobando individualmente que cada uno de ellos estuviera bien calculado.

Indicador	**AzterTest**	**Coh-Metrix**	**Textalyser**
Number of words (total)	564	565	564
Number of distinct words (total)	287	-	290
Number of words with punctuation (total)	655	-	-
Number of paragraphs (total)	19	19	-
Number of paragraphs (incidence per 1000 words)	33.6879	-	-

Number of sentences (total)	32	33	34
Number of sentences (incidence per 1000 words)	56.7376	-	-
Length of paragraphs (mean)	1.7369	1.737	-
Standard deviation of length of paragraphs	1.0683	1.098	-
Number of words (length) in sentences (mean)	17.5	17.121	17.09
Number of words (length) in sentences (standard deviation)	9.5982	10.208	-
Number of words (length) of sentences without stopwords (mean)	12.9375	-	-
Number of words (length) of sentences without stopwords (standard deviation)	7.9331	-	-
Mean number of syllables (length) in words	1.6525	1.616	1.78
Standard deviation of the mean number of syllables in words	0.9433	0.895	-
Mean number of letters (length) in words	5.0036	5.018	-
Standard deviation of the mean number of letters in words	2.6552	2.659	-
Mean number of letters (length) in words without stopwords	6.4796	-	-
Standard deviation of the mean number of letters in words without stopwords	2.5199	-	-
Mean number of letters (length) in lemmas	4.7179	-	-
Standard deviation of the mean number of letters in lemmas	2.5346	-	-

Tabla 35. Verificación de indicadores generales.

La densidad léxica se ha verificado mediante la comparación entre la herramienta *AzterTest* y *UsingEnglish*, que también calcula dicho parámetro. Como se puede observar en la Tabla 36, los resultados son similares, por lo que se considera que *AzterTest* calcula este valor de manera correcta.

Por otro lado, la densidad de sustantivos, verbos, adjetivos y adverbios se considera correcta puesto que tanto el cálculo del número de sustantivos, verbos, adjetivos y adverbios (Tabla 43) como el número de palabras (Tabla 35) se ha determinado que se calculan de manera correcta y, por consiguiente, estos indicadores también deben estar calculados de manera correcta, puesto que, tal y como se explicó en el apartado 5.2, se trata de un ratio.

Indicador	AzterTest	UsingEnglish
Lexical Density	0.5372	0.5186
Noun Density	0.2606	-
Verb Density	0.1436	-
Adjective Density	0.0904	-
Adverb Density	0.0426	-

Tabla 36. Verificación de la densidad léxica.

En la Tabla 37 se puede observar la comparación entre *AzterTest* y *Coh-Metrix* para los indicadores de riqueza léxica.

En cuanto al ratio de tipo-*token* (TTR), se puede realizar la comprobación mediante la comparación de resultados entre *AzterTest* y *Coh-Metrix* para los indicadores de TTR simple y TTR para lemas de palabras de contenido, pero para el resto de los casos se ha tenido que comprobar manualmente que los indicadores estuvieran bien calculados.

Por otro lado, aunque el indicador MTLD se ha podido verificar mediante la comparación de los resultados entre entre *AzterTest* y *Coh-Metrix*, para el caso de los índice de *Honoré* y *Maas*, se ha realizado la verificación comprobando que las fórmulas estuvieran bien calculadas, y utilizando un texto de menor tamaño para facilitar la tarea.

Indicador	AzterTest	Coh-Metrix
STTR (Simple Type-Token Ratio)	0.5089	0.519
CTTR (Content Type-Token Ratio)	0.7062	-
NTTR (Noun Type-Token Ratio)	0.6259	-
VTTR (Verb Type-Token Ratio)	0.7037	-
AdjTTR (Adj Type-Token Ratio)	0.7451	-

AdvTTR (Adv Type-Token Ratio)	0.75	-
LSTTR (Lemma Simple Type-Token Ratio)	0.5621	-
LCTTR (Lemma Content Type-Token Ratio)	0.6555	0.699
LNTTR (Lemma Noun Type-Token Ratio)	0.5714	-
LVTTR (Lemma Verb Type-Token Ratio)	0.5556	-
LAdjTTR (Lemma Adj Type-Token Ratio)	0.7451	-
LAdvTTR (Lemma Adv Type-Token Ratio)	0.75	-
Honoré Lexical Density	849.0506	-
Maas Lexical Density	0.0486	-
Measure of Textual Lexical Diversity (MTLD)	139.8171	145.375

Tabla 37. Verificación de los indicadores de riqueza léxica.

En la Tabla 38 se puede observar que tanto en *AzterTest* como en *Coh-Metrix*, *ReadabilityFormulas.com* y *Datayze* se obtiene unos resultados muy similares para los cuatro indicadores de lecturabilidad (excepto en *Coh-Metrix*, puesto que esta herramienta no calcula las métricas de Dale-Chall y SMOG. Por esa razón, se considera que la aplicación *AzterTest* realiza el cálculo de estos índices de manera correcta.

Indicador	**AzterTest**	**Coh-Metrix**	**ReadabilityFormulas**	**Datayze**
Flesch-Kincaid Grade level	10.783	10.156	10.3	10.68
Flesch readability ease	52.1456	52.744	52.8	48.94
Dale-Chall readability formula	8.3742	-	7.0409	8.15
Simple Measure Of Gobbledygook (SMOG) grade	8.8418	-	9.7	8.27

Tabla 38. Verificación de los indicadores de lecturabilidad.

Para determinar que *AzterTest* calcula correctamente el número de verbos en los distintos tiempos y modos, además del número de verbos irregulares en tiempo

pasado, se ha utilizado un texto de un tamaño pequeños (aproximadamente, 100 palabras), y se ha introducido un verbo de cada uno de los tipos y modos (es decir, un verbo en tiempo pasado, otro verbo en tiempo presente, etc.), y un verbo irregular en tiempo pasado. De esta manera, se ha conseguido comprobar que *AzterTest* reconoce correctamente cada uno de estos indicadores.

En la Tabla 39 se pueden observar los resultados que proporciona *AzterTest* para un texto de nivel intermedio de 564 palabras.

Indicador	**AzterTest**
Number of verbs in past tense	42
Number of verbs in past tense (incidence per 1000 words)	74.4681
Number of verbs in present tense	21
Number of verbs in present tense (incidence per 1000 words)	37.234
Number of verbs in future tense	2
Number of verbs in future tense (incidence per 1000 words)	3.5461
Number of verbs in indicative mood	44
Number of verbs in indicative mood (incidence per 1000 words)	78.0142
Number of verbs in imperative mood	0
Number of verbs in imperative mood (incidence per 1000 words)	0.0
Number of irregular verbs in past tense	26
Number of irregular verbs in past tense (incidence per 1000 words)	46.0993
Mean of irregular verbs in past tense in relation to the number of verbs in past tense	0.619

Tabla 39. Resultados de los indicadores de características morfológicas.

Tal y como se puede observar en la Tabla 40, los resultados de la aplicación *AzterTest* son los suficientemente cercanos a los de *Coh-Metrix*, por lo que se ha determinado que dichos resultados son correctos. De esta manera, quedan verificados los indicadores de pronombres.

Indicador	**AzterTest**	**Coh-Metrix**
Number of personal pronouns	20	19
Incidence score of pronouns (per 1000 words)	35.461	33.628

Number of pronouns in first person	8	7
Incidence score of pronouns in first person (per 1000 words)	14.1844	12.39
Number of pronouns in singular first person	3	2
Incidence score of pronouns in singular first person (per 1000 words)	5.3191	3.540
Number of pronouns, third person	12	10
Incidence score of pronouns, third person (per 1000 words)	21.2766	17.699

Tabla 40. Verificación de los indicadores de pronombres.

Para verificar que estos indicadores se han calculado correctamente, se ha comprobado que, efectivamente, las palabras consideradas raras estuvieran dentro del listado de palabras cuya *WordFrequency* fuese menor o igual que 4. Para ello, se ha imprimido el listado de palabras raras, se ha realizado el cálculo del número de apariciones de las distintas palabras dentro de dicho listado y se ha comprobado que los resultados de *AzterTest* coincidieran con los resultados realizados manualmente. En la Tabla 41 se puede observar los resultados que devuelve *AzterTest* para un texto de 564 palabras.

Indicador	**AzterTest**
The minimum word frequency per sentence	2.8784
Number of rare nouns (word frequency <= 4) :	26
Number of rare nouns (word frequency <= 4) (incidence per 1000 words)	46.0993
Number of rare adjectives (word frequency <= 4)	1
Number of rare adjectives (word frequency <= 4) (incidence per 1000 words)	1.773
Number of rare verbs (word frequency <= 4)	11
Number of rare verbs (word frequency <= 4) (incidence per 1000 words)	19.5035
Number of rare adverbs (word frequency <= 4):	1
Number of rare adverbs (word frequency <= 4) (incidence per 1000 words)	1.773
Number of rare content words (word frequency <= 4)	39

Number of rare content words (word frequency <= 4) (incidence per 1000 words):	69.1489
Number of distinct rare content words (word frequency <= 4)	27
Number of distinct rare content words (word frequency <= 4) (incidence per 1000 words):	47.8723
Mean of rare lexical words (word frequency <= 4)	12.8713
Mean of distinct rare lexical words (word frequency <= 4)	13.2353

Tabla 41. Resultados de los indicadores de WordFrequency.

