



UNIVERSIDADE DA CORUÑA

FACULTAD DE INFORMÁTICA

TRABAJO FIN DE MÁSTER

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA

Profiling automático de usuarios para la detección temprana de trastornos mentales

Autora: Piot Pérez-Abadín, Paloma
Directores: Martín Rodilla, Patricia
Parapar López, Javier

A Coruña, junio de 2021

En este Trabajo de Fin de Máster se ha desarrollado un sistema que realiza un perfilado automático de usuarios con una aplicabilidad en la detección temprana de trastornos mentales, apoyado en una aplicación web donde se permite validar y corregir los distintos datos extraídos por el sistema (tales como el perfil demográfico o como indicativos de depresión).

La motivación principal de este trabajo es realizar un perfilado automático de usuarios basándonos en información no estructurada. Gracias a este, se podrá estudiar el caso del diagnóstico de trastornos psicológicos, para, así, ayudar a detectar enfermedades difíciles de predecir, o que suelen pasar desapercibidas. Esto será posible a partir del análisis de comentarios y expresiones en redes sociales. Para ayudar con esta problemática, se realizará una aplicación para automatizar la tarea de recogida de datos y creación de colecciones, así como la validación y corrección de datos que el sistema de perfilado automático realice.

Existen dos tipos de usuarios principales: el usuario administrador, que puede realizar las operaciones de carga y procesado de datos, creación de corpus y exportación de colecciones de datos, tanto conjuntos de datos etiquetados, como datos demográficos. Y el usuario experto, aquel que se encarga de las tareas de corrección y validación de los datos extraídos por el sistema. Estas funcionalidades se ven complementadas con filtros para indicar de donde obtener los datos y la cantidad de estos, exportación de datos basándose en los distintos corpus definidos, procesado de usuarios de Reddit indicando el nombre de usuario solamente o búsquedas más acertadas gracias al filtrado.

Esta división entre los dos perfiles permite a los distintos usuarios ejercer sus tareas pertinentes en la aplicación web y colaborar así para poder obtener un perfil automático basándose en comentarios de usuarios en redes sociales, afinado gracias a la corrección del experto y ofreciendo la posibilidad de crear corpus de datos para futuros experimentos.

Para la ejecución del proyecto se ha seguido una adaptación de la metodología de desarrollo ágil Scrum. Se ha desarrollado en paralelo la aplicación web y los distintos modelos de perfilado, obteniendo así un incremento observable y valor añadido para el usuario tras cada iteración. La aplicación web se ha implementado utilizando tecnologías de código abierto como Django, Flask, PostgreSQL, Docker, PRAW, Celery, Redis, Selenium u OpenAPI. Para la parte de los modelos del perfilado, se ha optado por tecnologías como spaCy, scikit-learn, NLTK y lightgbm, entre otras. También, se han empleado herramientas como Jupyter Notebook, Visual Studio Code, AWS, Heroku, Docker Hub, SonarCloud, GitLab o Taiga, para facilitar el desarrollo.

Palabras clave:

- * *Author Profiling*
- * Redes sociales
- * Datos demográficos
- * Depresión
- * Trastornos psicológicos
- * Procesado de lenguaje natural
- * Aprendizaje automático
- * Aplicación web
- * Integración y entrega continua
- * Python
- * Celery
- * Redis
- * PostgreSQL
- * spaCy
- * LightGBM
- * BERT
- * *Question Answering*

Índice general

	Página
1. INTRODUCCIÓN	3
1.1. Motivación	3
1.1.1. Estado del arte	4
1.2. Objetivos	5
1.3. Estructura de la memoria	7
2. FUNDAMENTOS	9
2.1. Tarea de clasificación de género	9
2.1.1. Proceso de trabajo	9
2.1.2. Objetivo de la tarea	10
2.1.3. Base o referencia	12
2.1.4. Conjuntos de datos	12
2.1.4.1. “PAN Author Profiling 2019”	13
2.1.4.2. “PAN Celebrity Profiling 2019”	14
2.1.5. Preprocesado	15
2.1.6. <i>Feature engineering</i> o preparación de los datos	16
2.1.6.1. <i>Features</i> sociolingüísticas	16
2.1.6.2. Características de análisis de sentimiento	17
2.1.6.3. <i>Topic modelling</i>	18
2.2. Tarea de medición de la gravedad de los signos de depresión	18
2.2.1. Proceso de trabajo	18
2.2.2. Objetivo de la tarea	19
2.2.3. Base o referencia	20
2.2.4. Conjuntos de datos	21
2.2.5. Preprocesado	22
3. FUNDAMENTOS TECNOLÓGICOS Y HERRAMIENTAS UTILIZADAS	23
3.1. Tecnologías	23
3.1.1. Aplicación web y servicios	23
3.1.1.1. Python	24

3.1.1.2. Django	25
3.1.1.3. Django REST Framework	26
3.1.1.4. OpenAPI	27
3.1.1.5. PRAW	28
3.1.1.6. Celery	29
3.1.1.7. Redis	30
3.1.1.8. PostgreSQL	31
3.1.1.9. Flask	31
3.1.1.10. HTML	32
3.1.1.11. CSS	33
3.1.1.12. Bootstrap	33
3.1.1.13. Locust	34
3.1.1.14. Behave	35
3.1.1.15. Selenium WebDriver	35
3.1.1.16. Docker	36
3.1.2. Perfilador	38
3.1.2.1. spaCy	38
3.1.2.2. NLTK	39
3.1.2.3. scikit-learn	40
3.1.2.4. LightGBM	41
3.1.2.5. Pyphen	42
3.1.2.6. Gensim	42
3.1.2.7. TensorFlow	43
3.1.2.8. Keras	43
3.2. Herramientas de desarrollo	44
3.2.1. Visual Studio Code	44
3.2.2. Jupyter	45
3.2.3. Google Colaboraty Pro	46
3.2.4. PlantUML	46
3.2.5. Pencil	47
3.2.6. GitLab	47
3.2.7. Docker Hub	48
3.2.8. AWS	48
3.2.9. Heroku	50
3.2.10. Taiga	51
3.2.11. SonarCloud	51
3.2.12. LaTeX y Overleaf	52
3.3. Infraestructura	53
4. METODOLOGÍA	55
4.1. Scrum	55
4.1.1. Roles	57

4.1.2.	Eventos de Scrum	58
4.1.2.1.	Sprint	58
4.1.2.2.	Artefactos	59
4.1.2.3.	Entidades	60
4.2.	Adaptación	61
4.2.1.	Personas	61
4.2.2.	Reuniones	61
4.3.	Control de versiones	62
4.3.1.	CI/CD	62
4.4.	<i>Behaviour-driven development (BDD)</i>	63
4.5.	<i>API first development</i>	64
5. ANÁLISIS		67
5.1.	Descripción general	67
5.2.	Requisitos	67
5.2.1.	Actores	68
5.2.2.	Historias de usuario	69
5.2.2.1.	Historia de usuario 1: Gestión de usuarios	70
5.2.2.2.	Historia de usuario 2: Gestión de perfiles	71
5.2.2.3.	Historia de usuario 3: Gestión de datos	72
5.2.2.4.	Historia de usuario 4: Perfilado automático	73
5.3.	Diagrama entidad relación	74
5.4.	Modelo de datos	76
6. DISEÑO		81
6.1.	Maquetas	81
6.2.	Diagramas de secuencia	92
6.2.1.	Administrador: cargar datos	93
6.2.2.	Experto: flujo completo	94
6.3.	Arquitectura global	95
6.3.1.	Aplicación web	96
6.3.1.1.	Cliente	96
6.3.1.2.	Servidor	96
6.3.1.3.	Diagrama de clases	96
6.3.1.4.	Distribución en paquetes	97
6.3.1.5.	Internacionalización	98
6.3.2.	API interno	99
6.3.2.1.	Capa web (<i>viewset</i>)	99
6.3.2.2.	Serializadores	99
6.3.2.3.	Capa modelo	100
6.3.2.4.	Diagrama de clases	100
6.3.2.5.	Distribución en paquetes	101

6.3.3. Servicio perfilador	102
6.3.3.1. Capa web	102
6.3.3.2. Lógica de negocio	102
6.3.3.3. Diagrama de clases	102
6.3.3.4. Distribución en paquetes	103
7. PLANIFICACIÓN Y EVALUACIÓN DE COSTES	105
7.1. Planificación inicial	105
7.2. Coste	106
7.3. Seguimiento	107
7.3.1. Sprint 0 (20.01.2020 - 10.02.2020)	108
7.3.2. Sprint 1 (10.02.2020 - 02.03.2020)	108
7.3.3. Sprint 2 (02.03.2020 - 23.03.2020)	109
7.3.4. Sprint 3 (23.03.2020 - 13. 04.2020)	109
7.3.5. Sprint 4 (13.04.2020 - 04.05.2020)	109
7.3.6. Sprint 5 (04.05.2020 - 25.05.2020)	110
7.3.7. Sprint 6 (25.05.2020 - 15.06.2020)	110
7.3.8. Sprint 7 (15.06.2020 - 06.07.2020)	110
7.3.9. Sprint 8 (06.07.2020 - 27.07.2020)	110
7.3.10. Sprint 9 (27.07.2020 - 17.08.2020)	110
7.3.11. Sprint 10 (17.08.2020 - 07.09.2020)	110
7.3.12. Sprint 11 (07.09.2020 - 28.09.2020)	111
7.3.13. Sprint 12 (28.09.2020 - 19.10.2020)	111
7.3.14. Sprint 13 (19.10.2020 - 09.11.2020)	111
7.3.15. Sprint 14 (09.11.2020 - 30.11.2020)	111
7.3.16. Sprint 15 (30.11.2020 - 21.12.2020)	111
7.3.17. Sprint 16 (21.12.2020 - 11.01.2021)	112
7.3.18. Sprint 17 (11.01.2021 - 01.02.2021)	112
8. PRUEBAS Y RESULTADOS	113
8.1. Pruebas	113
8.1.1. Pruebas de unidad	113
8.1.2. Pruebas de integración	114
8.1.3. Pruebas de aceptación automatizadas	115
8.1.4. Pruebas de rendimiento	119
9. SOLUCIÓN IMPLEMENTADA	123
9.1. Página de inicio	123
9.2. Funcionalidades de gestión de usuarios	125
9.2.1. Iniciar sesión	125
9.2.2. Recordar contraseña	125
9.2.3. Cerrar sesión	126

9.2.4. Cuenta	126
9.2.4.1. Editar detalles de la cuenta	127
9.2.4.2. Cambiar contraseña	128
9.3. Funcionalidades de gestión de perfiles	129
9.3.1. Ver perfiles	129
9.3.2. Buscar perfiles	130
9.3.3. Ver detalle perfil	131
9.3.4. Validar perfil	132
9.3.5. Ver y corregir formulario de depresión	133
9.4. Funcionalidades de gestión de datos	134
9.4.1. Panel de administración	134
9.4.2. Obtener datos de Reddit	135
9.4.3. Exportar datos	136
9.4.4. Crear corpus	137
9.4.5. Procesar usuario de Reddit	138
9.5. Interfaces en dispositivo móvil	139
10. EVALUACIÓN	145
10.1. Tarea de clasificación de género	145
10.1.1. Funcionalidad	145
10.1.2. Experimentos	146
10.1.2.1. k-fold cross-validation	146
10.1.2.3. Resultado	148
10.2. Tarea de medición de la gravedad de los signos de depresión	152
10.2.1. Funcionalidad	152
10.2.2. Experimentos	153
10.2.2.1. QA	153
10.2.2.2. BERT	154
10.2.2.3. Modelo implementado	156
10.2.3. Resultados	158
11. CONCLUSIONES	161
11.1. Trabajo realizado	161
11.2. Lecciones aprendidas	162
11.3. Líneas futuras	163
A. Glosario de términos	167
Bibliografía	171

Índice de figuras

Figura	Página
2.1. Diagrama del proceso de trabajo de la tarea de clasificación de género.	10
2.2. Diagrama del proceso de trabajo de la tarea de medición de la gravedad de los signos de depresión.	19
3.1. Word frente a LaTeX. Complejidad del documento ante el esfuerzo que conlleva. <i>Fuente: stackexchange.com</i>	52
4.1. Metodología Scrum. <i>Fuente: scrum.org</i>	57
4.2. Ejemplo de la documentación del API del sistema	65
5.1. Diagrama de actores	69
5.2. Diagrama de las funcionalidades de la <i>Gestión de usuarios</i>	71
5.3. Diagrama de las funcionalidades de la <i>Gestión de perfiles</i>	72
5.4. Diagrama de las funcionalidades de la <i>Gestión de datos</i>	73
5.5. Diagrama de las funcionalidades del <i>Perfilado automático</i>	73
5.6. Diagrama entidad-relación completo	75
5.7. Núcleo del diagrama entidad-relación	76
6.1. Maqueta de la interfaz de login	82
6.2. Maqueta de la interfaz de cambio de contraseña	83
6.3. Maqueta de la interfaz de recuperación de la contraseña	83
6.4. Maqueta de la interfaz de actualización de detalles de la cuenta	84
6.5. Maqueta de la interfaz de los perfiles	85
6.6. Maqueta de la interfaz de detalle del perfil	86
6.7. Maqueta de la interfaz del panel de administración	87
6.8. Maqueta de la interfaz del panel de detalles de la cuenta	88
6.9. Maqueta de la interfaz de validación y edición de un perfil	89
6.10. Maqueta de la interfaz de carga de datos de Reddit	89
6.11. Maqueta de la interfaz del questionario de Beck	90
6.12. Maqueta de la interfaz de exportación de datos	90
6.13. Maqueta de la interfaz de creación de corpus	91

6.14. Maqueta de la interfaz de procesamiento de un usuario	91
6.15. Diagrama de secuencia de la carga de datos de Reddit	93
6.16. Diagrama de secuencia de un flujo normal del usuario experto	94
6.17. Arquitectura global del proyecto	95
6.18. Diagrama de clases de la aplicación web	97
6.19. Diagrama de clases del API interno	101
6.20. Diagrama de clases del microservicio de perfilado	103
 7.1. Evolución del desarrollo del proyecto	108
8.1. Estadísticas de las peticiones de las pruebas de carga	120
8.2. Gráfica de las peticiones por segundo realizadas	120
8.3. Gráficas del tiempo de respuesta y el número de usuarios en cada instante de tiempo	121
 9.1. Interfaz de inicio (primera parte)	124
9.2. Interfaz de inicio (segunda parte)	124
9.3. Interfaz de login	125
9.4. Interfaz de recordar contraseña	126
9.5. Panel de la cuenta de usuario	127
9.6. Interfaz de actualización de los detalles de la cuenta	128
9.7. Interfaz de cambio de la contraseña	129
9.8. Interfaz de perfiles	130
9.9. Interfaz de perfiles con filtrado	131
9.10. Interfaz de detalle de un perfil	132
9.11. Modal para validar y editar un perfil	133
9.12. Modal para ver y editar el cuestionario de depresión	134
9.13. Interfaz del panel de administración	135
9.14. Modal que permite configurar experimentos de carga de datos	136
9.15. Modal que permite personalizar la exportación de datos	137
9.16. Modal que permite crear un corpus	138
9.17. Pantalla que permite procesar un usuario de Reddit	139
9.18. Pantalla que permite cargar datos de Reddit	140
9.19. Pantalla que permite exportar datos	140
9.20. Pantalla de detalle de un perfil	141
9.21. Pantalla de detalle de un perfil	141
9.22. Pantalla de detalle del cuestionario de depresión	142
9.23. Pantalla de edición de los datos de un perfil	142
9.24. Pantalla del menú de la cuenta del usuario	143
 10.1. Diagrama de la tarea de clasificación de género. <i>Publicado en Piot-Perez-Abadin et al.</i>	
[1]	146
10.2. Relevancia de las <i>features</i> en nuestro modelo. <i>Publicado en Piot-Perez-Abadin et al. [1]</i>	150
10.3. Impacto de las <i>features</i> en la categoría de clasificación “male”.	151

10.4. Impacto de las <i>features</i> en la categoría de clasificación “female”.	152
10.5. <i>Segment embeddings</i> para la tarea de <i>Question Answering</i>	155
10.6. Proceso de <i>fine-tuning</i> para BERT	156
10.7. Ejemplo de pregunta, fragmento de texto y respuesta extraída	157
10.8. Ejemplo de <i>Word Mover’s Distance</i> entre dos documentos	158

Listings

3.1. Función que realiza la validación de un perfil en el navegador	35
8.1. Test de unidad	114
8.2. Test de integración	115
8.3. Escenarios escritos en Gherkin	116
8.4. Funciones que simulan un usuario logeado y la creación de un perfil	117
8.5. Función que realiza la validación de un perfil en el navegador	118
8.6. Función que realiza la validación de un perfil en el navegador	118

Capítulo 1

INTRODUCCIÓN

En este capítulo se tratan los aspectos básicos para comprender el proyecto a desarrollar, sus líneas maestras, la motivación tras él y los objetivos que se persiguen.