Para verificar los indicadores de conocimiento del vocabulario (Tabla 42), puesto que no hay ninguna otra herramienta además de *AzterTest* que calcule esta información, se ha realizado de manera manual mediante la comprobación de la aparición de las palabras de los niveles A1, A2, B1, B2 y C1, utilizando el correspondiente listado para cada uno de ellos, y comprobando que los resultados dados por *AzterTest* fueran correctos.

Indicador	**AzterTest**
Number of A1 vocabulary in the text	185
Incidence score of A1 vocabulary (per 1000 words)	328.0142
Number of A2 vocabulary in the text	30
Incidence score of A2 vocabulary (per 1000 words)	53.1915
Number of B1 vocabulary in the text	15
Incidence score of B1 vocabulary (per 1000 words)	26.5957
Number of B2 vocabulary in the text	14
Incidence score of B2 vocabulary (per 1000 words)	24.8227
Number of C1 vocabulary in the text	6
Incidence score of C1 vocabulary (per 1000 words)	10.6383
Number of content words not in A1-C1 vocabulary	131
Incidence score of content words not in A1-C1 vocabulary (per 1000 words)	232.2695

Tabla 42. Resultados de los indicadores de conocimiento del vocabulario.

Coh-Metrix utiliza el *treebank* de *Penn* (McNamara, Graesser, McCarthy, & Cai, 2014) a la hora de etiquetar las palabras (POS tags), mientras que, como se ha mencionado anteriormente, *AzterTest* utiliza *NLP-Cube*, que utiliza el *UD Treebank* (Boros, Dumitrescu, & Burtica, 2018). Por esa razón, se considera que está dentro de lo normal, tal y como se puede observar en la Tabla 43, que algunos de los resultados de

los indicadores que miden las palabras de contenido (verbos, adverbios, adjetivos y sustantivos) no sean similares (por ejemplo, *Number of content words*). Por esa razón, se considera que los resultados de *AzterTest* son correctos. Además, algunos de los indicadores producen resultados muy similares a los de *Coh-Metrix* (por ejemplo, *Number of verbs in gerund form*).

Por otro lado, para este tipo de indicadores también ocurre que no se pueden comparar con los resultados de ninguna otra herramienta, puesto que no existe una herramienta que los calcule. Por esa razón, se ha realizado el cálculo de estos indicadores de manera manual y con textos de menor tamaño, con el objetivo de comprobar que estuvieran bien calculados. Por ejemplo, para el indicador *Mean of the number of levels of dependency tree*, se ha realizado la prueba con tres frases y se obtenido el árbol de dependencias de dichas frases. Después, se ha obtenido de la profundidad de dicho árbol, se ha hecho la media de las tres profundidades y se ha comprado con el resultado dado por *AzterTest*.

Indicador	**AzterTest**	**Coh-Metrix**
Number of content words	303	333
Number of content words (incidence per 1000 words)	537.234	589.381
Number of nouns	147	187
Number of nouns (incidence per 1000 words)	260.6383	330.974
Number of adjectives	51	55
Number of adjectives (incidence per 1000 words)	90.4255	99.115
Number of adverbs	24	23
Number of adverbs (incidence per 1000 words)	42.5532	38.938
Number of verbs	81	68
Number of verbs (incidence per 1000 words)	143.617	120.354
Left embeddedness (Mean of number of words before the main verb) (SYNLE)	4.4375	4.045
Number of decendents per noun phrase (mean)	4.4375	-
Number of modifiers per noun phrase (mean) (SYNNP)	0.5467	0.894
Mean of the number of levels of dependency tree (Depth)	5.3438	-
Number of subordinate clauses	32	-
Number of subordinate clauses (incidence per 1000 words)	56.7376	-

Number of relative subordinate clauses	4	-
Number of relative subordinate clauses (incidence per 1000 words)	7.0922	-
Mean of punctuation marks per sentence	2.6875	-
Number of propositions	96	-
Mean of the number of propositions per sentence	3.0	-
Mean of the number of VPs per sentence	2.5312	-
Mean of the number of NPs per sentence	6.6875	-
Noun phrase density, incidence (DRNP)	379.4326	391.150
Verb phrase density, incidence (DRVP)	143.617	189.381
Number of agentless passive voice verbs	2	3
Agentless passive voice density, incidence (DRPVAL)	3.5461	5.310
Number of negative words	3	3
Negation density, incidence (DRNEG)	5.31	5.31
Number of verbs in gerund form	3	4
Gerund density, incidence (DRGERUND)	5.3191	7.08
Number of verbs in infinitive form	6	7
Infinitive density, incidence (DRINF)	10.6383	12.389

Tabla 43. Verificación de los indicadores de características sintácticas.

Para verificar los indicadores de verbos en voz pasiva (es decir, el número de verbos en voz pasiva y la media de verbos en voz pasiva), se ha utilizado la herramienta *Datayze*, puesto que *Coh-Metrix* no calcula esta métrica. Como se puede observar en la Tabla 44, los resultados obtenidos en ambas herramientas (*AzterTest* y *Datayze*) son lo suficientemente similares como para verificar que son correctos.

Indicador	**AzterTest**	**Datayze**
Number of passive voice verbs	3	4
Number of passive voice verbs (incidence per 1000 words)	5.3191	7.0922
Mean of passive voice verbs	0.0053	0.0071

Tabla 44. Verificación de los indicadores de voz pasiva.

Como se puede observar en la Tabla 45, los resultados obtenidos por *Coh-Metrix* difieren bastante con respecto a los obtenidos por *AzterTest*. Esto puede deberse a que *Coh-Metrix* utiliza una versión anterior de *WordNet*, mientras que *AzterTest* utiliza la última disponible. Por lo tanto, el *WordNet* utilizado por *AzterTest* es más actual y, en consecuencia, es probable que contenga más definiciones y significados de las palabras y, por esa razón, *AzterTest* proporciona valores más altos. Por ello, se considera que los indicadores de polisemia e hiperonimia son correctos.

Indicador	AzterTest	Coh-Metrix
Mean values of polysemy in the WordNet lexicon	6.6567	4.022
Mean hypernym values of verbs in the WordNet lexicon	1.716	1.567
Mean hypernym values of nouns in the WordNet lexicon	7.2381	5.795
Mean hypernym values of nouns and verbs in the WordNet lexicon	5.2763	3.986

Tabla 45. Verificación de los indicadores de polisemia e hiperonimia.

En cuanto a los indicadores de superposición, se puede observar en la Tabla 46 que los valores de ambas herramientas (*AzterTest* y *Coh-Metrix*) es similar, por lo que se considera que los resultados obtenidos son correctos.

Indicador	AzterTest	Coh-Metrix
Noun overlap, adjacent sentences, binary, mean (CRFNOl)	0.2	0.213
Noun overlap, all of the sentences in a paragraph or text, binary, mean (CRFNOa)	0.1429	0.193
Argument overlap, adjacent sentences, binary, mean (CRFAOl)	0.5	0.565
Argument overlap, all of the sentences in a paragraph or text, binary, mean (CRFAOa)	0.3571	0.395

Stem overlap, adjacent sentences, binary, mean (CRFSOl)	0.3	0.391
Stem overlap, all of the sentences in a paragraph or text, binary, mean (CRFSOa)	0.2143	0.249
Content word overlap, adjacent sentences, proporcional, mean (CRFCWO1)	0.0248	0.028
Content word overlap, adjacent sentences, proporcional, standard deviation (CRFCWO1d)	0.0403	0.036
Content word overlap, all of the sentences in a paragraph or text, proporcional, mean (CRFCWOa)	0.0177	0.071
Content word overlap, all of the sentences in a paragraph or text, standard deviation (CRFCWOad)	0.0359	0.093

Tabla 46. Verificación de los indicadores de superposición.

Debido a que para calcular los indicadores de similitud semántica *Coh-Metrix* no utiliza el mismo método de *AzterTest*, se ha decidido que, en lugar de comparar los valores para un único texto (puesto que, evidentemente, no se obtendrían resultados similares), se realice el cálculo para dos textos distintos y, si en una herramienta el valor crece o decrece, en la otra también debería crecer o decrecer.

Es decir, tal y como se puede observar en la Tabla 47, algunos de los valores de los indicadores (por ejemplo, *Semantic Similarity between adjacent sentences*), del primer fichero (ID Prueba: 1) crecen respecto al segundo fichero (ID Prueba: 2) y, como en *Coh-Metrix* también crecen (aunque no sea un crecimiento de proporcionalidad lineal), se considera que los valores calculados por *AzterTest* son correctos. Ocurre lo mismo para los indicadores que decrecen (por ejemplo, *Semantic Similarity between all possible pairs of sentences in a paragraph*).

Indicador	**ID Prueba**	**AzterTest**	**Coh-Metrix**
Semantic Similarity between adjacent sentences (mean)	1	0.227	0.115
	2	0.2357	0.213

Semantic Similarity between all possible pairs of sentences in a paragraph (mean)	1	0.0901	0.083
	2	0.0285	0.025
Semantic Similarity between adjacent paragraps (mean)	1	0.5809	0.201
	2	0.6766	0.268
Semantic Similarity between adjacent sentences (standard deviation)	1	0.2304	0.127
	2	0.2808	0.199
Semantic Similarity between all possible pairs of sentences in a paragraph (standard deviation)	1	0.1775	0.105
	2	0.1103	0.096
Semantic Similarity between adjacent paragraps (standard deviation)	1	0.2201	0.154
	2	0.2195	0.143

Tabla 47. Verificación de indicadores de similitud semántica.

Por último, tal y como se puede observar en la Tabla 48, los valores para la incidencia de los distintos conectores en el texto son similares tanto en *AzterTest* como en *Coh-Metrix*, aunque para algunos casos (especialmente, los conectores causales), es resultado varía bastante. Esto puede deberse a que *Coh-Metrix* no utiliza el mismo listado de conectores que *AzterTest*, por lo que es de esperar que se obtengan resultados como ese. Por esa razón, se considera que estos indicadores son correctos.

Indicador	**AzterTest**	**Coh-Metrix**
Number of connectives (incidence per 1000 words)	97.5177	106.195
Causal connectives (incidence per 1000 words)	12.4113	35.398
Logical connectives (incidence per 1000 words)	54.9645	61.947
Adversative connectives (incidence per 1000 words)	12.4113	17.699
Temporal connectives (incidence per 1000 words)	17.7305	14.159
Conditional connectives (incidence per 1000 words)	0.0	0

Tabla 48. Verificación de los indicadores de la incidencia de conectores.

7.3 PRUEBAS DEL CLASIFICADOR

Para testear el clasificador, se han realizado varias pruebas con el objetivo de encontrar el modelo que mejores resultados proporcione. En concreto, se han entrenado un total de tres modelos usando tres conjuntos de datos distintos, utilizando tanto los resultados de *AzterTest* como los de *Coh-Metrix*, de manera que se pudieran realizar comparaciones entre ellos.

El rendimiento del clasificador multiclase se evalúa, por lo tanto, utilizando unos conjuntos de datos balanceados y, para comparar dicho rendimiento, se utiliza la *accuracy*. La *accuracy* es una de las medidas más utilizadas para comparar el desempeño de clasificadores multiclase que utilizan conjuntos de datos balanceados (Carrillo, Castellanos, & Brodersen).

7.3.1 Conjunto de datos 1: Resultados de *AzterTest* – Todos los indicadores

En primer lugar, se ha entrenado un modelo utilizando todos los indicadores que se calculan utilizando la aplicación desarrollada. Es decir, para crear el conjunto de datos, se ha ejecutado *AzterTest* sobre el *corpus* de *OneStopEnglish*, tal y como se explica en el apartado 6.4 y, por tanto, se ha entrenado utilizando todos los indicadores.

Puesto que se utiliza el método de Chi-cuadrado para encontrar los K mejores atributos, se han realizado distintas pruebas para comprobar cuál es la K que mejores resultados produce. Es decir, con qué cantidad de atributos se consiguen mejores resultados. Los resultados que se han obtenido se pueden observar en la Tabla 49.

K	**Accuracy** (en %)
40	82.2368
60	83.1140
80	83.1140
100	**83.7719**
120	83.5526
140	83.3334
Todos	83.3334

Tabla 49. Accuracy obtenida según el número de atributos (K). Conjunto 1.

Como se puede observar, la mejor *accuracy* (obtenida mediante *10-Fold Cross Validation*, tal y como se explicó en el apartado 6.4) se consigue cuando se utilizan 100

atributos. Por lo tanto, el modelo se ha entrenado utilizando los 100 mejores atributos, y la mejor combinación de parámetros se consigue con el algoritmo *Gradient Boosting Classifier* (scikit-learn, s.f.). Dichos parámetros son los siguientes:

- `'learning_rate': 0.15`
- `'max_depth': 8`
- `'max_features': 'log2'`
- `'n_estimators': 500`

Como es el clasificador que mejores resultados devuelve en el *10-Fold Cross Validation* (83,7719% de *accuracy*), este será el modelo utilizado por el programa para predecir la dificultad de los textos.

De esta manera, para el conjunto de pruebas (que corresponde al 20% del conjunto total) se han obtenido los resultados que se pueden observar en la matriz de confusión de la Tabla 50.

		Predicción		
		Advanced	***Intermediate***	***Elementary***
Real	***Advanced***	34	2	1
	Intermediate	1	23	1
	Elementary	13	5	31

Tabla 50. Matriz de confusión del Conjunto de datos 1.

Por lo tanto, la precisión (*accuracy*) del clasificador en el conjunto de pruebas, para este caso, es del 79.28% aproximadamente.

7.3.2 Conjunto de datos 2: Resultados de *AzterTest* – Solo indicadores en común

Por otro lado, se ha entrenado un modelo utilizando únicamente aquellos indicadores que se calculen tanto en *Coh-Metrix* como en *AzterTest*. Es decir, para crear el conjunto de datos, se ha ejecutado *AzterTest*, de nuevo, sobre el *corpus* de *OneStopEnglish* pero, en este caso, no se han calculado todos los indicadores, sino que se han calculado solamente aquellos que se tengan en común con *Coh-Metrix*. De esta manera, se podrán realizar comparaciones con los resultados de ambas herramientas.

De igual forma que para el conjunto de datos anterior, se han realizado pruebas con distintos números de *K* para encontrar los mejores atributos mediante el método de Chi-cuadrado. En la Tabla 51 se pueden observar los resultados obtenidos.

K	**Accuracy** (en %)
30	74,5139
35	75,5833
40	**76,4306**
Todos	73,6250

Tabla 51. Accuracy obtenida según el número de atributos (K). Conjunto 2.

Como se puede observar, en este caso, el mejor resultado se obtiene cuando la cantidad de atributos (*K*) a conservar es 40. A partir de ese valor, la precisión disminuye. Por tanto, se utiliza el método de Chi-cuadrado para conservar los 40 mejores atributos del conjunto de datos y, posteriormente, entrenar el modelo.

El mejor algoritmo para *K=40* resulta ser el *Random Forest Classifier* (scikit-learn, s.f.) con los siguientes parámetros:

- `'criterion': ‘gini`
- `'max_features': 'log2'`
- `'n_estimators': 1000`

Por lo tanto, para el conjunto de pruebas (que corresponde al 20% del conjunto total) con los resultados de *AzterTest* (y adaptados a los indicadores que tienen en común con *Coh-Metrix*) se han obtenido los resultados que se pueden observar en la matriz de confusión de la Tabla 52.

		Predicción		
		Advanced	***Intermediate***	***Elementary***
Real	***Advanced***	28	9	0
	Intermediate	9	23	5
	Elementary	1	7	29

Tabla 52. Matriz de confusión del Conjunto de datos 2.

Por lo tanto, la precisión (*accuracy*) del clasificador para este caso, es del 72,07% aproximadamente.

7.3.3 Conjunto de datos 3: Resultados de *AzterTest* – Solo indicadores distintos

Para realizar comparaciones, se ha creado otro conjunto de datos. Esta vez, nuevamente, se ha ejecutado *AzterTest* sobre el *corpus* de *OneStopEnglish* pero, en este caso, solamente se han calculados los indicadores que son únicos de *AzterTest*.

Para este conjunto de datos, no se ha realizado una selección de atributos, sino que se ha decidido mantenerlos todos. La *accuracy* que se obtiene (tras realizar *10-Fold Cross Validation*) es 76,4722%.

En este caso, los mejores parámetros del mejor algoritmo (*Gradient Boosting Classifier* (scikit-learn, s.f.)) son:

- `'learning_rate': 0.05`
- `'max_depth': 8`
- `'max_features': 'log2'`
- `'n_estimators': 1750`

Si se ejecuta el modelo para el conjunto de pruebas (que corresponde al 20% del conjunto total) se obtiene la matriz de confusión de la Tabla 53.

		Predicción		
		Advanced	***Intermediate***	***Elementary***
Real	***Advanced***	31	5	1
	Intermediate	5	30	2
	Elementary	0	6	31

Tabla 53. Matriz de confusión del Conjunto de datos 3.

Por lo tanto, la precisión (*accuracy*) del clasificador para este caso, es del 82.88% aproximadamente.

7.3.4 Conjunto de datos 4: Resultados de *Coh-Metrix* – Todos los indicadores

Para poder comparar los resultados de *Coh-Metrix* con los de *AzterTest*, se ha ejecutado la herramienta *Coh-Metrix* sobre el *corpus* de *OneStopEnglish* y, después, se han conservado aquellos atributos que se tuvieran en común con *AzterTest*. De esta manera, por tanto, se podrán realizar comparaciones con los resultados de ambas herramientas.

De nuevo, se han realizado pruebas con distintos números de *K* para encontrar los mejores atributos mediante el método de *ANOVA F-value* (*f_classif* en *sklearn*), puesto que el método de Chi-cuadrado no es capaz de trabajar con valores negativos, y *Coh-Metrix* tiene algunas métricas que producen valores negativos. En la Tabla 54 se pueden observar los resultados obtenidos.

K	**Accuracy** (en %)
40	79.6053
60	79.1667
80	79.1667
100	**79.8246**
Todos	79.1667

Tabla 54. Accuracy obtenida según el número de atributos (K). Conjunto 4.

Como se puede observar, en este caso, el mejor resultado se obtiene cuando la cantidad de atributos (*K*) a conservar es igual a 100. Es decir, la *accuracy* es mayor cuando se trabaja sobre los 100 mejores atributos.

El mejor algoritmo para *K* es igual a todos los atributos resulta ser, de nuevo, el *Random Forest Classifier* (scikit-learn, s.f.) con los siguientes atributos:

- `'criterion': entropy`
- `'max_features': 'auto'`
- `'n_estimators': 100`

Por lo tanto, para el conjunto de pruebas (que corresponde al 20% del conjunto total) con los resultados de *Coh-Metrix* (y adaptados a los indicadores que tienen en común con *AzterTest*) se han obtenido los resultados que se pueden observar en la matriz de confusión de la Tabla 55.