1.1. Motivación

Los trastornos psicológicos son muy comunes en la actualidad. Solo en Europa, ochenta y tres millones de personas fueron afectadas con algún problema en el año 2013, según un estudio de la OMS (Organización Mundial de la Salud) de 2014. La depresión, por ejemplo, es un trastorno mental frecuente que afecta a más de trescientos millones de personas en el mundo [2].

En la era tecnológica en la que vivimos, las redes sociales se han convertido en una plataforma popular para millones de usuarios que comparten actividades y pensamientos. La actitud de una persona en las redes sociales suele dar pistas sobre su estado de salud mental. Gracias a la gran cantidad de datos generados, es de interés utilizarlos para detectar trastornos lo antes posible.

La motivación principal es lograr conseguir hacer un perfilado automático de usuarios, basándonos en información no estructurada, gracias al cual se podrá estudiar el caso del diagnóstico de trastornos psicológicos. Este será de ayuda para detectar enfermedades difíciles de predecir, o que suelen pasar desapercibidas, y siendo la fuente de este diagnóstico lo que escriben los distintos usuarios en redes sociales.

Será de gran utilidad el perfilado de los usuarios, ya que contar con información demográfica

y el perfil psicológico de los usuarios de redes sociales supondrá una gran ventaja para los especialistas. Sobre todo, disponer de información acerca del perfil demográfico y psicológico es de vital importancia en los sistemas actuales de diagnóstico médico temprano o de riesgo de enfermedad mental. Especialmente relevante resulta este perfilado si el objetivo es la detección temprana de enfermedades mentales, para así mejorar las expectativas de curación y ahorro de costes a los sistemas sociales.

Al mismo tiempo, uno de los principales problemas en los sistemas sanitarios es la carga excesiva de pacientes que sufren trastornos mentales y la falta de profesionales para evaluar y detectar correctamente este tipo de problemas. A esto se le suman las consecuencias de la pandemia mundial producida por la COVID-19 en temas de salud mental [3]. Por lo tanto, contar con una herramienta que evalúe y alerte de forma temprana sobre posibles riesgos en lo relacionado con la salud mental es de vital importancia.

1.1.1. Estado del arte

Actualmente, existen distintas iniciativas y proyectos relacionados con el presente trabajo.

Podemos decir que el proyecto está compuesto por:

- El análisis de textos en inglés para realizar un perfilado de usuarios.
- La estimación del nivel de depresión de un usuario.
- La validación y corrección de los datos extraídos en los puntos anteriores.

Algunos ejemplos reales que se han encontrado son los siguientes:

- Iniciativas como **PAN** [4] realizan competiciones para el perfilado del género y el análisis de la autoría de textos. A partir de un conjunto de datos, un *feed* de Twitter, se debe determinar si lo ha escrito un **bot** o un **humano**, y en el segundo caso, perfilar el **género** de este. En el caso del conjunto de datos en inglés, el mejor equipo obtuvo una precisión del 0.8356. También incluyen problemas como la detección de la personalidad del autor y su perfilado a partir de un conjunto de información no estructurada intentando determinar si dos textos han sido escritos por el mismo autor.
- Iniciativas como **eRisk** [5] organizan competiciones para detectar señales de autolesiones a partir de un conjunto de datos no estructurados. También miden la severidad de los signos de depresión, basado en la completitud automática de cuestionarios de ánimo, a

partir de un hilo de comentarios de usuarios. Se han conseguido acertar más de un 40 % de respuestas sobre el cuestionario mencionado.

- Existen distintos estudios y artículos académicos publicados donde se llevan a cabo las tareas del perfilado de la edad y el género del usuario que escribe textos [6, 7].
- Existen distintos estudios sociológicos donde se determinan las diferencias a la hora de escribir entre distintos géneros y basándonos en la edad de una persona [8, 9].

No se ha encontrado ninguna plataforma ni sistema que combine los tres puntos fundamentales sobre los que se sustenta el proyecto actual (perfilado de usuarios, estimación del nivel de depresión y sistema que permite esta gestión y automatización). Aun así, sí existen distintas iniciativas que se han centrado en algún punto concreto del mismo.

En la siguiente tabla 1.1 se pueden observar las distintas aproximaciones a nivel de modelo que ya han sido aplicadas según el aspecto a identificar.

Género	Edad	Localidad	Personalidad
SVM	SVM	NER Taggers	SVR
DCNN	BFTree	LNEEx	Ridge Regression
Logistic Regression	LIWC	LIWC	Word N-grams
Gaussian Naive Bayes	Gaussian Naive Bayes	Gazzetteer	
Ridge Regression	Linear SVC	Namelist	
Regular Patterns	Regular Patterns	Regular Patterns	
	Word N-grams		

Cuadro 1.1.: Estrategias aplicadas según el aspecto a identificar.

1.2. Objetivos

El objetivo de este proyecto, como se comentó en la sección anterior, es el *desarrollo de una plataforma para el diagnóstico psicológico a partir del análisis de textos de redes sociales*.

El perfilado de usuarios a partir de datos no estructurados tiene numerosas aplicaciones, entre las cuales destacamos el objeto de este TFM: la creación de un perfil demográfico en detección y tratamiento de enfermedades mentales. Particularmente, en detección y diagnóstico de depresión, la elaboración de un perfil demográfico del paciente ayuda al profesional médico en

su valoración y detección de signos tempranos. Los rasgos más estudiados generalmente son: la edad, el género, características de personalidad e idioma nativo.

Una de las principales ventajas es mantener la información de forma persistente, con una trazabilidad entre cada dato extraído del sistema y su origen, de una forma accesible a lo largo del tiempo. Apoyado en una aplicación que permite la corrección y validación de la tarea de automatización a desarrollar.

De esta forma, más detalladamente, en el sistema a desarrollar:

- Se pretende conectar la plataforma con Reddit¹, para recuperar textos de usuarios y que sea una fuente de entrada al sistema.
- A partir de las colecciones de datos, se implementará un sistema que analizará esos textos para realizar el perfilado de los usuarios, manteniendo la trazabilidad con el dato o texto que lanzó un indicio de la salida.
- Se realizarán una solución para clasificar los textos del usuario según distintos datos demográficos, centrándonos en el género.
- Se implementará una solución que permita detectar posibles trastornos a partir de los textos y sus datos demográficos extraídos.
- Se pondrá en marcha una solución que estimará el nivel de depresión de un usuario con la cumplimentación de cuestionarios de estado de ánimo [10].
- Se implementará un módulo para ofrecer una herramienta de ayuda para especialistas en la detección de enfermedades mentales, donde validarán, revisarán y corregirán las salidas del sistema.
- Se implementará un módulo para administradores, donde se permitirá ejecutar las consultas a la API de Reddit y exportar colecciones de datos.

Otro de los principales objetivos es contribuir a la detección de trastornos psicológicos, a la par que se permite a los especialistas realizar una supervisión sobre los datos que el sistema extrae. Es importante recalcar que, durante la ejecución del trabajo se construirán distintos modelos para predecir el género y otros aspectos demográficos del usuario, teniendo como una entrada únicamente los textos que escriben en redes sociales. Estos datos le servirán al experto para evaluar ciertos aspectos de la persona y analizar o predecir comportamientos y otros atributos.

¹<https://reddit.com/>

El mayor objetivo personal de este proyecto, además del propio desarrollo del sistema de procesado de textos, perfilado y la aplicación web, es la ampliación de conocimientos en lo concerniente al procesado de lenguaje natural y desarrollo web: patrones de diseño, buenas prácticas, *frameworks* de código abierto que puedan ser útiles, últimas tendencias de desarrollo software, etc.

1.3. Estructura de la memoria

La memoria del trabajo está estructurada en doce capítulos, siendo el primero de ellos esta introducción.

- Segundo capítulo: **Fundamentos**, se explicarán los objetivos y procedimientos para las distintas tareas de clasificación realizadas, así como los distintos conjuntos de datos utilizados.
- Tercer capítulo: **Fundamentos tecnológicos y herramientas utilizadas**, se dará un repaso a las diferentes tecnologías y herramientas empleadas en el desarrollo del sistema y a los motivos por los que fueron escogidas.
- Cuarto capítulo: **Metodología**, se hablará sobre la forma de trabajo seguida y su adaptación durante el desarrollo del proyecto.
- Quinto capítulo: **Análisis**, se describirán las funcionalidades de la aplicación. Entrando en detalle mediante una descripción de los requisitos, actores y casos de uso.
- Sexto capítulo: **Diseño**, se verá en detalle la arquitectura que sigue la aplicación web, tanto del lado del servidor como del lado del cliente.
- Séptimo capítulo: **Planificación y evaluación de costes**, se verá la estimación realizada en un primer momento y se comentará el posible precio real de la plataforma, teniendo en cuenta las horas empleadas y los diferentes perfiles adoptados.
- Octavo capítulo: **Pruebas y resultados**, se expondrán las diferentes pruebas realizadas y se comentarán los problemas conocidos.
- Noveno capítulo: **Solución desarrollada**, una vez expuesto el problema inicial que se ha desarrollado, se expondrán las funcionalidades destacables y las características principales del sistema.

- Décimo capítulo: ***Evaluación de modelos***, se realizará una evaluación del rendimiento, funcionalidad y resultado de los modelos propuestos, así como su marco dentro de la aplicación desarrollada.
- Undécimo capítulo: ***Conclusiones***, en este último capítulo se hablará tanto del resultado general obtenido mediante el desarrollo del proyecto como de las posibles líneas futuras de trabajo.

Capítulo 2

FUNDAMENTOS

En el presente capítulo se explicarán los conceptos científicos del trabajo, así como las tareas de investigación llevadas a cabo, la base o referencia para cada tarea, el procesado realizado, la preparación de los datos y los conjuntos de datos utilizados.

2.1. Tarea de clasificación de género

2.1.1. Proceso de trabajo

En la siguiente figura 2.1 se puede ver el proceso de trabajo seguido a la hora de realizar la tarea de clasificación de género. Este *workflow* de experimentación está compuesto por las fases de estudio y entendimiento del objetivo de la tarea (*Task Goal*), investigación sobre la base o referencia de la misma (*Task Reference*), selección de conjuntos de datos (*Datasets Selection*), preprocessado (*Preprocessing*) e ingeniería de *features* (*Feature Engineering*). Tras completar estos pasos, da comienzo la fase de experimentación (*Experiments*) y tras esta obtendremos el resultado de la tarea.

En este capítulo se explicarán en detalle los cinco primeros pasos, delegando los restantes al capítulo 10.

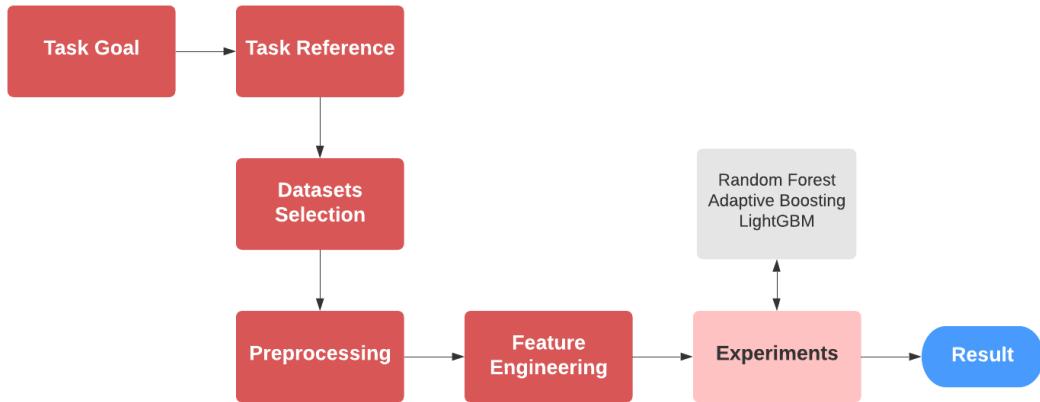


Figura 2.1.: Diagrama del proceso de trabajo de la tarea de clasificación de género.

2.1.2. Objetivo de la tarea

Profiling es el tratamiento automatizado de datos personales [11]. Consiste en utilizar datos personales para evaluar ciertos aspectos de la persona y analizar o predecir sus intereses, comportamientos y otros atributos. El perfilado de usuarios a partir de datos no estructurados tiene numerosas aplicaciones, entre las cuales destacamos el objeto de este TFM: la creación de un perfil demográfico en detección y tratamiento de enfermedades mentales. Particularmente, en detección y diagnóstico de depresión, la elaboración de un perfil demográfico del paciente ayuda al profesional médico en su valoración y detección de signos tempranos. Los rasgos más estudiados generalmente son: la edad, el género, características de personalidad e idioma nativo.

Como ya se ha comentado anteriormente, un dato de interés es el género de la persona. Estudios de la OMS [12] afirman que la depresión afecta en mayor medida a las mujeres que a los hombres, por lo que contar con este dato es de vital importancia. Los aspectos de género también influyen en los motivos disparadores de la enfermedad, así como en la evolución de la misma. Asimismo, distintos estudios indican esto mismo: la depresión afecta en mayor medida a mujeres, doblando los casos en edades de 14 a 25 años, siendo el inicio de la pubertad donde mayor riesgo existe. Antes de este momento y en edades de más de 65 años, los ratios de depresión entre hombres y mujeres son similares. A esto se le suma que las mujeres sufren formas de depresión específicas (incluyendo el trastorno disfórico premenstrual, la depresión postparto y la depresión y la ansiedad posmenopáusica) que están asociadas con cambios en las hormonas ováricas y podrían contribuir al aumento de la prevalencia en las mujeres [13].

También destacar que la pandemia ocasionada por el COVID-19 supuso un desafío en la salud mental de la población, debido a, por ejemplo, acciones tomadas por la salud pública, como distanciamiento social, que pueden hacer sentir a la gente aislada y sola. Estudios indican que a finales de junio de 2020, un 40% de adultos informaron sufrir un problema mental, como depresión o ansiedad, o consideraron cometer suicidio [14].

Dado que el objetivo principal es la elaboración temprana de un *profiling* automático desde redes sociales, nuestras fuentes de información primaria son publicaciones de redes sociales escritas en lenguaje natural. Por lo tanto, trataremos en esta tarea clasificar en “*male*” (hombre) o “*female*” (mujer) los distintos usuarios que forman parte de nuestra entrada de datos. Esta clasificación se conoce bajo el nombre de “Tarea de clasificación de género”, donde el objetivo es conocer el sexo de la persona que ha escrito las publicaciones o comentarios. Es importante resaltar la distinción entre sexo y género, dentro del contexto de los estudios de género, siendo “sexo” lo que refiere a la anatomía del sistema reproductivo y a las características sexuales secundarias, mientras que “género” se utiliza para referirse a los roles sociales basados en el sexo de la persona.

Tomaremos como referencia la definición de la Asociación Estadounidense de Psiquiatría, donde en el DSM-5, separa la definición de género del construcciónismo social y señala “[...] a diferencia de algunas teorías constructivistas sociales, se considera que los factores biológicos son los que contribuyen, en interacción con los factores sociales y psicológicos, para el desarrollo del género” [15]. Por lo que utilizaremos el término género en este documento y en el perfilado, manteniendo su relación con el sexo biológico, ya que, tal y como se comentaba antes, es de vital importancia conocer el sexo de la persona a la hora del diagnóstico psicológico. Aunque somos conscientes de que esta equiparación sexo-género está sometida a debate social y existen otros modelos teóricos, estos todavía no se están aplicando en el dominio de trabajo y por lo tanto mantendremos la equiparación como correspondencia al perfil actual que se hace a nivel médico.

De modo que, teniendo en cuenta que existen *features* puramente lingüísticas, como aspectos léxicos, *features* de estilometría, como frecuencia de palabras o términos, sociolingüísticas, incluyendo aspectos de gramáticas o vocabulario, y demás, y aplicando distintos algoritmos de aprendizaje automático, podremos predecir el género del autor.

En resumen, nuestro objetivo en esta tarea es, dados textos en lenguaje natural escritos en redes sociales, inferir el género del usuario en virtud de *features* lingüísticas y sociolingüísticas y clasificar así las publicaciones con el género del usuario.

2.1.3. Base o referencia

La clasificación de textos en función del género ha sido estudiada previamente en diversos foros y publicaciones científicas. Destaca PAN [4], una serie de eventos científicos y tareas compartidas sobre texto forense digital y estilometría. En este ejercicio la base para la evaluación que se ha tomado ha sido la que marca la competición de PAN de 2019 de “Bots and Gender Profiling 2019”.

Esta tarea consiste en dado un *feed* de *Twitter*, determinar si el autor es un *bot* o un humano y, en caso de ser un humano, identificar su género.

De cara a evaluar los resultados de nuestro propio clasificador para la detección del género, tomamos como referencia los tres mejores clasificadores de la categoría de PAN. La métrica que se utiliza para clasificar es la *accuracy* (precisión). Esta métrica refleja lo cercana una medida es hacia un valor conocido o aceptado. En *machine learning* se conoce como el número de predicciones correctas entre el total de predicciones. De una forma más formal, se define como el número de verdaderos positivos y verdaderos negativos dividido por el número de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

Por lo que, en la categoría de clasificación de género de textos en inglés, en la competición de 2019, los tres mejores resultados son:

- Valencia et al. con una precisión de **0.8432** [16]
- Bacciu et al. con una precisión de **0.8417** [17]
- Espinosa et al. con una precisión de **0.8413** [18]

2.1.4. Conjuntos de datos

Para poder llevar a cabo la tarea de clasificación de género, se han utilizado distintos conjuntos de datos etiquetados, proporcionados por PAN.