		Predicción		
		Advanced	***Intermediate***	***Elementary***
Real	***Advanced***	33	4	0
	Intermediate	7	30	0
	Elementary	0	5	32

Tabla 55. Matriz de confusión del Conjunto de datos 4.

Por lo tanto, la precisión (*accuracy*) del clasificador para este caso, es del 85.59% aproximadamente.

7.3.5 Conjunto de datos 5: Resultados de *Coh-Metrix* – Solo indicadores en común

También se ha entrenado un modelo utilizando únicamente aquellos indicadores que se calculen tanto en *Coh-Metrix* como en *AzterTest*. Es decir, para crear el conjunto de datos, se ha ejecutado *Coh-Metrix*, de nuevo, sobre el *corpus* de *OneStopEnglish* pero

únicamente se han calculado solamente aquellos que se tengan en común con *AzterTest*. De esta manera, se podrán realizar comparaciones con los resultados de ambas herramientas.

De igual forma que para el conjunto de datos anterior, se han realizado pruebas con distintos números de *K* para encontrar los mejores atributos mediante el método de *ANOVA F-value*. En la Tabla 56 se pueden observar los resultados obtenidos.

K	**Accuracy** (en %)
30	73,5833
35	73,3611
40	74,0139
Todos	**75,2917**

Tabla 56. Accuracy obtenida según el número de atributos (K). Conjunto 5.

Como se puede observar, en este caso, el mejor resultado se obtiene cuando la cantidad de atributos (*K*) a conservar es igual al número total de atributos. Es decir, no se hará ninguna selección de atributos, puesto que el mejor resultado se obtiene con todos los atributos.

Por tanto, se entrena el modelo para todos los atributos y, tras realizar la busca de los parámetros más óptimos con *GridSearchCV* (tal y como se explicó en el apartado 6.4), el mejor algoritmo resulta ser el *Gradient Boosting Classifier* (scikit-learn, s.f.) con los siguientes parámetros:

- `'learning_rate': 0.005`
- `'max_depth': 3`
- `'max_features': 'sqrt'`
- `'n_estimators': 500`

Por lo tanto, para el conjunto de pruebas (que corresponde al 20% del conjunto total) con los resultados de *Coh-Metrix* (y adaptados a los indicadores que tienen en común con *AzterTest*) se han obtenido los resultados que se pueden observar en la matriz de confusión de la Tabla 57.

		Predicción		
		Advanced	***Intermediate***	***Elementary***
Real	***Advanced***	31	6	0
	Intermediate	8	25	4
	Elementary	0	6	31

Tabla 57. Matriz de confusión del Conjunto de datos 5.

Por lo tanto, la precisión (*accuracy*) del clasificador para este caso, es del 78,38% aproximadamente.

7.3.6 Conjunto de datos 6: Resultados de *Coh-Metrix* – Solo indicadores distintos

Para realizar comparaciones, se ha creado este último conjunto de datos en el que, nuevamente, se ha ejecutado *Coh-Metrix* sobre el *corpus* de *OneStopEnglish* pero, en este caso, solamente se han calculados los indicadores que son únicos de *Coh-Metrix* respecto a *AzterTest*.

Para este conjunto de datos, no se ha realizado una selección de atributos, sino que se ha decidido mantenerlos todos. La *accuracy* obtenida (tras realizar *10-Fold Cross Validation*) es 64,5972%, la cual, si es comparada con la de *AzterTest* (76,4722%), es un resultado bastante pobre.

Para este caso, el mejor algoritmo resulta ser el *Ada Boost Classifier* (scikit-learn, s.f.) con los siguientes parámetros:

- `'learning_rate': 0.1`
- `'n_estimators': 500`

Una vez entrenado el modelo, si se ejecuta para el conjunto de pruebas (que corresponde al 20% del conjunto total) se obtiene la matriz de confusión de la Tabla 58.

		Predicción		
		Advanced	***Intermediate***	***Elementary***
Real	***Advanced***	27	10	0
	Intermediate	7	29	1
	Elementary	1	9	27

Tabla 58. Matriz de confusión del Conjunto de datos 6.

Por lo tanto, la precisión (*accuracy*) del clasificador para este caso, es del 74,77% aproximadamente.

7.3.7 Comparación y conclusiones

En primer lugar, cabe destacar que el número de indicadores que tiene *AzterTest* es 148, mientras que *Coh-Metrix* dispone de 106. Es decir, *AzterTest* calcula 42 indicadores más que *Coh-Metrix*, por lo que tiene en cuenta más factores y, por tanto, es más robusto a la hora de evaluar la complejidad de un texto. Por otro lado, los indicadores que tienen en común son 49, y son los que se han utilizado para crear los conjuntos de datos 2 y 5.

Tras haber hallado los mejores atributos y algoritmos para cada uno de los conjuntos de datos, se pueden utilizar los resultados obtenidos para compararlos y sacar conclusiones.

Cuando se utilizan todos los indicadores (conjuntos de datos 1 y 4), los resultados que se obtienen son mejores en *AzterTest*, tal y como se puede observar en la Tabla 59.

		Accuracy
Coh-Metrix	10-Fold (80%)	79,8246
	Test (20%)	85,5856
AzterTest	10-Fold (80%)	83,7719
	Test (20%)	79,2793

Tabla 59. Conjunto de datos 1 vs. Conjunto de datos 4

Por lo tanto, con los resultados de los indicadores de *AzterTest*, se obtienen mejores resultados que con los de *Coh-Metrix* para el *10-Fold Cross Validation* y, puesto que este valor de *accuracy* es el mejor que se ha obtenido para los conjuntos de datos de *AzterTest*, el modelo que se utiliza en la aplicación para predecir los textos se ha entrenado utilizando este conjunto de datos.

Por otro lado, cuando únicamente se utilizan los indicadores que ambas herramientas tienen en común (conjuntos de datos 2 y 5), los resultados que se obtienen son también mejores en *AzterTest*, tal y como se puede observar en la Tabla 60.

		Accuracy
Coh-Metrix	10-Fold (80%)	75,2917
	Test (20%)	78,3784
AzterTest	10-Fold (80%)	76,4306
	Test (20%)	72,0721

Tabla 60. Conjunto de datos 2 vs. Conjunto de datos 5

De nuevo, *AzterTest* produce mejores resultados cuando se utilizan los 49 indicadores que tienen en común ambas herramientas. Esto puede ser un indicio de

que *AzterTest* es más preciso a la hora de calcular estos indicadores. Aun así, los resultados son bastante similares (excepto para el conjunto de pruebas, en el que se producen mejores resultados con *Coh-Metrix*), por lo que no se puede afirmar este hecho con rotundidad.

Cuando se utilizan los indicadores que son únicos para cada herramienta (conjuntos de datos 3 y 6), los resultados que se obtienen son notablemente mejores en *AzterTest*, tal y como se puede observar en la Tabla 59.

		Accuracy
Coh-Metrix	10-Fold (80%)	64,5972
	Test (20%)	74,7748
AzterTest	10-Fold (80%)	76,4722
	Test (20%)	82,8828

Tabla 61. Conjunto de datos 3 vs. Conjunto de datos 6

En este último caso, *AzterTest* produce mejores resultados que *Coh-Metrix*. Esto puede ser un indicio de que los indicadores que calcula *AzterTest* son más determinantes que las de *Coh-Metrix*. Es decir, las métricas únicas (respecto a *Coh-Metrix*) que calcula *AzterTest* tienen más influencia sobre el valor de la categoría (la dificultad del texto) que las de *Coh-Metrix*. Por otro lado, esto puede deberse a que *AzterTest* dispone de 42 indicadores más que *Coh-Metrix*, por lo que es evidente que, como se tienen más métricas, es más probable que se obtengan mejores resultados.

En conclusión, generalmente los mejores resultados se obtienen con *AzterTest*, aunque la diferencia con los de *Coh-metrix* es tan pequeña que no se puede afirmar que exista una mejora notable, pero si podría decirse que ambas herramientas están igualadas, puesto que producen resultados similares.

7.4 PRUEBAS DE LA APLICACIÓN WEB

En primer lugar, cabe destacar que la aplicación web se encuentra accesible desde la dirección http://178.128.198.190/ y puede ser probada por cualquier usuario que acceda a dicho enlace.

Por otro lado, con la finalidad de comprobar el correcto funcionamiento de las funcionalidades de la aplicación web, se han realizado las pruebas expuestas a continuación.