Hemos seleccionado dos *datasets* distintos que incluyen información de género para utilizar como *corpora* externa para nuestros experimentos. Los *datasets* utilizados son de los concursos “PAN Author Profiling 2019” y “PAN Celebrity Profiling 2019”, ambos en inglés.

Hemos seleccionado estos conjuntos de datos, ya que son los más recientes disponibles en el momento de ejecución de nuestros experimentos.

El primer conjunto de datos está dividido en dos partes: un *dataset* de entrenamiento y otro de prueba o validación. Ambos están formados por *bots* y usuarios humanos, ya que el objetivo de la tarea era detectar si el autor era un *bot* o un humano (y en caso de humano, su género). Por lo tanto, se eliminaron los *bots*. Hemos utilizado cada *dataset* en su fase adecuada.

El segundo conjunto de datos solo disponía de la parte de entrenamiento. En este caso, el objetivo de la tarea era predecir rasgos de celebridades a partir de su historial de *Tweets*. En lo concerniente al género, eliminamos aquellos usuarios con género “nonbinary” (ya que nuestro modelo solo contempla los géneros masculino/femenino). Este conjunto de datos solo se ha utilizado en la fase de entrenamiento.

2.1.4.1. “PAN Author Profiling 2019”

Cada usuario está representado mediante un fichero .xml donde dentro del *tag author* tenemos un *tag documents* que contiene un *document* por cada *Tweet*. Para conocer si se trata de textos de un hombre o mujer, debemos compararlo con un fichero maestro donde para cada *id* del autor, aparece acompañado de la etiqueta *male* o *female*.

Cuadro 2.1.: “Author Profiling 2019” *dataset*

	author_id	gender
0	ccbe6914a203b899882	male
1	a3b93437a32dba31def	male
2	a1655b4b89e7f4a76a9	male
3	de3eee10fbac25fe396	male
4	2a61915c1cd27b842ee	male
...
2055	f92806b515385388c83	female
2056	a820cb38384e19a3043	female
2057	f17345aeea69b649063	female
2058	f334e25ccf9a18f1eb2	female
2059	b2eb427fb56beace062	female

Disponíamos de un *dataset* equilibrado, ya que de los 2060 usuarios, 1030 eran hombres y 1030 mujeres.

Cuadro 2.2.: “Author Profiling 2019” *dataset* usuarios por género

	total
male	1030
female	1030

2.1.4.2. “PAN Celebrity Profiling 2019”

Los datos disponibles consisten en dos ficheros: el fichero `feed` que contiene el `id` del autor y una lista con todos los *Tweets* por usuarios y el fichero `labels` que contiene para cada `id` del autor, sus rasgos (fama, ocupación, año de nacimiento y género). Se han eliminado todos menos el género para los experimentos de esta tarea.

Cuadro 2.3.: “PAN Celebrity Profiling 2019” *dataset*

	author_id	gender
0	3849	male
1	957	female
2	14388	female
3	7446	male
4	1107	female
...
14494	33530	female
14495	29315	male
14496	36954	male
14497	4554	male
14498	4512	male

Tras preprocesar los datos, hemos obtenido un conjunto de datos no balanceado, ya que tenemos más del doble de usuarios del género masculino que femenino.

Cuadro 2.4.: “PAN Celebrity Profiling 2019” *dataset* usuarios por género

	total
male	10409
female	4072
nonbinary	18

Por la complejidad de la tarea y los objetivos de nuestro trabajo, estos son los conjuntos de datos que más se adecuan a nuestras necesidades, por lo que han sido los empleados. Además, nos permiten comparar nuestro resultado con una base clara.

2.1.5. Preprocesado

Antes de calcular y extraer las características del texto, se han preprocesado los datos. Para ello, se han juntado todos los comentarios en un único documento.

Se formó un documento donde la primera columna refleja el `id` del autor del texto, la segunda todos los textos concatenados (*corpus*) y la tercera el género etiquetado (lo que queremos predecir).

Es habitual que en este tipo de tareas se realicen preprocesados y limpiezas de datos entre las que se incluyen tokenización, *Stemming*, *Lemming* y eliminación de *stop words* [19]. Estas operaciones suponen la eliminación total o parcial de palabras que forman parte de la entrada de datos de nuestro clasificador y puede resultar en una acción contraproducente [20].

Por lo tanto, no se ha realizado ningún tipo de preprocesado adicional. No se ha aplicado ni *Stemming* ni *Lemming*, ya que se perdería información potencialmente relevante para la clasificación del género del autor. Por el mismo motivo tampoco se eliminaron ni *stopwords* ni caracteres especiales, ya que supondría en una reducción significativa del *corpus* y por lo tanto en una pérdida de contenido y precisión. Es de especial relevancia para el perfil contar con toda la información disponible, ya que, habitualmente, los mensajes de redes sociales contienen gran cantidad de elementos escritos en lenguaje informal, donde en un proceso tradicional de preprocesado se eliminarían.

2.1.6. Feature engineering o preparación de los datos

La idea principal tras el concepto de *feature engineering* es utilizar conocimiento de dominio para obtener *features* del *corpus* para así poder compararlas entre los distintos autores y ver patrones que tienen en común.

Las *features* extraídas han sido analizadas para conocer su relevancia y efectos en el modelo. Para esto, hemos dividido las *features* encontradas en tres grandes grupos basados en la naturaleza intrínseca de ellas.

1. *Features sociolingüísticas.*
2. Características de análisis de sentimiento.
3. *Topic modelling*

2.1.6.1. Features sociolingüísticas

La sociolingüística es el estudio de los efectos de cualquier aspecto de la sociedad en la forma en la que se usa el lenguaje [21]. En sociolingüística, el género hace referencia a la identidad sexual en lo concerniente a la cultura y sociedad. Cómo se usan las palabras puede tanto reflejar como reforzar actitudes sociales en torno al género. A partir de esta definición, se ha calculado cuántas veces una determinada *feature* estilística aparece en los textos en lo referente a esta categoría. Esta aproximación nos ayudará, a encontrar un lexicón común generalizado compartido por hombres y otro para mujeres, o inferir estructuras de uso gramaticales o discursivas diferenciadas en el género.

En particular, algunas de las *features* extraídas son:

- Uso de emojis, relacionado con la derivación léxica y formación de nuevas palabras y con las implicaciones semánticas y neurolingüísticas de las emociones y símbolos.
- Símbolos de puntuación, relacionados con la sintaxis y el análisis del discurso, tal y como las palabras y la longitud del texto.
- *Features* tales como repetición de caracteres, legibilidad y similitud del coseno, estudiadas como *features* pragmáticas, así como también URL y hashtags, que son a la vez referencias en el espacio personal-temporal.

- Las auto-referencias es comúnmente una característica que se añade a las *stopwords* y, por lo tanto, no se registra, pero tiene un efecto sociolingüístico muy potente en la tarea de clasificación.
- Part-of-Speech (POS) como POS Tags son rasgos sociolingüísticos que tienen en cuenta la sintaxis, pero también está relacionado con el análisis del habla y discurso.

En la tabla 2.5 se puede ver un resumen de las *features* sociolingüísticas analizadas.

Cuadro 2.5.: Descripción de las características sociolingüísticas

Nombre	Descripción
Uso de emojis	Ratio de emojis entre los documentos
Caracteres especiales	Ratio de caracteres especiales entre los documentos
Símbolos de puntuación	Ratio de símbolos de puntuación (? ! . , ; :) entre los documentos
URL, hashtag	Por separado, ratio de URL y hashtags entre los documentos
Tokens	Ratio de palabras entre los documentos
Longitud palabras y texto	Longitud media de palabras y texto
POS Tags	Part-of-speech tagging: ratio entre los documentos
Repetición de caracteres	Ratio de caracteres repetidos entre los documentos
Autorreferencias	Ratio de frases con referencias a uno mismo
Legibilidad	Métrica de la facilidad con la que se puede enter un texto escrito
Similitud del coseno	Medida de la similitud entre dos documentos entre su totalidad

2.1.6.2. Características de análisis de sentimiento

El análisis de sentimiento es el campo de estudio que analiza las opiniones de las personas, sentimientos, emociones, etc. a partir del lenguaje escrito [8]. En este proceso, normalmente se intenta determinar si un fragmento escrito es positivo, negativo o neutral.

El análisis de sentimiento nos ayuda a entender las experiencias del autor y puede ser un patrón diferenciador entre hombres y mujeres. Por lo tanto, la información extraída gracias a esta aproximación constituye un factor a tener en cuenta en el modelo de clasificación de género.

2.1.6.3. *Topic modelling*

Latent Dirichlet Allocation (LDA) es un modelo de temas propuesto por David Blei et al [22]. Se utiliza para clasificar un texto en un documento que hace referencia a un tema particular. Es un modelo generativo estadístico no supervisado que construye un tema por cada documento y asocia palabras para cada tema, modelado como distribuciones Dirichlet.

LDA se utiliza comúnmente para extracción automática y para encontrar patrones escondidos entre el corpus. Esta característica nos puede ayudar a modelar relaciones entre temas disgregados por género.

Se ha ejecutado esta aproximación, dividiendo en hombres y mujeres para obtener los temas más representativos para cada género, y utilizar los temas como *features* en nuestro modelo.

El género del autor o autora se infiere de los textos, a partir de las *features* mencionadas y el algoritmo utilizado, no existe ningún tipo de meta-information para la construcción del modelo.

2.2. Tarea de medición de la gravedad de los signos de depresión**2.2.1. Proceso de trabajo**

En la siguiente figura 2.2 se puede ver el proceso de trabajo seguido a la hora de realizar la tarea de medición de la gravedad de los signos de depresión. Este *workflow* de experimentación está compuesto por las fases de estudio y entendimiento del objetivo de la tarea (*Task Goal*), investigación sobre la base o referencia de la misma (*Task Reference*), selección de conjuntos de datos (*Datasets Selection*) y preprocesado (*Preprocessing*). Tras completar estos pasos, da comienzo la fase de experimentación (*Experiments*) y tras esta obtendremos el resultado de la tarea.

En este capítulo se explicarán en detalle los cuatro primeros pasos, delegando los restantes al capítulo 10.

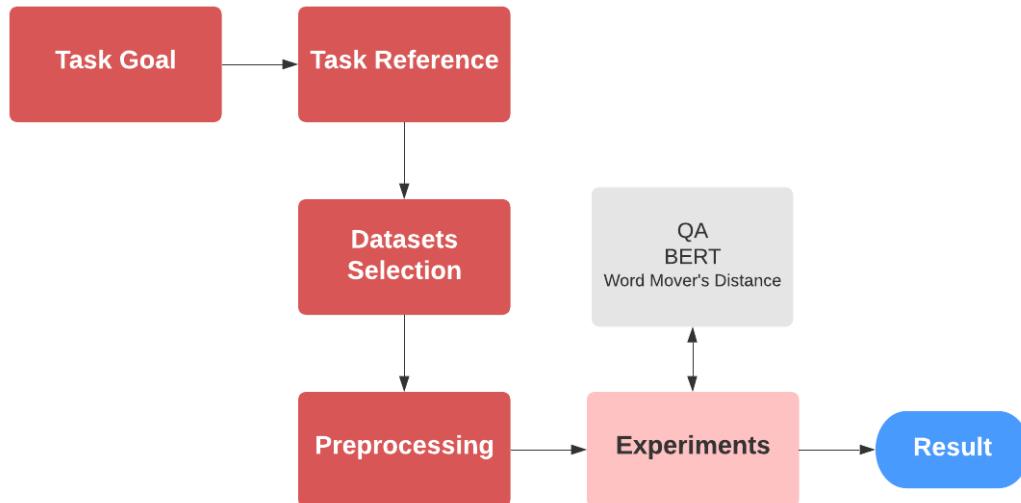


Figura 2.2.: Diagrama del proceso de trabajo de la tarea de medición de la gravedad de los signos de depresión.

2.2.2. Objetivo de la tarea

La tarea de medición del nivel de depresión de una persona consiste en estimar el nivel de depresión de esta, a partir de comentarios o publicaciones en redes sociales. Por lo tanto, a partir de un conjunto de datos no estructurados, se deben encontrar evidencias de algún signo de depresión que el usuario pueda sufrir [23].

Existen distintas técnicas para medir el nivel de depresión de una persona. Una de las más conocidas es a partir del cuestionario de Beck “Beck’s Depression Inventory” (BDI) [24]. Este es un inventario de autoevaluación que, a partir de 21 preguntas, mide las actitudes y síntomas característicos de la depresión. Evalúa la presencia de sentimientos como tristeza, pesimismo, falta de energía, apetito, etc. para poder detectar este trastorno.

La presente tarea tiene como objetivo autocompletar de forma automática este cuestionario a partir de la información disponible en el historial social del usuario. De esta forma se deberá predecir la respuesta a cada pregunta (0-3) e indicar cuál es el nivel de depresión del usuario.

Cada respuesta tiene una puntuación entre 0 y 3, y la suma de estas da como resultado el nivel de depresión. Siento los distintos niveles:

- *minimal depression* (nivel de depresión 0-9)

- *mild depression* (nivel de depresión 10-18)
- *moderate depression* (nivel de depresión 19-29)
- *severe depression* (nivel de depresión 30-63)

2.2.3. Base o referencia

La tarea de medición del nivel de depresión ha sido estudiada en competiciones de eRisk [23, 25]. Desde el 2019 realizan esta competición. Para este trabajo, se ha tomado como base los resultados que marca la competición de 2019 de “Measuring the severity of the signs of depression”.

En esta competición se definen distintas métricas para evaluar la calidad de las respuestas que autocompleta el sistema, contra las respuestas reales de los usuarios.

- **Average Hit Rate** (AHR): media de respuestas acertadas entre todos los usuarios. Con esta métrica se pretende medir el ratio de casos donde la respuesta automática tiene exactamente el mismo valor que la real.
- **Average Closeness Rate** (ACR): cercanía media (CR) de las respuestas entre todos los usuarios. Esta métrica tiene en cuenta que las respuestas del cuestionario representan una escala ordinal, y penaliza más a un sistema cuya predicción es más lejana a la respuesta real. Para cada pregunta, se calcula la diferencia absoluta (*ad*) entre la respuesta real y automática y, a continuación, esta diferencia absoluta se trasforma en la puntuación efectiva con la siguiente fórmula: $CR = (mad - ad)/mad$, donde *mad* es la diferencia máxima absoluta.
- **Average DODL** (ADODL): diferencia general del nivel de depresión (DODL) medio entre todos los usuarios. Como la métrica anterior, pero computando el nivel de depresión general (la suma de todas las respuestas) entre la puntuación real y automática. Como el nivel de depresión es un número entre 0 y 63, DODL se normaliza en [0,1] de la siguiente forma: $DODL = (63 - ad_overall)/63$.
- **Depression Category Hit Rate** (DCHR): fracción de casos donde las respuestas automáticas predicen la misma categoría de depresión a la obtenida del cuestionario real.

Sabiendo esto, los mejores resultados obtenidos en el año 2019 para cada métrica son:

- **AHR:** 41.43 % (UNSLC [26])
- **ACR:** 71.27 % (UNSLE [26])
- **ADODL:** 81.03 % (CAMH_GPT_nearest_unsupervised [23])
- **DCHR:** 45.00 % (CAMH_GPT_nearest_unsupervised [23])

2.2.4. Conjuntos de datos

Para la ejecución de la tarea, se han utilizado los datos proporcionados por eRisk, ya que es la única fuente de datos con la que podemos comparar nuestro resultado. Por lo tanto, se ha utilizado el *dataset* de la competición de eRisk 2019 “Measuring the severity of the signs of depression”.

Este conjunto de datos está dividido en dos partes: un *dataset* con el historial de publicaciones en Reddit de los usuarios y un fichero maestro con el identificador del usuario y las respuestas a las preguntas del inventario de Beck.

Cada usuario está representado mediante un fichero .xml donde tenemos un *tag id*, que contiene el identificador del usuario, y una lista de *writing*, conteniendo cada uno la fecha (*date*), el título de la publicación si es el caso de una nueva publicación (*title*), y el texto (*text*), para respuestas a publicaciones existentes.

En total se disponen de 20 usuarios, con su historial de actividad en Reddit y las respuestas reales al cuestionario de Beck.

Cuadro 2.6.: “Measuring the severity of the signs of depression 2018” *dataset*

	experiment_id	real answers
0	subject2341	1 2 3 2 3 2 3 3 1 0 0 2 3 3 2 2b 2 3b 2 2 0
1	subject2827	1 3 3 2 3 2 2 3 2 2 1 1 2 3 1 2b 1 0 2 2 1
...
18	subject5791	1 0 1 1 0 0 2 1 0 0 0 1 0 1 1 1b 1 0 1 0 0
19	subject6900	1 1 2 1 0 0 1 1 1 1 1 1 0 1 2 2a 0 0 0 1 0

Este conjunto de datos se ha utilizado exclusivamente para validar el modelo, y así compararlo con la competición de eRisk 2019.