ID	Prueba realizada	Resultado esperado	Resultado obtenido	¿Correcto?
1	Pulsar el botón *Analizar* sin introducir ningún fichero.	Se muestra un mensaje de error que advierte al usuario de que debe introducir al menos un fichero de texto.	Se muestra un mensaje de error que advierte al usuario de que debe introducir al menos un fichero de texto.	SÍ.
2	Pulsar el botón *Analizar* habiendo introducido un fichero en formato ODT	El fichero se analiza y se muestran los resultados en una tabla, además de aparecer un enlace para la descarga de los resultados en formato CSV.	El fichero se analiza y se muestran los resultados en una tabla, además de aparecer un enlace para la descarga de los resultados en formato CSV.	SÍ.
3	Pulsar el botón *Analizar* habiendo introducido un fichero en formato DOCX	El fichero se analiza y se muestran los resultados en una tabla, además de aparecer un enlace para la descarga de los resultados en formato CSV.	El fichero se analiza y se muestran los resultados en una tabla, además de aparecer un enlace para la descarga de los resultados en formato CSV.	SÍ.
4	Pulsar el botón *Analizar* habiendo introducido un fichero en formato DOC	El fichero se analiza y se muestran los resultados en una tabla, además de aparecer un enlace para la descarga de los resultados en formato CSV.	El fichero se analiza y se muestran los resultados en una tabla, además de aparecer un enlace para la descarga de los resultados en formato CSV.	SÍ.
5	Pulsar el botón *Analizar* habiendo introducido un fichero en formato TXT	El fichero se analiza y se muestran los resultados en una tabla, además de aparecer un enlace para la descarga de los resultados en formato CSV.	El fichero se analiza y se muestran los resultados en una tabla, además de aparecer un enlace para la descarga de los resultados en formato CSV.	SÍ.
6	Pulsar el botón *Analizar* habiendo introducido uno o más ficheros en un formato distinto al permitido (los formatos permitidos son TXT, DOCX, DOC o ODT).	Se muestra un mensaje de error en el que se especifica que se han seleccionado ficheros en un formato no permitido.	Se muestra un mensaje de error en el que se especifica que se han seleccionado ficheros en un formato no permitido.	SÍ.
7	Descargar resultados tras haber analizado uno o más ficheros.	Se descargan los resultados en formato CSV.	No se encuentra, en el servidor, el fichero a descargar. No se había especificado	NO.

			correctamente el directorio.	
			Se descargan los resultados en formato CSV.	SÍ.
8	Pulsar el botón *Más información*.	Se muestra información sobre la página web y las distintas métricas.	Se muestra información sobre la página web y las distintas métricas.	SÍ.

8 Conclusiones y trabajo futuro

En este capítulo final, se busca recapitular y recoger las conclusiones que han surgido tras la finalización de este proyecto, analizando qué trabajo se ha realizado, qué resultados se han obtenido, y comparando la planificación inicial con lo que realmente se ha cumplido. Además, se mencionan qué posibles mejoras pueden realizarse en la aplicación, de cara a una posible futura ampliación de la misma.

8.1 CUMPLIMIENTO DE LOS OBJETIVOS

Cuando dio comienzo el proyecto, se marcaron una serie de objetivos que se querían cumplir y, ahora, como se ha llegado al final del mismo, se procede a analizar y comprobar cuáles de estos objetivos se han cumplido. Dichos objetivos son los siguientes:

- *Desarrollar una aplicación que permita la obtención de métricas sólidas y validadas referentes a la idoneidad de textos escritos en inglés con objetivos didácticos en una etapa educativa primaria o secundaria. Además, la aplicación debe proporcionar información acerca de dichas métricas, para que los usuarios que utilicen la aplicación puedan ser capaces de interpretarlas correctamente.*

Se considera que este objetivo se ha cumplido de manera satisfactoria, ya que la aplicación es capaz de analizar documentos de texto y mostrarle al usuario, de una manera limpia y clara, métricas que sirven para evaluar la complejidad de material didáctico en inglés, puesto que la principal segunda lengua (L2) en el sistema educativo español es el inglés. Además, se muestra una tabla que informa al usuario sobre dichas métricas y cómo deben ser interpretadas.

- *La aplicación, además, hará uso de un clasificador creado en base a las métricas desarrolladas, de manera que será capaz de determinar la complejidad de un texto*.

La aplicación, en base a las métricas que se calculan, es capaz de predecir la complejidad de un texto. Para ello, se ha utilizado un clasificador supervisado entrenado con un *corpus* en el que se dividen cientos de artículos en tres categorías diferentes: *elementary*, *intermediate* y *advanced*. Cada una de estas categorías corresponde, por tanto, a un nivel de dificultad distinto, siendo *elementary* el nivel más bajo y *advanced* el más alto. En definitiva, se puede decir que este objetivo se ha cumplido satisfactoriamente.

- *Crear una aplicación web intuitiva, de manera que no se requieran amplios conocimientos informáticos para poder utilizar el sistema de una manera simple y eficaz.*

La aplicación web se ha desarrollado intentando que ésta fuera lo más sencilla posible, de manera que un usuario inexperto no tuviera problemas para analizar ficheros de texto. Para comprobar que se ha cumplido el objetivo, se ha pedido a usuarios con conocimientos básicos de informática que utilizasen la aplicación web para analizar ficheros de texto y todos ellos han logrado hacerlo sin ninguna complicación. Por lo tanto, por lo que se puede considerar que el sistema es lo suficientemente intuitivo y explicativo para que este tipo de usuarios puedan utilizarlo sin ningún problema.

- *Adquirir experiencia con el lenguaje de programación Python. Pese a que se dispone de nociones básicas de este lenguaje, se pretende ampliar estos conocimientos y terminar el proyecto conociendo bien el lenguaje.*

Se considera que, durante el desarrollo del proyecto y, especialmente, al comienzo del mismo (puesto que, al principio, se tuvieron que estudiar y aprender las nociones básicas del lenguaje), se ha adquirido suficiente experiencia en el lenguaje de programación *Python* como para volver a desarrollar un programa de estas características. Por lo tanto, se considera que este objetivo personal, definido con la finalidad de aumentar conocimientos de *Python*, ha sido cumplido.

8.2 APARICIÓN DE RIESGOS Y COMPLICACIONES

Durante el desarrollo del proyecto, se cumplió el riesgo *Fallo o recursos insuficientes en el servidor*. Debido a la falta de memoria RAM, el servidor no era capaz de manejar algunos procesos que eran necesarios para el correcto funcionamiento de la aplicación y, por lo tanto, se tuvo que aplicar el plan de contingencia definido. Por ello, se contrató un nuevo servidor remoto y se restauró la copia de seguridad del servidor anterior, de manera que la aplicación web pudiera volver a ponerse en marcha. En este caso, el servidor pasaría de tener 3GB de memoria RAM a tener 8GB. Gracias a esta mejora, se pudo continuar con el desarrollo del proyecto sin más inconvenientes.

Por otro lado, más tarde, cerca de la finalización del desarrollo del proyecto, surgieron algunas complicaciones para las cuales se tuvieron que aplicar soluciones que modificaban el diseño inicial de la aplicación y, por consiguiente, de la aplicación web. Estas complicaciones surgieron con el cálculo de la similitud semántica entre frases y párrafos mediante el uso de *Universal Sentence Encoders*. El problema estaba en que esta herramienta consumía grandes cantidades de recursos, para lo cual no

estaban preparados ni el servidor utilizado para alojar la página web, ni el propio equipo donde se desarrollaba la aplicación. Esto es, la aplicación, en caso de que se introdujeran más de 5 ficheros, podría llegar a consumir más de 8 GB de memoria RAM, lo cual hacía que el equipo se ralentizara y el sistema operativo detuviera la ejecución de la aplicación. Además, este proceso podría llegar a tardar más de 10 minutos, en función de la cantidad de ficheros seleccionada. En consecuencia, la solución fue hacer que los indicadores de similitud semántica fuesen opcionales, y limitar a 5 el número de ficheros a analizar en caso de que el usuario decidiera calcular la similitud semántica de sus textos. Con esta solución, no se vio afectada en exceso la duración final de la tarea y, además, no fue necesario contratar un servidor con más memoria RAM, de manera que el coste del proyecto no aumentó más de lo necesario.

8.3 ANÁLISIS ENTRE PLANIFICACIÓN ESTIMADA Y REAL

Pese a que los objetivos del proyecto se han cumplido de manera satisfactoria, no ha sido posible ajustarse a la planificación inicial. Debido a que algunas de las tareas sufrieron variaciones temporales, la duración total del proyecto se vio afectada. En la Tabla 62 se refleja la duración real que han tenido las distintas tareas del proyecto, para compararla con la duración que se estimó al comienzo del mismo.

ID	Tarea	Duración estimada	Duración total estimada	Duración real	Duración total real
1.1	Reuniones con el director del proyecto	20 h	42 h	24 h	45 h
1.2	Definición de los objetivos	4 h		4 h	
1.3	Definición de tareas a realizar	12 h		12 h	
1.4	Selección de herramientas a utilizar	6 h		5 h	
2.1	Aprendizaje de Python	34 h	92 h	40 h	100 h
2.2	Aprendizaje de las herramientas a utilizar	16 h		14 h	
2.3	Búsqueda y lectura de información	42 h		46 h	
3.1	Definición de las funcionalidades	12 h	28 h	12 h	24 h
3.2	Definición de Casos de Uso	6 h		4 h	

3.3	Definición del Modelo de Dominio	10 h		8 h	
4.1	Selección de indicadores	16 h	28 h	24 h	34 h
4.2	Realización de Diagrama de Clases	6 h		4 h	
4.3	Selección de paquetes y módulos	4 h		4 h	
4.4	Diseño interfaz de la aplicación web	2 h		2 h	
5.1	Indicadores generales	10 h	170 h	10 h	198 h
5.2	Indicadores de riqueza léxica	20 h		22 h	
5.3	Indicadores de lecturabilidad	20 h		24 h	
5.4	Indicadores morfológicos	10 h		10 h	
5.5	Indicadores de frecuencia de palabra	5 h		5 h	
5.6	Indicadores de conocimiento del vocabulario	5 h		4 h	
5.7	Indicadores sintácticos	10 h		12 h	
5.8	Indicadores de cohesión	20 h		28 h	
5.9	Clasificador	15 h		12 h	
5.10	Desarrollo de la aplicación web	35 h		35 h	
5.11	Realización de pruebas	20 h		36 h	
6.1	Redacción del DOP	25 h	110 h	25 h	115 h
6.2	Redacción de la memoria	85 h		90 h	

Tabla 62. Planificación temporal estimada vs. real

Como se puede observar, a este proyecto se le han dedicado un total de 516 horas, mientras que lo que se estimó fue que la duración total iba a ser de 470 horas. Esto supone que se han dedicado 46 horas más de las previstas, es decir, un incremento del 9,79% sobre la planificación temporal inicial del proyecto.

Las tareas que más variación temporal han sufrido se encuentran en el módulo 5, que corresponde al desarrollo de la aplicación. Principalmente, en la tarea de *Indicadores de cohesión* es donde más problemas han surgido, puesto que se tuvieron

que desarrollar las métricas de similitud semántica, y se encontraron problemas de rendimiento con *Google Sentence Encoder*, lo cual obligó a realizar cambios en la aplicación para que dichas métricas pudieran ser calculadas, tal y como se ha explicado anteriormente.

Además, también se tuvieron que realizar más pruebas de las que inicialmente se habían planeado, con el fin de pulir y depurar lo máximo posible la aplicación, por lo que la tarea de *Realización de pruebas* se vio afectada.

Por otro lado, para algunas de las tareas se estimó más duración de la que realmente era necesaria (por ejemplo, *Definición del Modelo de Dominio*), por lo que esta variación también se ha visto reflejada en la duración final del proyecto.

Por último, se considera oportuno calcular el coste final del proyecto puesto que, teniendo en cuenta la variación temporal que ha sufrido el proyecto, y el riesgo que se ha comentado en el apartado anterior, la evaluación económica realizada al comienzo del proyecto habrá sufrido algunos cambios.

Estos cambios se verían reflejados en el *Coste de la mano de obra* (que corresponde a 10,88 $^{€}/_{hora}$), que ascendería a 5614,08 €, y en el *Coste del servidor* (correspondiente a 35,748 $^{€}/_{mes}$), lo cual ascendería a 169,79 €. En consecuencia, los *Costes indirectos* también se verían afectados, los cuales ascenderían a 296,14 €. En la Tabla 63 se pueden observar los costes finales del proyecto.

Descripción	Coste
Coste de la mano de obra	5614,08 €
Coste del equipo informático	104,34 €
Coste del servidor	169,79 €
Coste del software	34,50 €
Costes indirectos	296,14 €
Total	6218,85 €

Tabla 63. Coste final del proyecto.

Como se puede observar, esto supondría un coste de 619,37 € por encima de lo inicialmente estimado.

8.4 LÍNEAS FUTURAS

Pese a que se ha cumplido con los objetivos y se considera que el resultado final de la aplicación es satisfactorio, existen ciertas funcionalidades y aspectos que pueden ser mejorados o añadidos. A continuación, se proponen dichas funcionalidades y aspectos a mejorar:

- **Escoger qué métricas o indicadores se desean calcular.** A la hora de analizar un fichero, en lugar de calcular todos los indicadores, el sistema podría darle al usuario la opción de elegir cuáles de esos indicadores desea que la aplicación calcule. Por ejemplo, si el usuario quisiera calcular únicamente el número de palabras en el texto, simplemente tendría que seleccionar dicho indicador en un listado antes de pulsar el botón *Analizar*.

- **Sistema de usuarios para guardar los resultados en la cuenta del usuario.** La aplicación podría tener un sistema de registro e inicio de sesión, de manera que, si el usuario inicia sesión, éste podría guardar los resultados de todos los análisis que haya realizado, sin que sea necesario descargarlos en formato CSV.

- **Analizar ficheros de texto en otros idiomas.** La aplicación podría dar la opción de analizar texto en cualquier otro idioma como, por ejemplo, en euskera o en castellano.

- **Permitir la utilización de rúbricas del profesorado para facilitar la corrección de los textos de sus alumnos.** Los profesores y profesoras de los centros educativos, generalmente, utilizan rúbricas que utilizan para corregir exámenes y/o textos escritos por sus alumnos y alumnas. La funcionalidad que se propone es que el profesorado pueda introducir en la aplicación una rúbrica y que *AzterTest* sea capaz de determinar qué puntos o aspectos de dicha rúbrica se cumplen en el texto.

- **Realizar mejoras en la eficiencia de la aplicación.** Puesto que existen algunas métricas que necesitan más tiempo para ser calculadas, la aplicación puede tardar más tiempo del deseado en analizar múltiples ficheros. Por ello, sería conveniente realizar mejorar de rendimiento, optimizando al máximo la computación de los indicadores, de manera que estos puedan ser calculados en el menor tiempo posible.

8.5 VALORACIÓN PERSONAL

A nivel personal, quiero resaltar que me ha resultado muy interesante realizar un proyecto de este tipo, además de que me ha servido para adquirir conocimientos acerca del procesamiento de lenguaje natural y sobre las distintas métricas que sirven para determinar la complejidad de un texto. Además, también me ha servido para adquirir experiencia con el lenguaje de programación *Python*, puesto que a lo largo del grado se ha trabajado principalmente con *Java*, y con *Python* se había trabajado muy poco. Por ello, tenía gran interés en aprender más sobre este lenguaje y realizar un trabajo más extenso con él, puesto que considero que es un lenguaje que tiene mucho futuro

Por otro lado, especialmente al comienzo del proyecto, fue complicado compaginar el desarrollo del mismo, con la asistencia a clase y la realización de las prácticas en empresa, pero no tardé mucho en organizarme correctamente para que pudiera realizar las tres actividades de manera satisfactoria y evitar que se me acumulara el trabajo. Por lo tanto, considero que también he conseguido realizar un buen trabajo en cuanto a la organización de un proyecto se refiere, y he intentado ceñirme lo máximo posible a la planificación temporal inicial, aunque algunas de las estimaciones que hice fueron algo optimistas, la mayoría de estimaciones fueron bastante acertadas.

En este proyecto, además de desarrollar la aplicación en *Python*, se ha desarrollado una aplicación web mediante la cual los usuarios podrán, de manera sencilla, analizar todos aquellos documentos de texto en inglés que deseen. El código de esta aplicación, pese a que tiene algún método implementado en *JavaScript* (para la parte cliente), es mayormente *PHP*. Elegí desarrollar la parte servidor en este lenguaje porque disponía experiencia previa con él, y consideraba que sería una buena opción. En cambio, mientras estaba desarrollándolo, comencé a cursar la asignatura *Desarrollo de Aplicaciones Web Enriquecidas* (correspondiente al segundo cuatrimestre del cuarto curso), donde se da temario en el que se trabaja con *NodeJS* y me di cuenta de que desarrollar la parte servidor en *JavaScript* hubiese sido una mejor opción. Sin embargo, puesto que el desarrollo de la aplicación web ya estaba muy avanzado, descarté la opción de desarrollar la parte servidor en *JavaScript* aunque, en caso de que volviera a hacer un proyecto similar a este, me decantaría por *NodeJS* desde el principio.

En definitiva, me encuentro muy satisfecho con el trabajo que he realizado en este proyecto, y considero que he aprendido mucho y he mejorado mis habilidades con el lenguaje de programación *Python* (y, además, he aprendido a utilizar la herramienta de *machine learning* scikit-learn), lo cual puede ser muy beneficioso para mi futuro, puesto que muchas empresas del sector lo consideran un lenguaje muy importante.

Bibliografía

Agrawal, S. (2019). *Hyperparameters in Deep Learning*. Obtenido de TowardsDataScience: https://towardsdatascience.com/hyperparameters-in-deep-learning-927f7b2084dd

Asaithambi, S. (2018). *Why, How and When to apply Feature Selection*. Obtenido de TowardsDataScience: https://towardsdatascience.com/why-how-and-when-to-apply-feature-selection-e9c69adfabf2

Boros, T., Dumitrescu, D. S., & Burtica, R. (1 de 11 de 2018). *NLP-Cube: End-to-end raw text processing with neural networks*. Bruselas, Bélgica.

Capsada Blanch, R., & Torruella Casañas, J. (2017). *Métodos para medir la riqueza léxica de los textos.*

Carnegie Mellon University. (s.f.). *The CMU Pronouncing Dictionary*. Obtenido de http://www.speech.cs.cmu.edu/cgi-bin/cmudict

Carrillo, H., Castellanos, J., & Brodersen, K. (s.f.). Probabilistic performance evaluation for multiclass classification using the posterior balanced accuracy.

Cer, D., Yang, Y., & Kong, S.-y. (2018). *Universal Sentence Encoder*.

Crossley, S. A., & Allen, D. B. (Abril de 2011). *Text readability and intuitive simplification: A comparison of readability formulas*.

Freyhoff, G., Hess, G., Kerr, L., Menzel, E., Tronbacke, B., & Van Der Veken, K. (1998). *European Guidelines for the Production of Easy-to-Read Information*.

Graesser, McNamara, Louwerse, & Cai. (2004). *Coh-Metrix: Analysis of text on cohesion and language*.

Johansson, V. (2008). *Lexical diversity and lexical density in speech and writing: a developmental perspective.* Lund University.

Le, Q., & Mikolov, T. (s.f.). *Distributed Representations of Sentences and Documents*.

Lu, X. (2017). *Haiyang Ai*. Obtenido de https://aihaiyang.com/software/l2sca/

Madrazo, I. (2016). *Towards Multipurpose Readability Assessment*.

Mandera, P. (2016). *Psycholinguistics on a Large Scale: Combining Text Corpora, Megastudies, and Distributional Semantics to Investigate Human Language Processing*.