Para entrenar el modelo no se ha utilizado ningún conjunto de datos, sino que se ha relegado el trabajo en el modelo y en SQuAD, explicado en la sección 10.2.

2.2.5. Preprocesado

Para poder evaluar nuestro modelo, se ha realizado un preprocesado de los datos de validación para que nuestro modelo los entienda. Para ello, cada fichero .xml se ha convertido en un fichero .json que siga la especificación del API en el que está implementado nuestro modelo.

El .json necesario está formado por un campo experiment_id y otro comments, que contiene una lista de los textos del usuario y su fecha de publicación.

No se ha realizado ningún preprocesado más sobre los datos, ya que se tratan datos de validación del modelo, y necesitamos que esté toda la información posible para no perder ningún dato que pueda ser relevante para alguna pregunta.

Capítulo 3

FUNDAMENTOS TECNOLÓGICOS Y HERRAMIENTAS UTILIZADAS

Para el desarrollo del proyecto se han utilizado una serie de tecnologías y herramientas que han facilitado la labor del desarrollo.

En este capítulo se explican brevemente esas tecnologías y herramientas y el uso que se les ha dado, así como posibles alternativas, junto con la infraestructura utilizada durante la ejecución del trabajo.

3.1. Tecnologías

Esta sección se divide en dos partes: aplicación web y servicios, y perfilador. La aplicación web y los servicios suponen el software que permite la gestión de los distintos perfiles, realizar las tareas de carga de datos y validación. Por otro lado, el perfilador está formado por los distintos clasificadores y modelos de ML y NLP, los cuales suponen el núcleo de este proyecto.

3.1.1. Aplicación web y servicios

En este apartado hablaremos de las tecnologías utilizadas para la construcción de la aplicación web y del microservicio de perfilado con el que se comunica. La primera se ha construido principalmente con Django, OpenAPI, PostgreSQL, PRAW, Redis y Celery y el segundo se ha implementado utilizando Flask e incluyendo las dependencias del perfilador, que se comentarán

en la siguiente sección.

3.1.1.1. Python



Python¹ es un lenguaje de programación interpretado, de alto nivel y de propósito general. La filosofía de diseño de Python enfatiza la legibilidad del código con su notable uso de espacios en blanco significativos. Sus construcciones de lenguaje y su enfoque orientado a objetos tienen como objetivo ayudar a los programadores a escribir código claro y lógico para proyectos de pequeña y gran escala.

Python tiene un tipado dinámico y admite múltiples paradigmas de programación, incluyendo estructurado (particularmente, procedimental), orientado a objetos y funcional.

Se ha utilizado Python como lenguaje tanto para la implementación de la aplicación web, el microservicio de perfilado y los modelos para el perfilado.

Existen numerosas alternativas a Python. Como las aplicaciones de Python en este trabajo fin de máster son: desarrollo de una aplicación web y construcción de modelos de aprendizaje automático con tecnologías de procesamiento de lenguaje natural, vamos a exponer las alternativas para cada grupo.

Respecto a la aplicación web, las alternativas más populares son Java, C#, Ruby, Golang, NodeJS, etc. Las ventajas más importantes de usar Python frente a estas opciones son:

- Facilidad de uso. Es un lenguaje fácil de aprender en comparación con los otros. Es simple y es sencillo programar y debuggear.
- Código más corto. Python permite desarrollar una aplicación web con menos código que otros lenguajes como Java o C#. Gracias a esto el código de Python es más fácil de leer y más eficiente, por lo que la simplicidad de este lenguaje hace que Python sea uno de los mejores lenguajes para el desarrollo web.

¹<https://www.python.org/>

- Flexibilidad. Python permite la integración con muchos otros lenguajes y también permite ejecutar Python en distintos escenarios.
- Django. El *framework* web más popular de Python es Django, siendo Flask un *framework* muy potente también. Django ofrece una gran ventaja competitiva a la hora de construir tu aplicación web, por lo que otro motivo para utilizar Python es poder disfrutar de todas las ventajas que ofrece Django (explicadas en la siguiente subsección).

En lo que respecta a la parte de aprendizaje automático y procesado de lenguaje natural:

- Python encaja perfectamente en este ámbito gracias a su plataforma independiente y su popularidad en la comunidad de desarrolladores de software.
- Es flexible, simple por lo que es lo más apropiado para *machine learning*.
- Es independiente a través de plataformas. Se puede ejecutar en distintas plataformas sin requerir cambios o con muy pocos cambios.
- La variedad de *frameworks* y librerías para aprendizaje automático y procesamiento de lenguaje natural, hacen a Python un lenguaje muy atractivo.

3.1.1.2. Django



Django² es un *framework* web de Python de alto nivel, que permite un desarrollo rápido y un diseño limpio y pragmático. Permite al desarrollador centrarse en escribir la aplicación sin necesidad de reinventar la rueda. Es gratuito y de código abierto.

Algunas de sus características principales son:

- **Rapidez.** Django fue diseñado para ayudar a los desarrolladores a llevar aplicaciones desde el concepto hasta su finalización lo más rápido posible.

²<https://www.djangoproject.com/>

- **Seguridad.** Django es un *framework* seguro y ayuda a los desarrolladores a evitar muchos errores de seguridad comunes.
- **Escalabilidad.** Permite escalar de forma rápida y flexible aplicaciones web.

Se ha utilizado Django para la realización de la aplicación web.

- Django ofrece la posibilidad de diseñar el modelo, aunque también se puede usar sin base de datos. Django viene con un mapeador relacional de objetos en el que puedes describir el diseño de la base de datos en código Python.
- Django crea las tablas de la base de datos y gestiona con ayuda de un par de comandos las migraciones de tu proyecto.
- Ofrece también una API para acceder a los datos que crea al vuelo, sin necesidad de generar código a mayores.
- Gracias a las *views*, Django permite gestionar los *endpoints* de la aplicación y recuperar los datos acordemente o lanzar una excepción HTTP.
- Permite diseñar las interfaces por medio de *templates*. Gracias a estos componentes, es posible minimizar la redundancia entre *templates*, acceder a variables de las vistas y heredar de otros *templates*.

Existen numerosas alternativas a Django, pero una de las principales es Flask. Aunque en este trabajo de fin de máster sí se ha utilizado Flask, no se decidió utilizarlo para la aplicación web. Flask es idóneo para una aplicación muy básica, que no necesite base de datos, operaciones de ficheros o nada muy complejo. Ya que nuestro sistema precisa de base de datos, una API y un cliente web, se optó por utilizar Django, ya que es un *framework* que nos ofrece directamente la posibilidad de trabajar con todo esto.

3.1.1.3. Django REST Framework



Django REST framework³ es un conjunto de herramientas muy potente y flexible para crear una API web.

Django REST framework es una tecnología que encaja a la perfección con la forma de trabajo llevada a cabo: *API First development*. *API First development* es una aproximación de trabajo en la que el diseño y desarrollo del API se realiza antes que la implementación. Una vez se ha definido esta, se confía en esta interfaz y se procede a construir el resto de la aplicación.

Django REST Framework ofrece:

- Una API navegable en la web.
- Políticas de autenticación que incluyen paquetes para OAuth1 y OAuth2.
- Serialización que admite fuentes de datos ORM y no ORM.
- Personalización.

Gracias a Django REST framework nuestra aplicación Django se ve complementada con:

- Serializadores. Suponen una representación de los datos que permitimos recibir o que enviamos como respuesta en nuestras peticiones.
- Vistas. Ofrece la posibilidad de agrupar todas las vistas en una única clase (`ViewSets`), así tenemos nuestra lógica bien organizada y concisa.
- URL. Generadas automáticamente y permitiendo registrarlas para tener un mayor control sobre estas.

Una alternativa a Django REST framework es TastyPie. Es un servicio web API para el *framework* de Django. Se decidió por Django REST framework, ya que es más cercano a Django, tanto en estilo como expresiones, por lo que hacen la combinación perfecta.

3.1.1.4. OpenAPI



³<https://www.django-rest-framework.org/>

OpenAPI⁴ es un *framework* de código abierto (licencia Apache 2.0) que sirve para documentar API⁵ RESTful de manera automática. Anteriormente era conocido como Swagger. Este *framework* genera una página web, a partir de un servicio REST, desde la que se puede consultar y comprobar, mediante llamadas reales, el funcionamiento del API.

OpenAPI complementa la aproximación *API First development* ya mencionada, ya que de esta forma podemos disponer de los *endpoints* y su documentación antes de realizar la implementación de estos.

Las especificaciones del API se pueden escribir tanto en YAML como JSON. Ambos son formatos sencillos de aprender y leer.

Este *framework* también facilita, al disponer de la documentación de la API, la posibilidad de crear diferentes clientes del servicio, como pueden ser aplicaciones móviles, extensiones de navegador o aplicaciones de escritorio, sin necesidad de conocer cómo está implementado el Back-End.

Tanto para la aplicación como para el microservicio del perfilado, se documentó la API con OpenAPI.

Existen alternativas como RAML O Apiary. Se decidió optar por OpenAPI, ya que es más popular y existe más documentación y comunidad a su alrededor. Además, define el estándar más conocido para la especificación de API REST.

3.1.1.5. PRAW



PRAW (Python Reddit API Wrapper)⁶ es un paquete de Python que permite un acceso simple a la API de Reddit. Su objetivo es ser fácil de usar y sigue internamente todas las reglas de la API de Reddit.

Se podría haber consumido directamente la API de Reddit, pero para simplificar el acceso se

⁴<https://www.openapis.org/>

⁵Interfaces de Programación de Aplicaciones

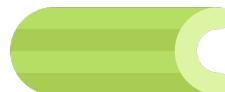
⁶<https://praw.readthedocs.io/en/latest/>

decidió añadir este paquete.

Gracias a ello, podemos acceder a subreddit, recuperar usuarios que hayan comentado este subreddit y de cada usuario, sus comentarios, permitiendo indicar el número de comentarios y demás.

Se desconoce si existe alguna alternativa que ofrezca la misma facilidad y flexibilidad a la hora de consumir la API de Reddit.

3.1.1.6. Celery



Celery⁷ es un sistema distribuido simple, flexible y confiable para procesar grandes cantidades de mensajes, mientras proporciona las herramientas necesarias para mantener dicho sistema. En otras palabras, es una lista de tareas en cola que se centra en el procesamiento en tiempo real, al mismo tiempo que admite programación de tareas. Celery tiene una comunidad grande y diversa.

Se ha utilizado Celery para procesar de forma asíncrona los datos de los usuarios recuperados de Reddit. De esta forma, se solicitan grandes cantidades de datos a Reddit, se añade a la cola una tarea por usuarios y se realiza una llamada al servicio del perfilador para obtener la clasificación del usuario. De esta forma, podemos procesar con una única acción numerosas peticiones a la API de Reddit.

Existen alternativas como dramatiq o RQ. Todos son compatibles con Redis (se comentará a continuación), pero se decidió utilizar Celery, ya que tiene mayor robustez y está más establecido.

⁷<https://docs.celeryproject.org/en/stable/>

3.1.1.7. Redis

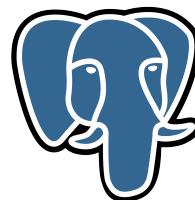
Redis⁸ es una estructura de datos en memoria de código abierto, que se utiliza como base de datos, caché y agente de mensajes. Redis proporciona estructuras de datos como cadenas, hashes, listas, conjuntos, mapas, etc. Redis tiene replicación incorporada, transacciones y diferentes niveles de persistencia en el disco.

Para lograr el máximo rendimiento, Redis trabaja con un conjunto de datos en memoria. También permite replicación asíncrona, con una primera sincronización sin bloqueo muy rápida, reconexión automática con resincronización parcial en la división de red.

Redis se usa para almacenar mensajes producidos por el código de la aplicación, que describen el trabajo a realizar en la cola de tareas de Celery.

Las alternativas más populares a Redis son RabbitMQ y Kafka. Redis y Kafka tienen una mayor escalabilidad que RabbitMQ, Redis no permite persistencia, es un almacenamiento en memoria. RabbitMQ es más adecuado cuando el enrutado es complejo. Por otro lado, Kafka está orientado para grandes cantidades de datos. Y Redis para mensajes de corta vida. Nuestro caso es este último, ya que, una vez obtenemos los usuarios de Reddit a procesar, se irán procesando las tareas poco a poco, pero el tiempo es corto, en cuestión de minutos estará todo procesado.

⁸<https://redis.io/>

3.1.1.8. PostgreSQL

PostgreSQL⁹ es un potente sistema de base de datos relacional de objetos de código abierto. Está en desarrollo activo y es muy confiable, sólido y ofrece un alto rendimiento.

PostgreSQL tiene muchas características que permiten a los desarrolladores crear aplicaciones, proteger la integridad de los datos y crear entornos tolerantes a fallos. Es altamente extensible y permite definir tipos de datos propios, crear funciones personalizadas, etc.

Los datos que se van a almacenar son siempre los mismos y conocidos de ante mano (datos demográficos, textos de redes sociales, usuarios, etc.) por lo que se ha optado por una base de datos relacional. Entre estas, las alternativas más conocidas son MySQL, Oracle, Mariadb, SQLite, etc. Las dos bases de datos más adecuadas para aplicaciones Python son PostgreSQL y MySQL. SQLite es una base de datos que se almacena en un único fichero en disco y solo permite una conexión cada vez, por lo que se descartó su utilización. PostgreSQL es más recomendada para trabajar con aplicaciones web Python, por lo que se optó por esta.

3.1.1.9. Flask

⁹<https://www.postgresql.org/>

Flask¹⁰ es un *framework* que proporciona herramientas, bibliotecas y tecnologías que permiten crear una aplicación web.

Flask es un microframework que se centra en la simplicidad, el minimalismo y el control de grano fino. Implementa lo mínimo, dejando al desarrollador con total libertad de elección en términos de módulos y complementos.

Se optó por utilizar Flask para la implementación del microservicio de perfilado, ya que necesitábamos únicamente unos *endpoint* que permitieran el perfilado de los usuarios. Para esto, es necesario añadir como dependencias todas las librerías de procesado de lenguaje natural y modelos de clasificación, por lo que se procuraba buscar un *framework* muy ligero para esta tarea.

Django es la alternativa más conocida a Flask, pero, como ya se comentó en su subsección, Django es más pesado que Flask y proporciona componentes y utilidades que no se necesitan para este microservicio, por lo que se rechaza utilizar Django y se opta por Flask.

3.1.1.10. HTML



HTML¹¹, siglas de HiperText Markup Language (lenguaje de marcación de Hipertexto), es el lenguaje de referencia para la elaboración de páginas web.

Es un estándar a cargo de la W3C (World Wide Web Consortium), organización dedicada a la estandarización de prácticamente todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación.

HTML define una estructura básica y un código (código HTML) para la definición de contenido de una página web, como texto, imágenes, vídeos, etc.

¹⁰<https://flask.palletsprojects.com/en/1.1.x/>

¹¹<https://html.spec.whatwg.org/>

3.1.1.11. CSS

CSS¹² es el lenguaje de hojas de estilos para describir la presentación de las páginas web, incluyendo colores, diseños y fuentes. Permite adaptar la presentación a diferentes tipos de dispositivos y tamaños de pantallas.

CSS se utiliza para dar estilo a documentos HTML y XML, separando el contenido de la presentación. Los estilos definen la forma de mostrar los elementos HTML y XML. CSS permite a los desarrolladores web controlar el estilo y el formato de múltiples páginas web al mismo tiempo. Cualquier cambio en el estilo marcado para un elemento en la CSS afectará a todas las páginas vinculadas a ese CSS en las que aparezca dicho elemento. Es importante la separación del CSS y el HTML para el mantenimiento y la reutilización de páginas, así como para lograr un diseño uniforme en toda la aplicación.

3.1.1.12. Bootstrap

Bootstrap¹³ es un *framework* CSS/JavaScript de código abierto, originalmente desarrollado por Twitter, para implementar la capa vista de una aplicación web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basados en HTML y CSS, también posee unas extensiones opcionales de JavaScript. Las páginas web hechas con Bootstrap se adaptan automáticamente al espacio disponible según el tipo de dispositivo, consiguiendo así un diseño *responsive*.

¹²<https://www.w3.org/Style/CSS/Overview.en.html>

¹³<https://getbootstrap.com/>

Una alternativa muy popular es Boilerplate. La diferencia principal con Bootstrap es que no es un *framework*, es una plantilla. Es un paquete con contratos útiles, técnicas y librerías, Bootstrap contiene reglas bien definidas para cada elemento de una interfaz. Esta diferencia primordial fue la que hizo decantarse por Bootstrap.

3.1.1.13. Locust



Locust¹⁴ es una herramienta de pruebas de rendimiento escalable, programable y fácil de usar. Permite definir el comportamiento de los usuarios en código Python, sin necesidad de interfaces de usuario o un lenguaje específico de dominio. Locust es expandible y muy amigable.