McNamara, Graesser, McCarthy, & Cai. (2014). *Automated Evaluation of Text and Discourse with Coh-Metrix.* Cambridge: Cambridge University Press.

Merril, W., & Baum, E. (s.f.). *Voynich2Vec: Using FastText Word Embeddings for Voynich Decipherment*.

Ministerio de Empleo y Seguridad Social. (2018). *XVII Convenio colectivo estatal de empresas de consultoría, y estudios de mercados y de la opinión pública.* Obtenido de Boletín Oficial del Estado: https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf

NLTK. (s.f.). *nltk.tokenize.punkt module*. Obtenido de https://www.nltk.org/api/nltk.tokenize.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 2825--2830.

Rodríguez, T. (21 de Abril de 2019). *Genbeta*. Obtenido de https://www.genbeta.com/desarrollo/estas-razones-que-programadores-estan-empezando-a-aprender-python

scikit-learn. (s.f.). *sklearn.ensemble.AdaBoostClassifier*. Obtenido de scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

scikit-learn. (s.f.). *sklearn.ensemble.GradientBoostingClassifier*. Obtenido de scikit-learn.com: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

scikit-learn. (s.f.). *sklearn.ensemble.RandomForestClassifier*. Obtenido de scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Siddharthan, A. (2002). *An Architecture for a Text Simplication System*.

Sieg, A. (2018). *Text Similarities : Estimate the degree of similarity between two texts*.

Spache, G. (1953). *A new readability formula for primary-grade reading materials.*

Speer, R. (s.f.). Obtenido de https://github.com/LuminosoInsight/wordfreq

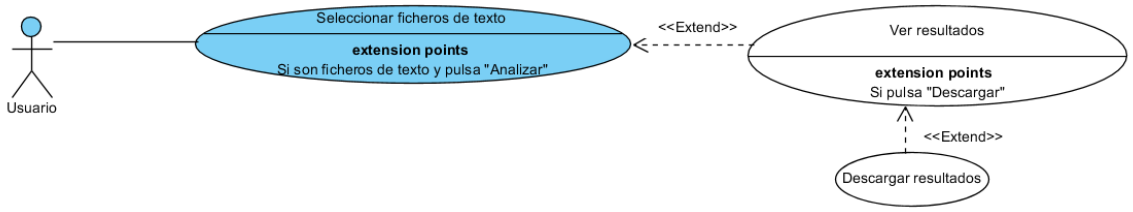
Stahl, S. A. (2003). *Vocabulary and Readability: How Knowing Word Meanings Affects Comprehension*. Lippincott Williams & Wilkins.

Tavernier, J., & Bellot, P. (2012). *Flesch and Dale-Chall Readability Measures for INEX 2011 Question-Answering Track*.

Vajjala, S., & Lucic, I. (2018). *OneStopEnglish corpus: A new corpus for automatic readability assessment and text simplification*.

ANEXO I: Casos de uso extendidos

En este anexo, se presentan los casos de uso extendidos mostrados en la sección 4.2 del capítulo 4. Por lo tanto, se muestra toda la información relativa a cada caso de uso, incluyendo qué tipo de decisiones se pueden tomar y como se gestionan. Para facilitar la visualización del caso de uso que se esté extendiendo, se éste será representado mediante el color azul.

<table>
<tr><th colspan="2">Seleccionar ficheros de texto</th></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td>Descripción</td><td>El usuario puede seleccionar, de su equipo, los ficheros que desee</td></tr>
<tr><td>Actores</td><td>Usuario.</td></tr>
<tr><td>Precondiciones</td><td>Situarse en la página principal del sistema web.</td></tr>
<tr><td>Requisitos no funcionales</td><td>Ninguno.</td></tr>
<tr><td colspan="2">Flujo de eventos</td></tr>
<tr><td colspan="2">1) El usuario pulsa sobre el botón “Elegir archivos” y selecciona los ficheros que desee analizar (Ilustración 105).
2) El usuario pulsa el botón “Analizar” (Ilustración 106).
[Si el usuario no selecciona ningún fichero]
2A) Se muestra un mensaje de error informando de que el usuario no ha introducido ningún fichero (Ilustración 107).
[Si el usuario introduce uno o más ficheros pero uno más no son de texto (DOC, DOCX, ODT o TXT)]
2B) Se muestra un mensaje de error informando de que el usuario no ha introducido ningún fichero (Ilustración 108).
[Si el usuario introduce uno o más ficheros de texto (DOC, DOCX, ODT o TXT)]
2C) Comienza el proceso de análisis de los ficheros introducidos (Ilustración 109).</td></tr>
<tr><td>Postcondiciones</td><td>El usuario habrá seleccionado ficheros.</td></tr>
<tr><td colspan="2">Interfaz de Usuario</td></tr>
</table>

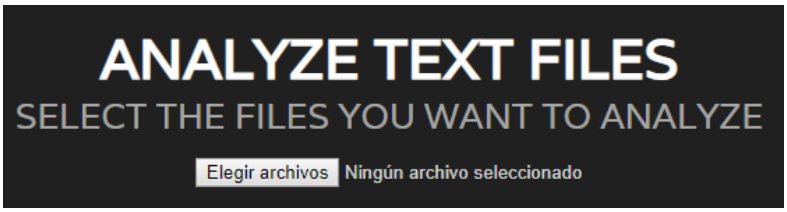


Ilustración 105. Elegir archivos.

Ilustración 106. Botón Analizar.

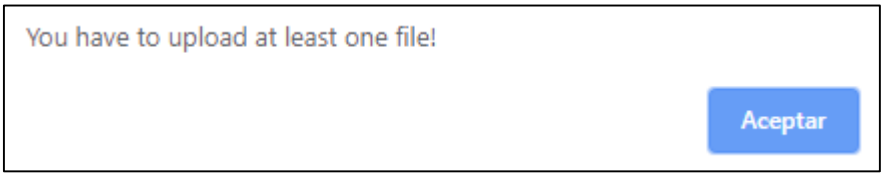


Ilustración 107. Alerta de cantidad de ficheros.

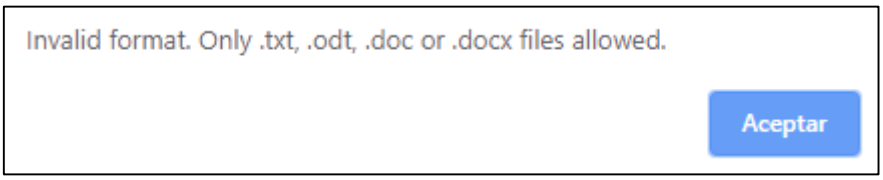


Ilustración 108. Alerta de formato inválido.

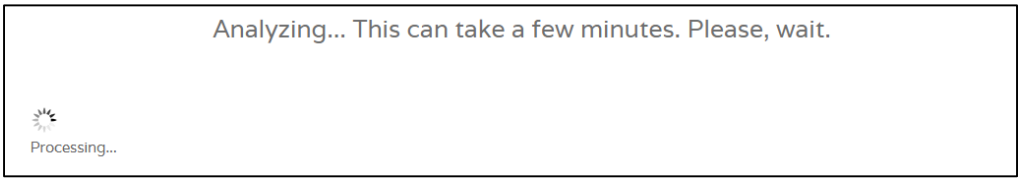
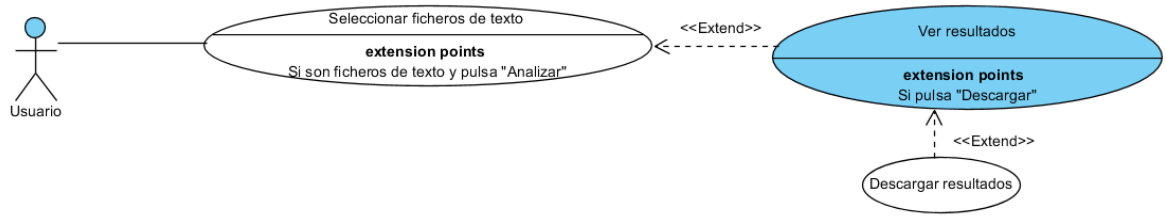


Ilustración 109. Analizando ficheros.

Ver resultados



Descripción	El usuario puede visualizar los resultados del análisis realizado por la aplicación.
Actores	Usuario.
Precondiciones	Haber analizado al menos un fichero.
Requisitos no funcionales	Ninguno.

Flujo de eventos

1) El usuario visualiza los resultados del análisis (**Ilustración 110**).
   [Si el usuario pulsa el botón "Descargar" (**Ilustración 111**)*]*
   1A) Se descargan, en formato CSV, los resultados de los ficheros analizados.

Postcondiciones	Se ha completado el análisis de los ficheros.

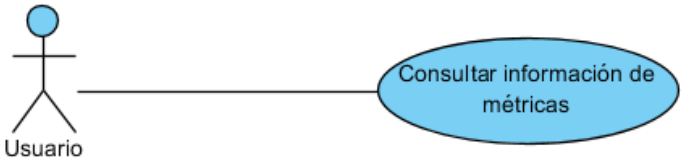
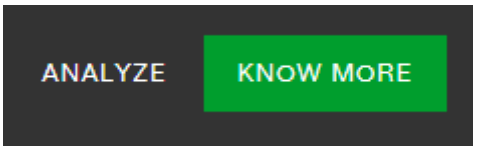
Interfaz de Usuario

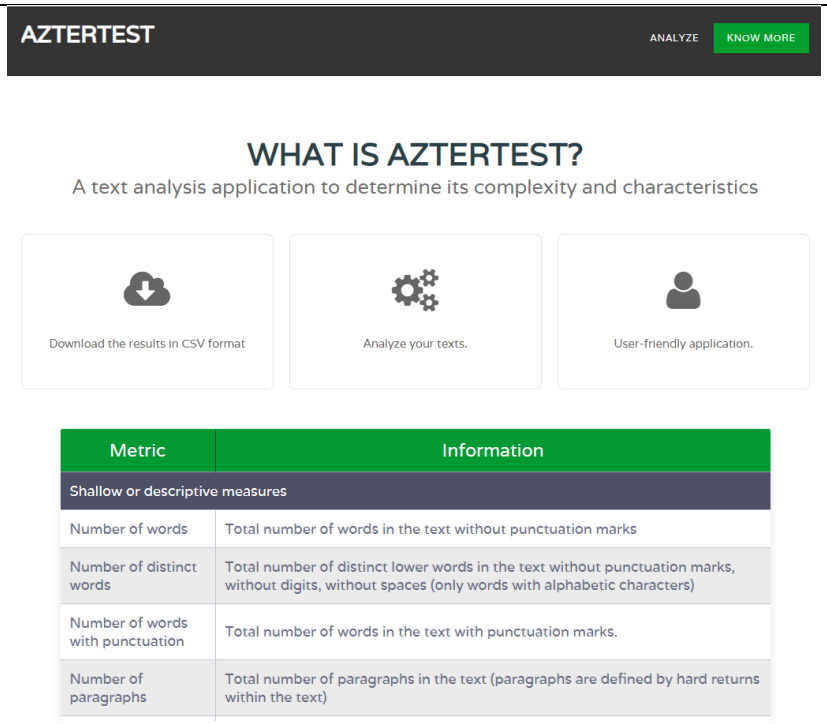
Results of E_DBH1_L1.doc.txt

File: E_DBH1_L1.doc.txt	
Shallow or descriptive measures	
Number of words (total)	269
Number of distinct words (total)	115
Number of words with punctuation (total)	316
Number of paragraphs (total)	5
Number of paragraphs (incidence per 1000 words)	18.587360594795538
Number of sentences (total)	24
Number of sentences (incidence per 1000 words)	89.21933085501858
Length of paragraphs (mean)	4.4
Standard deviation of length of paragraphs	3.32264954516723
Number of words (length) in sentences (mean)	10.916666666666666
Number of words (length) in sentences (standard deviation)	6.369959881262117
Number of words (length) of sentences without stopwords (mean)	7.708333333333333
Number of words (length) of sentences without stopwords (standard deviation)	4.237522533535629

Ilustración 110. Resultados de la aplicación.

Ilustración 111. Descargar resultados.

Consultar información de métricas	
	
Descripción	El usuario puede consultar información acerca de las métricas que la aplicación calcula.
Actores	Usuario.
Precondiciones	Ninguna.
Requisitos no funcionales	Ninguno.
Flujo de eventos	
1) El usuario pulsa sobre el botón "Saber más" situado en la parte superior de la aplicación (**Ilustración 112**). 2) Se visualizan información acerca de las métricas en una tabla, además de algo de información sobre la aplicación (**Ilustración 113**)**.**	
Postcondiciones	El usuario habrá seleccionado ficheros.
Interfaz de Usuario	
 *Ilustración 112. Botón "Saber más".*	



Metric	Information
Shallow or descriptive measures	
Number of words	Total number of words in the text without punctuation marks
Number of distinct words	Total number of distinct lower words in the text without punctuation marks, without digits, without spaces (only words with alphabetic characters)
Number of words with punctuation	Total number of words in the text with punctuation marks.
Number of paragraphs	Total number of paragraphs in the text (paragraphs are defined by hard returns within the text)

Ilustración 113. Información sobre los indicadores.

ANEXO II: Diagramas de secuencia

En este anexo, se presenta el diagrama de secuencia de la principal funcionalidad de la aplicación. Por lo tanto, se muestran las llamadas que se realizan entre los distintos métodos de las clases y se detallará el objeto que se devuelve cuando la secuencia finaliza.

La principal funcionalidad es la de analizar ficheros de texto. La secuencia, tal y como se puede observar en la Ilustración 114, comienza cuando el usuario selecciona uno o más ficheros de texto y pulsa el botón "Analizar". Si los ficheros seleccionados son correctos (es decir, están en formato ODT, DOCX, DOC o TXT), dará comienzo el proceso de análisis de los ficheros. Para ello, se realiza una llamada al método *start()* de la aplicación *Python*, que se encarga de cargar los documentos de texto (guardando en una lista los directorios completos en los que se encuentran dichos ficheros) y los ficheros necesarios para el análisis (el listado de Dale-Chall, etc.). Además, en este mismo método, se crea un directorio en el que se guardarán los resultados obtenidos por la herramienta y, después, por cada uno de los ficheros, se procesa el texto (eliminando saltos de línea y caracteres no deseados), se crea una instancia del analizador que analizará dichos ficheros y, con los resultados (*indicadores* será un diccionario de *Python* con todos los indicadores y sus resultados), creará un *DataFrame* que servirá para predecir la complejidad/dificultad del texto. Tras haber realizado todo este proceso, se imprimen los resultados y se crea el fichero CSV correspondiente al documento de texto analizado.

Por último, la aplicación web se encargará de interpretar los ficheros CSV devuelto por la aplicación *Python* y de representar su contenido en una tabla, de manera que sea cómodo para el usuario visualizar los resultados.

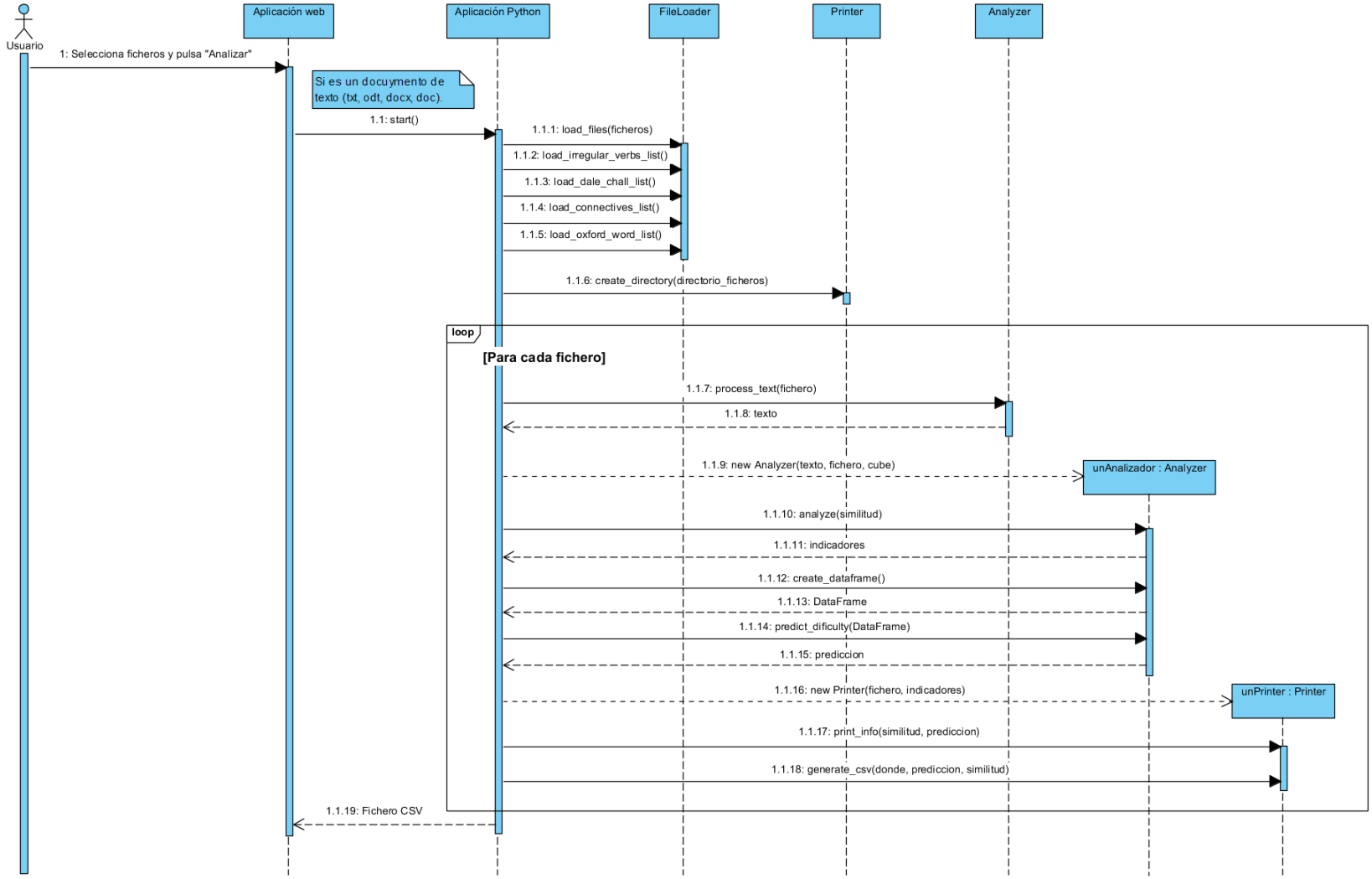


Ilustración 114. Diagrama de secuencia.