Locust permite:

- Escribir escenarios de prueba en código Python. Permite detallar tanto ciclos de comportamiento, como peticiones sencillas. Locust ejecuta cada usuario en una co-rutina.
- Distribuido y escalable, admitiendo cientos de miles de usuarios. Está basado en eventos, lo que hace posible que un solo proceso maneje muchos miles de usuarios concurrentes.
- Interfaz de usuario basada en web. Aunque no sea requerido lanzar las pruebas por la interfaz de usuario, Locust provee una interfaz sencilla de usar que muestra el progreso en tiempo real.
- Permite probar cualquier sistema. Aunque está pensada para probar sitios web o servicios, permite probar casi cualquier sistema o protocolo.

La alternativa principal a Locust es Gatling. Gatling permite escribir *tests* de rendimiento en lenguaje Scala y probar así la aplicación. Se decidió utilizar Locust, ya que su integración con una aplicación Python/Django es más sencilla que Gatling.

¹⁴<https://locust.io/>

3.1.1.14. Behave



Behave¹⁵ es un *framework* BDD (Behaviour-driven Development) para Python. BDD es una práctica de desarrollo de software ágil que mejora el paradigma de TDD (Test Driven Development) y las pruebas de aceptación, y fomenta la colaboración entre desarrolladores, QA (aseguramiento de la calidad), expertos en el dominio y partes interesadas.

En Behave, se definen historias como pruebas de aceptación (escenarios) antes de realizar la implementación. Se realiza con lenguaje Gherkin que sigue la siguiente estructura:

Given...

When...

Then...

Listing 3.1: Función que realiza la validación de un perfil en el navegador

Estos tests se podrán ejecutar contra la aplicación, con la ayuda de Selenium WebDriver (se explicará en la siguiente sección), donde se indicará el navegador a iniciar y ejecutará los distintos escenarios uno detrás de otro.

La alternativa más popular a Behave es Cucumber. También sigue una aproximación de BDD basada en Gherkin. Se descartó ya que para poder utilizar Cucumber en un proyecto Python, es necesario utilizar `rubypython` para lanzar un intérprete dentro de un proceso Ruby. Por lo que por simplicidad y para utilizar un *framework* escrito en Python que se integra más fácilmente con nuestra aplicación, se decidió emplear Behave.

3.1.1.15. Selenium WebDriver



¹⁵<https://behave.readthedocs.io/en/stable/>

Selenium¹⁶ es un *framework* de pruebas automatizado que se utiliza para validar aplicaciones web en diferentes navegadores y plataformas. Se puede utilizar en varios lenguajes de programación como Java, C#, Python, etc.

Selenium tiene a disposición de sus usuarios “Selenium Web Driver”, esto es una colección de API de código abierto que se utilizan para automatizar las pruebas de una aplicación web.

En nuestro caso se han utilizado Selenium WebDriver junto con Behave para ejecutar los tests de interfaz. Para nuestro conjunto de tests de aceptación, se lanza un navegador y se ejecutarán de forma automática y automatizada los tests implementados con Behave. De esta forma conseguimos ejecutar nuestras pruebas E2E (end-to-end).

Una de las alternativas más conocidas a Selenium compatibles con aplicaciones Python es Endtest. Se descartó su utilización, ya que se carecía de experiencia con esta librería y Selenium es la solución más usada por la comunidad y existe suficiente documentación y soporte.

3.1.1.16. Docker



Docker¹⁷ es una plataforma abierta para desarrollar, enviar y ejecutar aplicaciones. Docker permite separar las aplicaciones de la infraestructura para que sea posible entregar software rápidamente. Con Docker, se puede administrar la infraestructura de la misma forma que se administran las aplicaciones.

Los contenedores Docker proporcionan una forma de controlar el software. Se puede ejecutar un contenedor Docker en cualquier host compatible con el sistema operativo que tenga Docker runtime instalado.

Docker ofrece muchos beneficios tales como encapsulación, aislamiento, portabilidad y con-

¹⁶<https://www.selenium.dev/>

¹⁷<https://www.docker.com/>

trol. Los contenedores Docker son pequeños y se pueden compartir fácilmente a través del Docker Hub (se explicará en la sección de herramientas) público o del repositorio privado.

Los contenedores funcionan de una forma similar a las VM (Virtual Machines), pero de una forma más específica y granular.

Docker permite:

- Un uso más eficiente de los recursos.
- Ciclos de entrega de software más rápidos.
- Portabilidad de aplicaciones.
- Seguir una arquitectura de microservicios más fácilmente.

Por todas estas razones se decidió utilizar contenedores Docker para lograr tener cada componente de forma aislada y su despliegue dentro del ciclo CI/CD implementado (se comentará en secciones posteriores) se ve simplificado y se realiza de una forma más eficiente y rápida.

Docker define dos ficheros: `Dockerfile` y `docker-compose.yml`.

`Dockerfile` es un documento de texto que contiene todos los comandos que un usuario puede ejecutar en la línea de comandos para ensamblar una imagen. Las instrucciones se ejecutan en orden, siendo la primera de ellas `FROM`, donde se indica la imagen desde la que se construye. En este fichero se definieron las imágenes desde las que se construye nuestras aplicaciones, instrucciones para instalar dependencias, los requerimientos, los puertos expuestos y los comandos para iniciar nuestros servicios.

`Compose` es una herramienta para definir y ejecutar aplicaciones Docker de varios contenedores. Gracias al fichero `docker-compose.yml`, se puede configurar los servicios de la aplicación para que se puedan ejecutar juntos en un entorno aislado.

La forma de trabajo con estos dos ficheros es:

1. Definir el entorno con un `Dockerfile` para que sea reproducible en cualquier sitio.
2. Definir los servicios que componen la aplicación en el fichero `docker-compose.yml` para que puedan ejecutarse juntos en un entorno aislado.
3. Ejecutar `docker-compose up` para levantar y ejecutar toda la aplicación.

De esta forma, como nuestro sistema está compuesto por dos servicios (aplicación web y perfilador), cada uno se define en un `Dockerfile` y cada componente de la aplicación web tiene definido sus servicios en el `docker-compose.yml` (celery, redis, base de datos).

Existen distintas alternativas a Docker como RKT, PodMan o LXC (Linux Containers). Ninguna de ellas tiene la repercusión, comunidad, documentación y facilidad de uso que Docker, por lo que se decidió utilizar Docker. Otra opción habría sido no utilizar nada, pero de esa forma sería muy laborioso realizar cada despliegue.

3.1.2. Perfilador

En este apartado hablaremos de las tecnologías utilizadas para la construcción de los clasificadores y modelos. Para realizar la implementación de los clasificadores de aspectos demográficos y para predecir si un usuario puede sufrir de depresión, se ha utilizado un *stack* tecnológico de Python, con librerías como spaCy para el procesado de lenguaje natural, NLTK y LightGBM para modelos y clasificadores y otras librerías como pandas para manejar los datos.

3.1.2.1. spaCy



spaCy¹⁸ es una librería de código abierto para el procesamiento de lenguaje natural en Python. Cuenta con NER (Name Entity Recognition), etiquetado POS (Part-Of-Speech), análisis de dependencia, vectores de palabras y más.

spaCy está diseñada para construir productos reales y es capaz de procesar tareas a gran escala. Interopera con TensorFlow, PyTorch, scikit-learn y el resto del ecosistema de inteligencia artificial de Python. Con spaCy se puede construir modelos lingüísticos sofisticados para una variedad de problemas de NLP (Natural Language Processing).

Las funcionalidades principales de spaCy son:

¹⁸<https://spacy.io/>

- Tokenización.
- NER
- Soporte para más de 61 idiomas.
- 46 modelos estadísticos
- Vectores de palabras pre entrenados.
- Integración con *deep learning* sencillas.
- POS tagging
- Análisis de dependencias etiquetadas.
- Visualizadores para sintaxis y NER.

Las alternativas más populares a spaCy son NLTK, openNLP, Stanford CoreNLP, PyNLPI, Google Cloud Natural Language API, etc. Tanto openNLP como Stanford CoreNLP han sido concebidas para trabajar con Java, ya que nuestra base está en Python, se descartaron rápidamente. PyNLPI no es una alternativa muy popular ni con el soporte y potencia de spaCy, por lo que también se descartó su uso. Finalmente Google Cloud Natural Language API es muy potente, pero al ser de pago, también se descartó su uso. NLTK es una alternativa atractiva a spaCy, pero no tan potente. De todas formas, se hizo un uso combinado de spaCy y NLTK para la tarea de procesado de los textos.

3.1.2.2. NLTK



NLTK (Natural Language Toolkit)¹⁹ es una plataforma líder para crear programas Python que funcionen con datos de lenguaje natural. Proporciona interfaces fáciles de usar para más de

¹⁹<https://www.nltk.org/>

50 corpus y recursos léxicos como WordNet, junto con un conjunto de bibliotecas de procesamiento de texto para clasificación, tokenización, derivación, etiquetado, análisis y razonamiento semántico.

Como se mencionó anteriormente, las alternativas de NLTK son las mismas que spaCy, y se han usado tanto NLTK como spaCy.

NLTK fue creada para académicos e investigadores como herramienta para ayudar a crear funciones complejas de NLP. spaCy es un servicio que ayuda a realizar tareas específicas. Por esto, se ha hecho uso de ambas:

- NLTK es idónea para construir algo desde cero o para trabajos de investigación. Sus módulos son fáciles de construir y es la tecnología para explorar ideas.
- spaCy es la tecnología adecuada para desarrollar aplicaciones. Es más rápida y preciso que NLTK. También ofrece acceso a vectores de palabras más grandes y personalizables.

Por esto mismo, dependiendo de la altura del desarrollo del trabajo, se utilizó una u otra librería.

3.1.2.3. scikit-learn



scikit-learn²⁰ es una librería que contiene muchas herramientas para el aprendizaje automático y el modelado estadístico, que incluyen clasificación, regresión, agrupamiento y reducción de dimensionalidad.

scikit-learn proporciona métodos y herramientas para aprendizaje supervisado (modelos lineales, SVM, etc.), aprendizaje no supervisado (clustering, redes neuronales, etc.), selección y evaluación de modelos (cross-validation, ajuste de hiperparámetros, etc.), visualización, transformación de *datasets* y mucho más.

Se han utilizado las implementaciones de los modelos de aprendizaje de esta librería para nuestros distintos clasificadores del perfilador.

²⁰<https://scikit-learn.org/stable/>

Algunas alternativas de scikit-learn son MLlib (una librería de machine learning de Spark) y Weka (para código Java). Weka se descartó, ya que estamos trabajando con Python y MLlib también, ya que nuestro equipo cuenta con la suficiente memoria RAM para las ejecuciones y porque en caso de notar que necesitamos más recursos, utilizaremos una máquina de AWS.

3.1.2.4. LightGBM



LightGBM²¹ es un *framework* de *gradient boosting* que utiliza algoritmos de aprendizaje basados en árboles. Está diseñado para ser distribuido y eficiente. Sus ventajas son:

- Velocidad de entrenamiento más rápida y mayor eficiencia.
- Menor uso de memoria.
- Mejor precisión.
- Soporte de aprendizaje paralelo y GPU.
- Capaz de manejar datos a gran escala.

LightGBM se está utilizando últimamente en muchas soluciones que están liderando concursos de aprendizaje automático.

Los experimentos de comparación en conjuntos de datos públicos muestran que LightGBM puede superar los *frameworks* de *boosting* existentes tanto en eficiencia como en precisión, con un consumo de memoria significativamente menor.

Por estos motivos, se ha decidido utilizar LightGBM para experimentar con los clasificadores de nuestro perfilador. No ha sido el único *framework* o algoritmo utilizado, sino que también se ha hecho uso de otros proporcionados por scikit-learn.

²¹<https://lightgbm.readthedocs.io/en/latest/>

3.1.2.5. Pyphen



Pyphen²² es un módulo de Python para dividir palabras utilizando diccionarios de división de palabras Hunspell. Incluye diccionarios para varios idiomas y no tiene dependencias externas. Almacena en caché archivos dict y palabras con guiones y admite patrones de división de palabras no estándar.

Se desconocen alternativas válidas para Python que realicen esta función.

3.1.2.6. Gensim



Gensim²³ es una librería de código libre para procesado de lenguaje natural y *topic modeling* no supervisado. Utiliza modelos de aprendizaje automático estadístico moderno para dar solución a problemas como la construcción de documentos o vectores de palabras, corpora, identificar *topics*, comparar documentos y análisis de documentos para comparar su estructura semántica.

Se ha utilizado Gensim para comparar el significado semántico de distintos documentos, para así conocer qué documento se parece más a uno dado, en nuestras tareas.

Para esta tarea, Gensim es una de las soluciones más potentes, ya que tiene modelos pre-entrenados con vectores de palabras, por lo que consumiendo su API, es muy sencillo realizar una comparación semántica entre documentos. Existen alternativas para realizar esto, como spaCy o NLTK, pero ninguna ofrece modelos pre-entrenados tan completos como Gensim.

²²<https://pyphen.org/>

²³<https://radimrehurek.com/gensim>

3.1.2.7. TensorFlow



TensorFlow²⁴ es una librería de código libre para *machine learning*. Puede usarse para multitud de tareas, pero se centra principalmente en el entrenamiento e inferencia de redes neuronales. El ecosistema de TensorFlow abarca librerías como Tensorboard, despliegue y producción de API, y soporte para varios lenguajes de programación.

Permite crear modelos de manera sencilla. TensorFlow expone unas API de alto nivel que permiten construir y entrenar modelos. Es fácil entrenar y desplegar modelos en la nube, en el navegador, etc.

TensorFlow permite probar ideas en poco tiempo, es flexible y escalable. Aparte de redes neuronales, es posible realizar multitud de aplicaciones como aprendizaje por refuerzo, procesamiento de lenguaje natural, etc.

Existen distintas alternativas a TensorFlow como MXNet, Pytorch, Chainer y Caffe. Nos decantamos por TensorFlow, ya que, a parte de su popularidad, junto con Keras, suponen la combinación perfecta para aplicar un modelo BERT sobre nuestra tarea de clasificación.

3.1.2.8. Keras



Keras²⁵ es una librería de código libre que proporciona una interfaz Python para redes neuronales artificiales. Actúa como interfaz para TensorFlow, así como para otras librerías como

²⁴<https://www.tensorflow.org/>

²⁵<https://keras.io/>

CNTK o Theano. Keras está diseñada para reducir la carga cognitiva y permitir a los desarrolladores e investigadores minimizar la cantidad de acciones necesarias para casos de uso comunes.

TensorFlow 1.x tenía un sistema de clases muy complejo para construir modelos y, gracias a la popularidad de Keras, cuando salió TensorFlow 2.0, TF adoptó Keras como su API oficial. Por lo tanto, Keras se puede importar a través de TensorFlow, ya que forma parte de esta librería de forma oficial, aunque también está disponible como una librería separada.

Keras es uno de los *frameworks* más utilizados, ya que permite ejecutar nuevos experimentos de forma sencilla. Al estar construido sobre TensorFlow 2.0, puede escalar de una forma muy eficiente. Al mismo tiempo, los modelos implementados con Keras se pueden exportar y desplegar donde uno desee.

Keras permite de una forma sencilla utilizar y construir modelos simplemente añadiendo capas encima de otras con llamadas muy simples.

Las alternativas a Keras son MXNet, Pytorch y Caffe, pero se ha optado utilizar Keras, ya que su combinación con TensorFlow es muy potente. Asimismo, como el uso final es realizar *fine-tune* en un modelo BERT, ofrece la mejor combinación, ya que TensorFlow dispone de un modelo pre-entrenado de BERT.

3.2. Herramientas de desarrollo

En el presente apartado presentaremos todas las herramientas empleadas para el desarrollo de la aplicación, sus servicios, la implementación de los modelos y el despliegue en los entornos.

3.2.1. Visual Studio Code



Visual Studio Code²⁶ es un editor de código fuente gratuito creado por Windows. Incluye soporte para depuración, resaltado de sintaxis, autocompletado de código, refactorización de código y Git integrado.

El objetivo de Visual Studio Code es proporcionar las herramientas que un desarrollador necesita para el ciclo de codificación, compilación y depuración. Para flujos de trabajo más complejos, se debería utilizar un IDE.

Se ha utilizado Visual Studio Code para el desarrollo de la aplicación web y el microservicio de perfilado. No era necesario un IDE, ya que la potencia de Visual Studio Code es lo suficiente para este trabajo.

Existen otros editores como Atom, Geany o VSCodium, pero ninguno tiene la potencia, simplicidad y soporte que tiene Visual Studio Code, por lo que se optó por esta alternativa sin vacilar.

3.2.2. Jupyter



Jupyter²⁷ es una aplicación web de código abierto que permite crear y compartir documentos que contienen código, ecuaciones, visualizaciones y texto narrativo. Los usos principales de estos cuadernos son: limpieza y transformación de datos, simulación numérica, modelado estadístico, visualización de datos y aprendizaje automático.

Para utilizar Jupyter simplemente hay que iniciar el servidor de forma local y acceder por medio del navegador a este. A partir de ahí, es posible crear cuadernos en los que trabajar. Se han creado distintos cuadernos para trabajar con los distintos clasificadores y modelos.

Una alternativa a Jupyter es Google Colab. Una plataforma web con las mismas funcionalidades que Jupyter, pero realiza las ejecuciones en sus servidores. Se decidió utilizar Jupyter para la construcción del modelo de *gender*, ya que el equipo utilizado es lo suficientemente potente para estas ejecuciones.

²⁶<https://code.visualstudio.com/>

²⁷<https://jupyter.org/>

3.2.3. Google Colaboraty Pro

Google Colaboraty (versión PRO) es una plataforma que permite escribir y ejecutar código python a través del navegador. Está enfocado para *machine learning*, análisis de datos y educación. La diferencia con Jupyter es que el código que ejecutas, corre en los servidores de Google, sin utilizar recursos de tu equipo.

Para el entrenamiento del modelo de depresión se ha utilizado Google Colaboraty Pro, ya que se necesitaban más recursos que los que nuestro equipo tiene. Esto fue necesario para que el entrenamiento de estos modelos llevara menos tiempo y así responder más rápido a los resultados de estos.

3.2.4. PlantUML

PlantUML²⁸ es una herramienta para crear diagramas de secuencia UML. Su propósito es mejorar la eficiencia a la hora de crear y trabajar con diagramas de secuencia mediante la combinación de *scripts* de notación de texto y dibujo. PlantUML es una herramienta rápida y muy conveniente para trabajar.

Se decidió utilizar esta herramienta para los diagramas de secuencia, de actores de más, ya que la comodidad que supone crear un diagrama por medio de texto supone en un gran ahorro de tiempo para el mismo resultado que si se hubiera utilizado otra herramienta.

Se han considerado las alternativas de MagicDraw y Dia, pero finalmente se optó por PlantUML por los siguientes motivos:

²⁸<https://plantuml.com/>

- Código abierto, desarrollado en un repositorio GitHub.
- Facilidad de uso.
- Posibilidad de generar diagramas a partir de texto plano.
- Portabilidad. Además, al ser una aplicación web resulta muy cómoda y utilizable en todo tipo de dispositivos.

3.2.5. Pencil



Pencil²⁹ es una herramienta de escritorio de diseño de *mockups* de código abierto. Se ha utilizado para realizar el diseño de las maquetas de la aplicación, en las cuales se basó la programación de las interfaces reales de la aplicación.

Se ha considerado la alternativa de Moqups, pero se descartó rápidamente, ya que no era una herramienta de código abierto, y se persigue un desarrollo del proyecto con herramientas libres.

3.2.6. GitLab



GitLab³⁰ es una plataforma de almacenamiento remoto que permite la creación de repositorios fácilmente *clonables* y manejables desde cualquier equipo. Permite controlar todos los cambios realizados sobre el código fuente del proyecto. Es de código abierto, desarrollado por GitLab Inc. Esta plataforma utiliza el sistema de control de versiones Git.

²⁹<https://pencil.evolus.vn/>

³⁰<https://about.gitlab.com/>

Algunas alternativas a GitLab, para llevar el control de versiones, son GitHub, BitBucket o Gitorious. Se optó por GitLab por su facilidad de uso, su gran integración con herramientas externas y su documentación y comunidad abundantes.

El control de versiones del proyecto se llevó a cabo mediante un repositorio público en GitLab.

Además, se implementó un ciclo de integración y entrega continua (CI/CD). Esta funcionalidad de GitLab permite, a través del fichero *.gitlab-ci.yaml* definir distintos trabajos que se lanzarán si se producen cambios en las ramas asociadas.

El ciclo implementado se detallará en el siguiente capítulo.

3.2.7. Docker Hub



Docker Hub³¹ es un servicio proporcionado por Docker para buscar y compartir imágenes de contenedores. Permite crear automáticamente imágenes desde el ciclo de integración continua y enviarlos a Docker Hub y también desencadenar acciones después de que se hayan subido cambios al repositorio de forma exitosa para integrar Docker Hub con otros servicios.

Se ha utilizado Docker Hub para subir las imágenes de nuestros servicios y así desplegarlas en los entornos adecuados. No existe otra alternativa que realice estas funciones para contenedores Docker.

3.2.8. AWS



³¹<https://hub.docker.com/>

AWS (Amazon Web Services)³² es una herramienta que ofrece API y plataformas de computación en la nube bajo demanda, con un sistema *pay-as-you-go*. Es la plataforma *cloud* que lidera en la actualidad. Sus características principales son:

- Mayor funcionalidad. AWS es el proveedor en la nube que más servicios y funciones ofrece. Desde computación, almacenamiento y bases de datos, hasta aprendizaje automático e inteligencia artificial.
- Comunidad de clientes y socios. AWS tiene la comunidad más grande y dinámica, con millones de clientes activos.
- Seguridad. AWS está diseñado para ser el entorno de computación en la nube más flexible y seguro que existe en la actualidad.
- Innovador. Con AWS, se pueden utilizar las últimas tecnologías para experimentar e innovar con mayor rapidez.
- Experiencia operativa. AWS tiene una experiencia, madurez, confiabilidad, seguridad y rendimiento incomparable. Destaca sobre el resto de proveedores.

Se ha hecho uso de AWS para:

1. Ejecutar fases costosas de los modelos de clasificación, ya que AWS ofrece crear instancias donde se puede disponer de más recursos que el equipo personal disponible.
2. Desplegar el microservicio de perfilado. Se creó un contendor ECS (Elastic Container Service) con un balanceador de carga para tener disponible este servicio allí. El motivo de esta selección es porque se necesitaba más memoria para ejecutar el servicio que, por ejemplo, la del entorno donde se desplegó la aplicación web, y se prevé que será necesario el balanceador de carga, ya que es un servicio que va a tener trabajo por picos. Cuando se carguen datos va a recibir en un momento muchas peticiones que debe ser capaz de procesar y cuando este procesado finalice va a estar sin trabajo hasta que otro administrador decida solicitar más datos de Reddit. De esta forma, estaremos disponiendo siempre de los recursos necesarios para las necesidades del usuario.

Existen alternativas a AWS como Microsoft Azure, Joyent o Google Cloud Compute. Se decidió utilizar AWS por todas las ventajas competitivas mencionadas anteriormente y porque nos ofrece al mismo tiempo instancias donde ejecutar experimentos durante el tiempo deseado, con la flexibilidad de lanzarlas y pararlas al gusto, a la par que una buena infraestructura confiable

³²<https://aws.amazon.com/>

en la que tener nuestro servicio más importante desplegado.

3.2.9. Heroku



Heroku³³ es una de las primeras plataformas *cloud* como servicio (PaaS) que admite varios lenguajes de programación como Ruby, Go, Python, Java, Node.js, PHP o Scala. Heroku está basada en contenedores y permite a desarrolladores implementar, administrar y escalar aplicaciones modernas. Es elegante, flexible y fácil de usar, y ofrece un camino simple para que los desarrolladores puedan desplegar sus aplicaciones a producción.

Heroku tiene ofertado un plan gratuito que permite aprender y a iniciarse en esta plataforma. Se ha hecho uso de este plan para desplegar nuestra aplicación web, ya que cumple los requisitos. El plan gratuito de Heroku ofrece 1000 horas de uso al mes con 1 dyno por aplicación (un dyno es un contenedor ligero de Linux que constituye el corazón de Heroku y donde se ejecutan las aplicaciones) con 512 MB de RAM, base de datos Postgres gratuita hasta 10k filas de datos y máximo 20 conexiones concurrentes, y Redis con 25 MB de RAM y 20 conexiones concurrentes.

La desventaja principal es que al ser gratuito, Heroku “duerme” la aplicación tras 30 minutos de inactividad, provocando que la siguiente petición al servicio, tarde más tiempo en procesarse, porque tiene que despertar a la aplicación.

Se decidió utilizar Heroku y no AWS (como en el caso del microservicio), ya que al ser un trabajo fin de máster, se consideró que el plan gratuito de Heroku ofrece un entorno suficiente para la aplicación y así no se suman gastos al proyecto. En un escenario final de producción con usuarios a diario sí sería necesario mover la aplicación a AWS o contratar un plan de Heroku, donde no se duerma nunca la aplicación y permita más conexiones concurrentes.

Nuestro microservicio del perfilador no era adecuado para este plan gratuito, ya que excedía la memoria, y es por ello que se ha desplegado en AWS bajo un coste por uso y mantenimiento.

³³<https://www.heroku.com/home>

3.2.10. Taiga



Taiga³⁴ es una herramienta en línea para la gestión de proyectos ágiles Scrum. Esta herramienta es muy intuitiva y, al ser una aplicación web, cómoda y ligera.

Taiga permite organizar las tareas e historias de usuario de un proyecto, así como los *sprints* de Scrum. Además, genera gráficas de acuerdo al trabajo realizado y al trabajo pendiente y resulta muy visual.

Las alternativas más populares son MS Project, Trello y Redmine, pero se optó por Taiga por su gran facilidad de uso y comodidad.

3.2.11. SonarCloud



SonarCloud³⁵ es una plataforma para evaluar código fuente. Es software libre y usa diversas herramientas de análisis estático de código fuente como *Checkstyle*, *PMD* o *FindBugs* para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa.

Se ha configurado el ciclo de integración y entrega continua para que cada vez que se suba un cambio a cualquier rama, SonarCloud analice el proyecto. De esta forma, podemos asegurar que nuestro proyecto cumple los estándares de calidad establecidos y que no se añadan cambios a las ramas principales (*develop* y *master*) sin analizar.

Una herramienta con un potencial similar a SonarCloud es SonarQube. Básicamente ambas tienen las mismas funciones y realizan las mismas comprobaciones para asegurar la calidad del

³⁴<https://www.taiga.io/>

³⁵<https://sonarcloud.io/>

código del proyecto. Se decidió utilizar SonarCloud, ya que dispone de una versión web en la que se puede analizar directamente el proyecto subido a un repositorio GitLab si el proyecto es público, sin necesidad de tener un servicio propio desplegado (SonarQube lo requiere).

3.2.12. LaTeX y Overleaf



Overleaf³⁶ es una herramienta de software libre para la composición de documentos de texto de alta calidad. Por sus características y posibilidades es muy usado en la composición de artículos académicos, tesis y libros técnicos.

La mayor desventaja de LaTeX es su curva de aprendizaje, pero una vez superada facilita mucho la labor de escribir documentos más complejos y extensos, como puede verse en la figura 3.1. Además, la calidad tipográfica resultante en documentos LaTeX es mucho mayor.

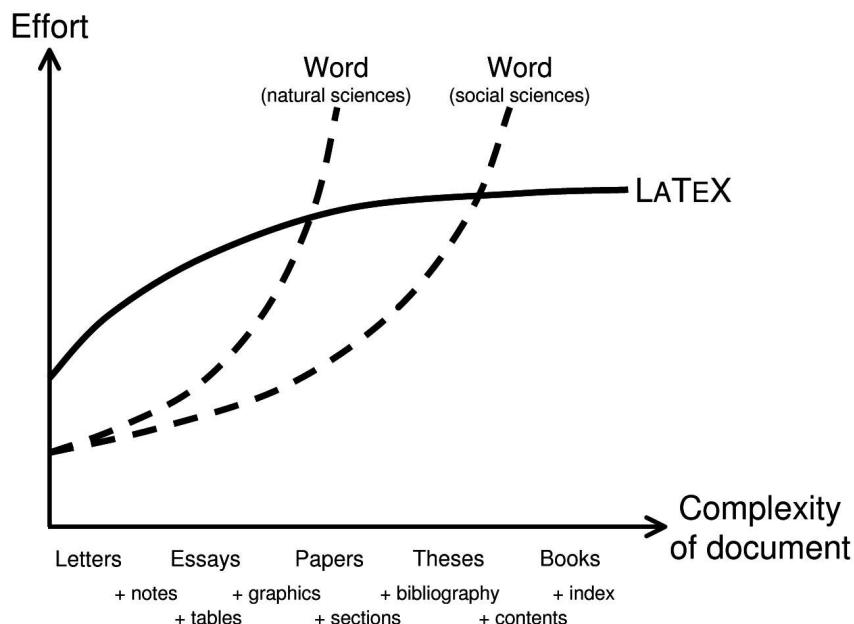


Figura 3.1.: Word frente a LaTeX. Complejidad del documento ante el esfuerzo que conlleva.

Fuente: stackexchange.com

³⁶<https://www.overleaf.com/about>

Overleaf es un editor online de LaTeX que, a su vez, hace la función de repositorio. Es de código abierto y posee funcionalidades adicionales como el control de versiones y la posibilidad de compartir proyectos LaTeX con otros miembros del equipo, que pueden colaborar en tiempo real.

La memoria del proyecto se ha redactado utilizando LaTeX a través de Overleaf.

3.3. Infraestructura

Los recursos hardware y de servicios utilizados para el desarrollo de este proyecto son:

- MacBook Pro Intel Core 2,3 GHz 8-Core Intel Core i9 32 GB 2667 MHz DDR4 RAM 1 TB SSD.
- Instancias AWS para ejecutar los modelos. Se lanzó cuando fue preciso una instancia EC2 a1.4xlarge con 16 vCPU y 32 GB de RAM.
- EC2 para desplegar la aplicación del perfilado. Imagen docker de nuestro servicio, 2 GB de RAM, 0.25 vCPU, balanceador de carga.
- Dos contenedores Heroku para desplegar la aplicación web. Uno para el entorno de *staging* y otro *production*. Ambos tienen las características del servicio gratuito: 1 dyno por aplicación, 512 MB de RAM, 1 CPU, 1000 horas al mes de uso.
- Acceso a internet de banda ancha.

Capítulo 4

METODOLOGÍA

En este capítulo se introduce el método de trabajo y metodología de desarrollo seguidos durante el desarrollo del proyecto, que es una adaptación de una de las más populares metodologías: Scrum. Se mencionarán también las distintas metodologías y buenas prácticas llevadas a cabo como BDD (*Behaviour-driven development*) y *API first development*.

4.1. Scrum

Scrum¹ es una metodología de desarrollo ágil e incremental, que define un flujo de trabajo que divide el proyecto en múltiples *sprints* (ciclos de desarrollo cortos) y un conjunto de entregables o roles con unas responsabilidades específicas [27]. Se caracteriza por aceptar que no se puede comprender o definir completamente el problema en cuestión (ya que se permite a los clientes cambiar de opinión sobre sus deseos y necesidades durante el desarrollo del proyecto). Se centra en maximizar la habilidad del equipo de adoptar nuevos cambios y responder rápidamente a estos.

En este proyecto, como el equipo de desarrollo está formado por una persona, Scrum permite gestionar las tareas entregables del proyecto. Las ventajas descubiertas durante el desarrollo del proyecto, gracias a aplicar Scrum, son:

- La aplicación de las técnicas de Scrum permiten un nivel de concentración mayor y ser más productivo durante el desarrollo del proyecto. La planificación del *sprint* elimina la preocupación de establecer cuál es la siguiente tarea que tienes que hacer ya que, en la

¹<https://www.scrum.org/>

fase de planificación, se dejan claras las historias de usuario a realizar.

- A medida que avanza el desarrollo, se verán que nuevas funcionalidades se pueden ir incorporando al Product Backlog, pero ceñirse al Sprint Backlog siempre dejará claro qué tareas deben realizarse en ese momento.
- Las historias de usuario se deben dividir en tareas más pequeñas que puedan ser completadas durante el *sprint* para producir un producto potencialmente entregable al finalizar este, es decir, las tareas deben finalizarse antes de que comience el siguiente *sprint*.
- Gracias al conocimiento que se va adquiriendo durante todo el proceso resulta más sencillo estimar la importancia y duración que deben tener los elementos que se incorporan al Sprint Backlog. Esto permite tener una lista, bien definida en el tiempo, de tareas a realizar en ese período de tiempo.
- Los *sprints* tienen una período definido por el equipo de desarrollo (generalmente de una a cuatro semanas, dependiendo del tamaño del proyecto) y llevan a un incremento en el software entregable. El artefacto “Sprint Backlog” consiste en las tareas del Product Backlog seleccionadas para su desarrollo durante la iteración.
- Siguiendo el principio de un enfoque flexible, en el transcurso del proyecto se pueden cambiar algunos objetivos o funcionalidades. En cada iteración se solaparán las distintas fases de desarrollo, lo que facilita la adaptación del proyecto a nuevos requisitos y cambios.
- Scrum no se centra en la calidad de los procesos, sino que intenta conseguir la mayor calidad posible en el resultado. Scrum elimina la documentación excesiva que en las metodologías clásicas son obligatorias, y que harían muy pesada y complicada la realización de un proyecto de este estilo por una única persona. Reducir la burocracia involucrada puede permitir reducir el esfuerzo para llevar a cabo el desarrollo del proyecto, al tiempo que se logra un mayor beneficio.

En la figura 4.1 se puede observar el flujo de trabajo que sigue la metodología Scrum.

SCRUM FRAMEWORK

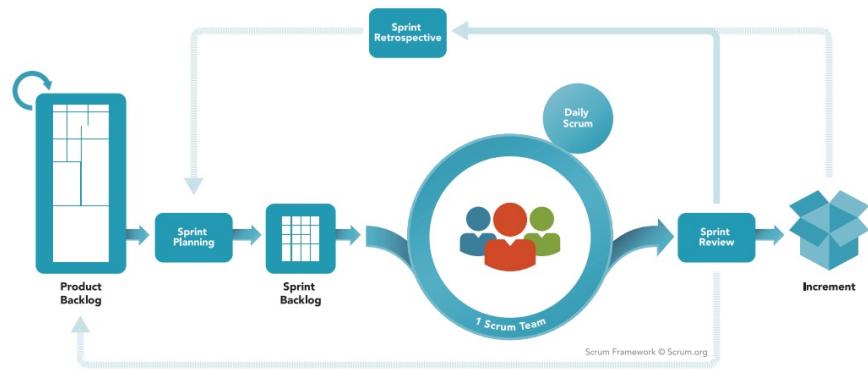


Figura 4.1.: Metodología Scrum. Fuente: scrum.org

4.1.1. Roles

Scrum define una serie de roles especializados, entre los que se pueden destacar los siguientes:

- **Product Owner:** Representa a las partes interesadas o *stakeholders* del producto y sus necesidades y es la voz del cliente. Es su responsabilidad asegurarse de que el equipo aporta valor al negocio. Crea las historias de usuario, las prioriza por su importancia y sus dependencias y las incorpora al *Product Backlog*. Uno de los aspectos más importantes del Product Owner es la comunicación. Debe acordar las prioridades y empatizar tanto con los miembros del equipo como con los *stakeholders*.
- **Development Team:** Consiste en un equipo autoorganizado y autogestionado de 3 a 9 personas, idealmente. Es responsable de contribuir con incrementos potencialmente entregables del producto al final de cada *sprint*. Su principal característica es que no tiene un rol específico. Como grupo disponen de todas las competencias necesarias para desarrollar del proyecto. Por lo tanto, deben ser capaces de poner en práctica todos los incrementos previstos.
- **Scrum Master:** Se asegura de que se sigan todos los procesos de la metodología acordados, evita las distracciones por parte del equipo y ayuda a que este mejore. El Scrum Master debe esforzarse para lograr aumentar la productividad del equipo y alcanzar los objetivos definidos.

Aunque no son parte del proceso de Scrum, también existen otros roles secundarios que son ne-

cesarios, ya que están vinculados a la revisión y retroalimentación del trabajo realizado durante el desarrollo del proyecto.

- **Usuarios:** Son los destinatarios finales del producto, las personas que lo van a usar.
- **Stakeholders:** Consisten en el conjunto de personas que tienen algún interés en la realización del proyecto. Participan en las revisiones de los *sprints*.
- **Managers:** Son los encargados de tomar las decisiones finales participando en la selección de los objetivos y de los requisitos.

4.1.2. Eventos de Scrum

Los eventos de Scrum se caracterizan por ser bloques de tiempo predefinidos (time-boxing). Es decir, tienen una duración máxima. El evento principal en Scrum es el *sprint*, que contiene del resto de eventos. Cada evento establece formas para la inspección y adaptación de aspectos específicos del proyecto.

4.1.2.1. Sprint

Un *sprint* es un bloque de tiempo de 1 semana a un mes en el que se crea un incremento del producto que tendrá que ser utilizable y potencialmente ejecutable. La duración de los *sprints* debe ser constante durante todo el desarrollo, por lo tanto es necesario estimarlo correctamente. Son cíclicos ya que justo al finalizar cada uno comienza el siguiente.

Cada *sprint* puede considerarse como un pequeño proyecto dentro de uno más grande que guía el trabajo y la creación del producto resultante. El objetivo del *sprint* no se puede modificar una vez que ha comenzado.

Se llevan a cabo varios tipos de reuniones durante todo el ciclo de trabajo: hay una reunión diaria para comprobar el estado del proyecto (*Daily Scrum*) y una reunión al principio del *sprint* (*Planning*) y otra al finalizarlo (*Review and retrospective*). De esta forma, se realiza así un seguimiento continuo del proyecto y se facilita la adaptación del mismo a posibles nuevos requisitos.

- **Planning:** Esta es una reunión previa al *sprint* en la que se seleccionan las historias de usuario del Product Backlog que se van a realizar durante el *sprint*. Las historias están clasificadas por orden de prioridad. Es importante que se estime la complejidad

y el esfuerzo de cada historia para poder organizar el *sprint* correctamente. Cuando las historias ya están seleccionadas y el *sprint* listo, los miembros del equipo las dividen en partes más manejables, llamadas tareas y descritas en un lenguaje de dominio de trabajo.

- **Daily Scrum:** Todos los días, el equipo se reúne para compartir lo que han estado haciendo y lo que tienen planeado hacer. Si algún miembro del equipo tiene algún problema, el resto del equipo deberá preguntarse cómo puede ayudarlo y ver cómo solucionarlo. En la reunión se reúne todo el equipo y el Scrum Master, a veces puede aparecer también la figura del Product Owner. Para mantener la brevedad que la caracteriza se hará generalmente de pie.
- **Review:** Esta es la primera reunión que se realiza al final del *sprint*. En ella se presenta el trabajo realizado durante el *sprint* para su revisión. El Product Owner y los *stackholders* son quienes finalmente deciden si los objetivos finales del *sprint* fueron completados. Los elementos sin terminar se deben poner de nuevo en el Product Backlog y de hacer falta, se calcularía de nuevo su prioridad.
- **Retrospective:** Esta es la última reunión del *sprint* y la más distintiva de Scrum. Esta reunión tiene lugar antes de la planificación del próximo *sprint* y tiene una duración de entre una y tres horas, dependiendo de la duración de cada *sprint*. Se centra, no en el contenido, si no en las formas en las que se está realizando el proyecto y el marco de trabajo, para así conocer las fortalezas y las debilidades del equipo. A veces se confunde con una revisión de *sprint*, ya que también se realiza al final de cada *sprint*, pero es importante mantenerlas de forma separada, ya que sirven para diferentes propósitos.

4.1.2.2. Artefactos

Los artefactos en Scrum son aquellos elementos que representan trabajo o valor para proporcionar oportunidades de inspección y adaptación. Deben diseñarse para maximizar el entendimiento de la información clave. Esto significa que deben utilizar un lenguaje de negocio para que así cualquier persona tenga un mismo concepto del artefacto.

- **Product Backlog:** El Product Backlog o la lista de producto es una lista priorizada formada por las tareas y las necesidades de desarrollo del producto. Esta lista enumera todas las características, funcionalidades, requisitos, mejoras y correcciones que deben ser empleadas sobre el producto en futuras versiones o entregas.
- **Sprint Backlog:** El Sprint Backlog o la lista pendiente del *sprint* es un subconjunto

del Product Backlog formado por los elementos de esta última seleccionados para su desarrollo en el *sprint*, más un plan para la entrega del Incremento. El objetivo del *sprint* se consigue cuando se completa el Sprint Backlog. Este hace visible todo el trabajo que el Equipo de Desarrollo identifica como imprescindible para lograr el objetivo del *sprint*. A medida que se avanza en el *sprint* y el trabajo se ejecuta, se va actualizando la estimación de trabajo que queda.

- **Incremento:** El Incremento es la suma de todos los elementos del Product Backlog completados durante un *sprint*. El Incremento siempre debe estar en condiciones de uso sin importar si el Product Owner decide liberarlo o no.

4.1.2.3. Entidades

Tal y como se mencionó anteriormente, el elemento básico de Scrum es el *sprint*. Se compone del Sprint Backlog, el cual se divide en elementos más manejables llamados User Stories o Historias de Usuario [28]. Estas divisiones son las tareas que se deben completar durante el *sprint*, pero a veces pueden ser demasiado grandes para que las lleve a cabo un único miembro del equipo o son demasiado pequeñas para que sean tratadas individualmente. Luego, hay varios tipos de entidades que se describen a continuación.

- **User Stories o Historias de Usuario:** Son los requisitos de trabajo expresados en lenguaje de negocio. El Product Owner es el encargado de su redacción y suponen la unidad básica de un *sprint*, caracterizándose por ser:
 - Independientes
 - Negociables
 - Valoradas
 - Estimables
 - Pequeñas
 - Verificables
- **Tarea:** Es un cometido individual que requiere generalmente entre medio y tres días de trabajo por un integrante del equipo. Las tareas se expresan en lenguaje de dominio técnico, lo que es el lenguaje propio del equipo de desarrollo.

- **Épicas:** Son agrupaciones de historias de usuario que definen los grandes bloques operativos dentro de un proyecto. Cuando las Épicas no son suficientes y se necesitan agrupaciones más grandes se usan los llamados Temas, los cuales detallan las grandes ideas o requisitos del cliente.

4.2. Adaptación

La metodología de desarrollo adoptada para este proyecto ha sido una adaptación de Scrum, usando las reuniones como punto clave para obtener el *feedback*, realizando ciclos cortos de desarrollo (*sprints* de tres semanas en este proyecto), apostando por la simplicidad y definiendo distintos roles.

El Sprint Backlog se definió de una manera más flexible a la que define Scrum, permitiendo así modificarlo a lo largo de la iteración y proporcionando así más libertad.

4.2.1. Personas

Se han mantenido los roles y responsabilidades de la metodología Scrum. En este caso y dadas las circunstancias, el *development team* estará compuesto por tan solo una persona.

- **Equipo de desarrollo:** Paloma Piot Pérez-Abadín.
- **Scrum Master:** Paloma Piot Pérez-Abadín.
- **Product Owner:** Patricia Martín-Rodilla y Javier Parapar.

4.2.2. Reuniones

- **Review, Retrospective and Planning Meeting:** Se realizó una única reunión de aproximadamente treinta minutos al final de cada *sprint*, para analizar el trabajo realizado en el *sprint* anterior y planificar, de acuerdo a esto, el siguiente. En esta reunión participó el equipo de desarrollo, los product owners y la scrum master.
- **Daily Scrum:** Se realizaron de forma diaria quince minutos de auto-análisis por parte del equipo de desarrollo, para analizar lo realizado el día anterior y planear la siguiente jornada de trabajo.

Además de estas reuniones, la comunicación siempre fue abierta y las personas involucrados en el proyecto nos mantuvimos disponibles en gran medida, hablando a menudo.

4.3. Control de versiones

El flujo de trabajo y el control de la evolución del código se llevó a cabo mediante *GitFlow* [29].

El trabajo se organiza en dos ramas principales:

- *master*: cualquier *commit* que esté en esta rama debe estar preparado para subir a producción.
- *develop*: rama en la que está el código que conformará la siguiente versión del proyecto.

Cada vez que se incorpora código a *master*, tenemos una nueva versión.

Además de estas dos ramas, también se han utilizado:

- *feature*: originadas a partir de la rama *develop* e incorporadas siempre a la rama *develop*. Se utilizan para desarrollar nuevas características de la aplicación que, una vez terminadas, se incorporan a la rama *develop*.
- *release*: se originan a partir de la rama *develop* y se incorporan a *master* y *develop*. Se utilizan para preparar el siguiente código en producción. En estas ramas se hacen los últimos ajustes y se corrigen los últimos *bugs* antes de pasar el código a producción incorporándolo a la rama *master*.
- *hot-fix*: se originan a partir de la rama *develop* y se incorporan a *master* y *develop*. Se utilizan para corregir errores y *bugs* en el código en producción. Funcionan de forma parecida a las *release branches*, siendo la principal diferencia que los *hot-fixes* no se planifican.

4.3.1. CI/CD

Para complementar el flujo de trabajo y para tener una mejor forma de trabajo, se implementó un flujo de integración continua y despliegues continuos (CI/CD) [30]. Para esto, nos apoyamos en herramientas como GitLab, SonarCloud, Heroku y AWS.

GitLab permite, a partir del fichero `.gitlab-ci.yaml` definir distintos trabajos que se lanzarán si se producen cambios en las ramas asociadas.

En nuestro caso, se definió un ciclo con cuatro escenarios, tanto para la aplicación web como el microservicio del perfilador:

- **test**: en esta fase se ejecutarán los tests de cada servicio. Se activará esta fase en cada *commit* que se realice a cualquier rama. Si esta fase no se realiza con éxito, no se continuará con las siguientes.
- **build**: supone la construcción del proyecto. Se construirá la imagen del contenedor de docker y se subirá. Igual que la fase de **test**, no se permite continuar si esta fase falla.
- **sonar**: en este escenario, se lanza el escáner de sonar. Solamente se ejecuta cuando se abre una PR (*pull request*) o cuando se añaden cambios las ramas principales (*develop* o *master*). Se permite seguir ejecutando el ciclo de CI/CD aunque esta fase no se ejecute con éxito.
- **deploy**: existen dos trabajos para esta fase, dependiendo de la rama en la que se ejecute. Este escenario realizará el despliegue del servicio en Heroku (en el caso de la aplicación web) y AWS (en el caso del servicio del perfilador). En el caso de la aplicación web, existen dos entornos (*staging* y *production*). Cuando se añaden cambios a la rama *develop*, se realizará el despliegue automático en el entorno de *staging* y en el caso de *master*, se desplegará a producción. Para el perfilador se ha decidido tener un único punto de acceso.

4.4. ***Behaviour-driven development (BDD)***

Behaviour-driven development (BDD) [31] es un proceso de desarrollo en el cual se definen en un lenguaje natural las descripciones de los requisitos, y estas suponen la base de los *tests*. Para realizar estas descripciones se utiliza el lenguaje Gherkin y, basándonos en estas, se escriben distintas pruebas para asegurar que la funcionalidad de nuestro sistema sea la adecuada.

Esta práctica alinea los pensamientos entre desarrolladores, QA y personas no técnicas del proyecto, ya que las descripciones de las pruebas están escritas en lenguaje natural. De esta forma, tanto *product owners* como partes interesadas pueden escribir estos escenarios, y luego los desarrolladores son los encargados de traducirlos a *tests* (que siguen una misma estructura que las descripciones) y, finalmente, añadir el código necesario para que esos *test* se ejecuten

con éxito.

Cada prueba que se crea tiene el objetivo de aportar valor de negocio al cliente y cómo se debe comportar el sistema para cumplir las expectativas. Estas pruebas se conocen como pruebas de caja negra, ya que, la persona encargada de definir la prueba sabe lo que quiere, pero no detalla en cómo se debe realizar.

Esta es una buena práctica en metodologías ágiles, ya que, te obliga a escribir las pruebas antes de implementar el código (asegurando que siempre vas a tener tu código probado) y aseguras que la funcionalidad que vas a implementar cumpla los requisitos que las partes interesadas definen.

En la sección de pruebas 8.1.3 se explica en mayor detalle el funcionamiento de esta metodología y cómo se llevó a cabo en el proyecto actual.

4.5. API first development

API first development es una práctica en la cual el primer paso en el desarrollo de un proyecto es la creación de las API. Consiste en desarrollar API consistentes y reutilizables, lo cual se puede lograr utilizando un lenguaje de descripción de API que establece un contrato sobre cómo la API se debe comportar.

Los principales beneficios de esta buena práctica son:

- Permitir el trabajo en paralelo entre equipos de desarrollo.
- Reducir el coste en el desarrollo de aplicaciones.
- Aumentar la velocidad de salir al mercado.
- Asegurar una buena experiencia de desarrollo.
- Reducir el riesgo de fallar.

A pesar de que muchas de las ventajas no aplican en este proyecto (equipo de desarrollo de una única persona, etc.), se consideró que es una buena práctica de trabajo que se debía adoptar. De esta forma se procedió a diseñar y documentar los distintos *endpoints* de nuestra aplicación y micro-servicios y, una vez finalizado esto, proceder con la implementación y el desarrollo del código.

Para esto se utilizó OpenAPI y, en la siguiente imagen 4.2, podemos ver el resultado a través de la interfaz que GitLab ofrece para visualizar y manipular interfaces de aplicaciones definidas con OpenAPI, del API de nuestro sistema.

The screenshot shows a GitLab page for the file 'openapi-schema.yml'. The page includes standard GitLab navigation buttons like 'Edit', 'Web IDE', 'Lock', 'Replace', and 'Delete'. Below the file name, it shows the path '/poltti/early/-/raw/develop/app/openapi-schema.yml'. The main content area is titled 'profile' and contains three API endpoints:

- GET /api/profiles/**: A blue button indicating this is the current active endpoint.
- POST /api/profiles/**: A green button.
- GET /api/profiles/{id}/**: A light blue button.

Below the endpoints, there is a description: "Get one profile." and a "Parameters" section. The "Parameters" section has a table with two columns: "Name" and "Description". It lists a single parameter:

Name	Description
id * required string (path)	A unique integer value identifying this profile. Id - A unique integer value identifying this profile.

At the bottom right of the parameters section is a "Try it out" button.

Figura 4.2.: Ejemplo de la documentación del API del sistema

De esta forma, se documentó cada *endpoint* del sistema construido, destacando la importancia de tener bien creada esta documentación en el caso del microservicio del perfilador, ya que constituye un servicio REST el cual, eventualmente, podrá ser consumido por distintas aplicaciones externas.

Capítulo 5

ANÁLISIS

En este capítulo se describe el proyecto a desarrollar y su modelo conceptual. Además, se analizan los requisitos que presenta.

El análisis de requisitos se lleva a cabo en los instantes iniciales de un proyecto y es una de las partes fundamentales del desarrollo software. En esa fase se determinarán las necesidades o condiciones clave que debe cumplir nuestra aplicación. Antes de iniciar la fase de diseño es muy importante que se analice con profundidad el alcance del sistema y sus funcionalidades.

5.1. Descripción general

La aplicación web desarrollada estará compuesta por dos partes fundamentales: un panel para el administrador, donde puede lanzar *queries* contra la API de Reddit y así crear colecciones de datos, combinado con la exportación a posteriori de estas colecciones, y un conjunto de interfaces para facilitar al experto la gestión de perfiles y la corrección y validación de estos.

5.2. Requisitos

La tarea principal del análisis de este proyecto es la recogida de requisitos, mediante la cual se logrará una comprensión general de la aplicación que se va a construir, tanto por parte de los desarrolladores como de los clientes.

5.2.1. Actores

A la hora de recoger los requisitos, se identificaron los siguientes actores que dispondrán de distintas funcionalidades:

- **Usuario no conectado:** Usuario visitante de la web que no ha iniciado sesión.
- **Usuario conectado:** Usuario que ha iniciado sesión en la web de manera exitosa.
- **Experto:** Usuario encargado de la corrección y validación de los datos perfilados.
- **Administrador:** Usuario administrador que puede cargar y exportar datos.

Además, también se identifica el actor **usuario de la aplicación**, que engloba el conjunto de todos los anteriores.

Los diagramas presentados en esta memoria se han definido en inglés, ya que fue el idioma utilizado a lo largo del desarrollo de este proyecto.

En la figura 5.1 se puede ver un diagrama de la herencia de los actores. Así, un administrador posee las funcionalidades de un experto y este posee las funcionalidades de los usuarios conectados, y estos, a su vez, poseen las funcionalidades de los usuarios de la aplicación. Los usuarios no conectados, además de sus propias funcionalidades, poseen las funcionalidades de los usuarios de la aplicación.

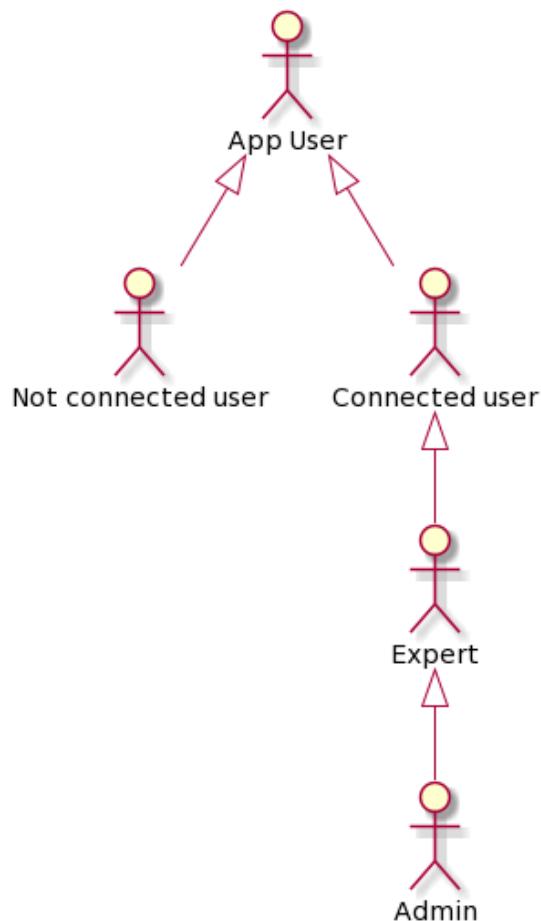


Figura 5.1.: Diagrama de actores

También existe un actor especial, “sistema”, que es aquel que no se corresponde con ningún usuario humano de la aplicación, pero realiza tareas automáticas accionadas a partir de otras funcionalidades.

5.2.2. Historias de usuario

Tras indagar en el dominio de la problemática de la aplicación, se elaboró, mediante historias de usuario, el denominado *Product Backlog* de Scrum. Este Product Backlog se fue modificando a medida que avanzó el desarrollo, ya fuese para añadir historias o para detallarlas.

Las historias de usuario sirven para especificar los requisitos del software, se trata de tarjetas de papel en las cuales se describe brevemente las características que el sistema debe poseer. El

tratamiento de las mismas es muy dinámico y flexible, pero cada historia de usuario debe ser lo suficientemente comprensible y delimitada para que los desarrolladores puedan implementarla dentro de la duración de un *sprint*.

Se detallarán las historias de usuario que los actores descritos pueden realizar, obviando aquellas relacionadas con la gestión de grupos, permisos, etc., ya que se delega en el *framework* utilizado, pero que el usuario administrador puede realizar.

Durante el desarrollo, cada historia de usuario se descompone en varias tareas.

5.2.2.1. Historia de usuario 1: Gestión de usuarios

Como usuario de la aplicación *quiero* gestionar los usuarios *para* saber quién hace qué cambios y garantizar que agentes anónimos no hagan un uso indebido de la aplicación.

Esta historia de usuario provoca la creación de la entidad *User*.

- **GU-1: Iniciar sesión.** *Como* usuario no conectado *quiero* iniciar sesión en la aplicación *para* poder acceder a la aplicación y así disponer de más funcionalidades.
- **GU-2: Cerrar sesión.** *Como* usuario conectado *quiero* desconectarme *para* cerrar mi sesión y evitar que otra persona, en el mismo dispositivo, la use.
- **GU-3: Editar detalles de la cuenta.** *Como* usuario conectado *quiero* editar los detalles de mi cuenta de usuario *para* poder actualizar o corregir los datos.
 - Notas:
 - El correo electrónico es obligatorio y debe corresponderse con un correo electrónico válido (debe contener “@”).
- **GU-4: Cambiar contraseña.** *Como* usuario conectado *quiero* cambiar mi contraseña *para* asegurar más mi cuenta.
 - Notas:
 - La confirmación de la contraseña debe coincidir con la nueva contraseña introducida.
- **GU-5: Recordar contraseña.** *Como* usuario no conectado *quiero* recordar mi contraseña *para* poder acceder a la aplicación con una contraseña nueva, ya que no se recuerda la

anterior.

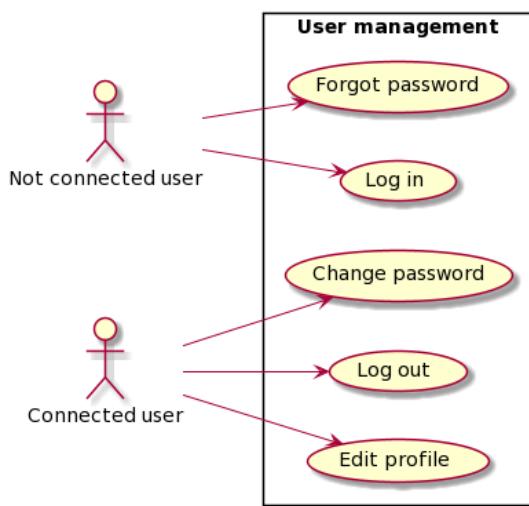


Figura 5.2.: Diagrama de las funcionalidades de la *Gestión de usuarios*

5.2.2.2. Historia de usuario 2: Gestión de perfiles

Como usuario de la aplicación quiero gestionar los perfiles para poder ver y corregir sus datos y tener conocimiento de los datos demográficos de los usuarios de redes sociales.

Esta historia de usuario provoca la creación de las entidades *Profile*, *ProfileData* y *Reason*.

- **GP-1: Ver perfiles.** *Como usuario experto quiero ver todos los perfiles para de un vistazo ver los perfiles que tiene el sistema y sus datos más importantes, sin entrar en el detalle.*
- **GP-2: Buscar perfiles.** *Como usuario experto quiero buscar en los perfiles para encontrar, mediante diversos filtros como nombre del usuario en la red social externa, rango de edad, género, depresión, etc. los perfiles de los que quiero leer datos o corregir.*
- **GP-3: Ver detalle perfil.** *Como usuario experto quiero ver el detalle de un perfil para ver en mayor detalle los datos del perfil y lograr un entendimiento mejor de ese perfil concreto.*
- **GP-4: Validar perfil.** *Como usuario experto quiero editar y validar un perfil para corregir cualquier dato que el sistema de perfilado automático haya extraído erróneamente y así disponer del perfil con datos correctos. Tras esto, se desea dejar el perfil marcado como “validado” para indicar que un experto ya ha hecho una corrección de los datos o*

ha afirmado explícitamente que esos datos eran correctos.

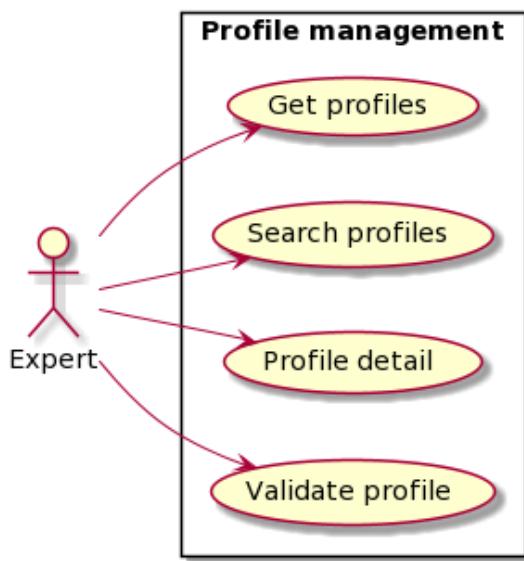


Figura 5.3.: Diagrama de las funcionalidades de la *Gestión de perfiles*

5.2.2.3. Historia de usuario 3: Gestión de datos

Como usuario de la aplicación *quiero* gestionar los datos *para* construir conjuntos de prueba, facilitarlos para su exportación y posteriormente poder realizar el perfilado automático.

Esta historia de usuario provoca la creación de las entidades *Corpus*, *Comment* y *GlobalData*.

- **GD-1: Obtener datos de Reddit.** *Como* usuario administrador *quiero* obtener datos de Reddit *para* disponer de colecciones de datos para manipular posteriormente.
- **GD-2: Exportar datos.** *Como* usuario administrador *quiero* exportar conjuntos de datos *para* tener en mi dispositivo las distintas colecciones de datos y poder usarlas para entrenamientos de modelos.
- **GD-3: Crear corpus.** *Como* usuario administrador *quiero* crear corpus *para* poder clasificar los perfiles en distintas categorías.
- **GD-4: Procesar usuario de Reddit.** *Como* usuario administrador *quiero* procesar un usuario de Reddit *para* poder conocer los datos demográficos de ese perfil en concreto.

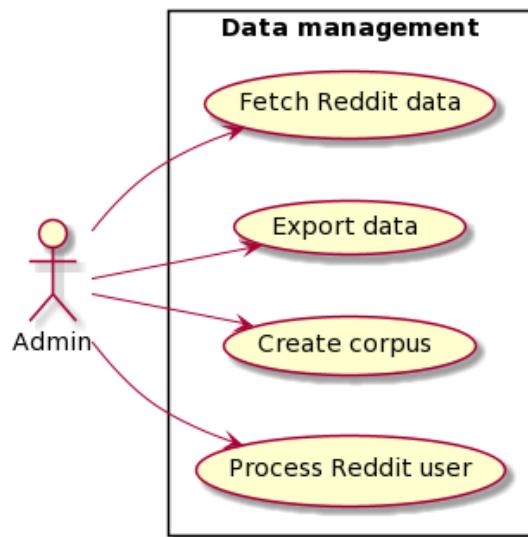


Figura 5.4.: Diagrama de las funcionalidades de la *Gestión de datos*

5.2.2.4. Historia de usuario 4: Perfilado automático

Como usuario “sistema” quiero realizar un perfilado automático para clasificar de forma automática y rápida, sin necesidad de intervención humana, un conjunto de usuarios de redes sociales.

- **PA-1: Perfilar usuario.** *Como usuario “sistema” quiero perfilar a un usuario de redes sociales para conocer sus datos demográficos a partir de una clasificación automática y saber también si es susceptible de sufrir depresión o no.*

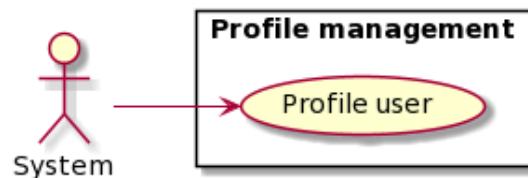


Figura 5.5.: Diagrama de las funcionalidades del *Perfilado automático*

5.3. Diagrama entidad relación

En la figura 5.6 se muestra el diagrama entidad-relación de la aplicación. Se muestran todas las entidades, incluyendo aquellas para la gestión de usuarios, grupos y permisos.

La notación que se usa es la siguiente. Cada entidad está representada en el cuadrilátero con la franja verde en la parte superior. Las relaciones 1-N están representadas por la línea que finaliza en un círculo relleno y en el caso de relaciones N-M, se utiliza la línea con círculos llenos en ambos lados. En el lado en el que se encuentra el círculo relleno se indica la entidad propietaria de la relación. Para indicar herencia de clases se utiliza una flecha sin rellenar.

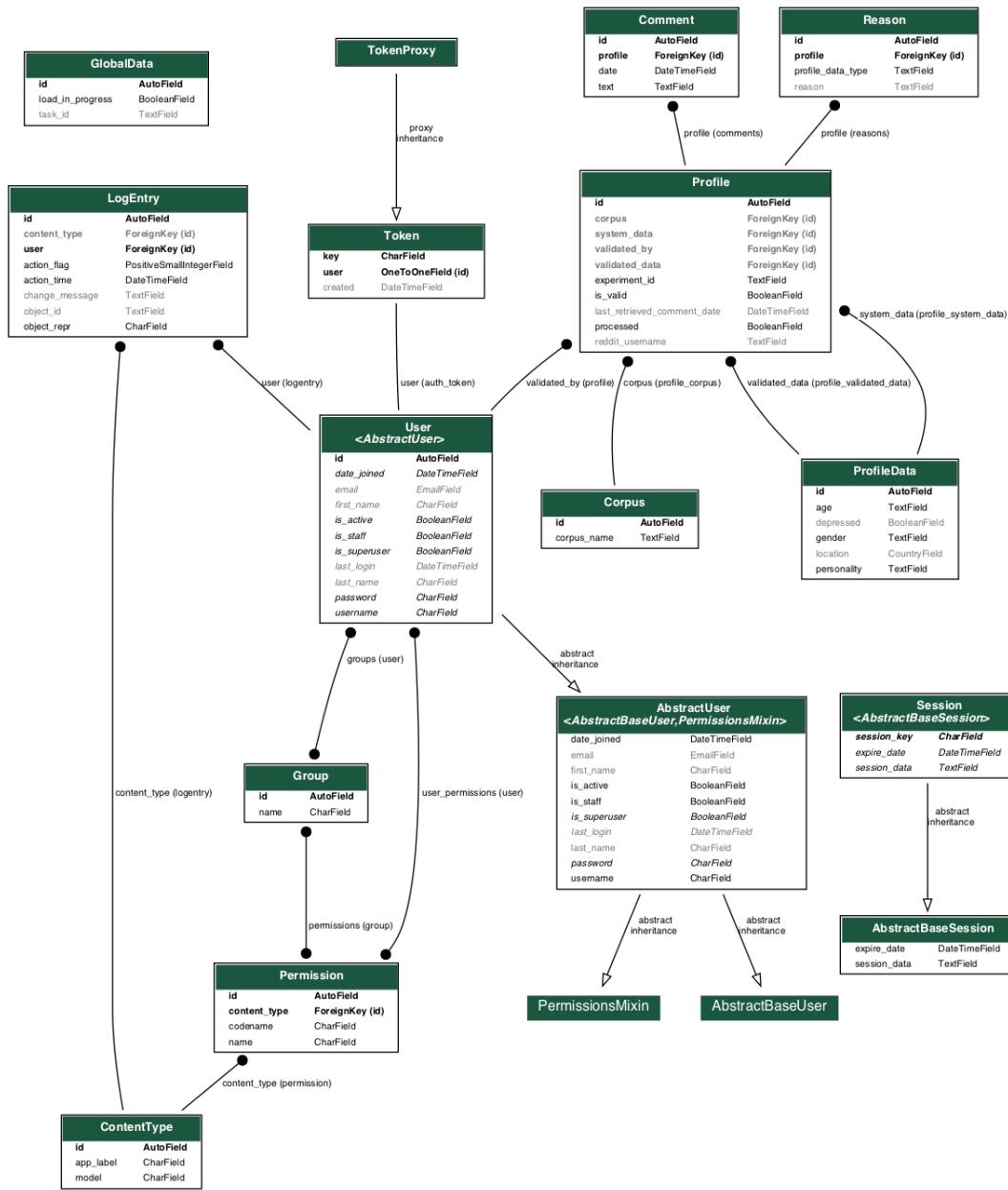


Figura 5.6.: Diagrama entidad-relación completo

Para una mayor apreciación del núcleo del proyecto, se mostrará el diagrama entidad-relación 5.7 eliminando las entidades relativas a la gestión de usuarios. Podemos apreciar que la clase principal es *Profile*, que se complementa con *Reason*, *Comment*, *ProfileData* / *Corpus*, siendo *Profile* la

Las relaciones de los diagramas son 1-1, entre *Profile* y *ProfileData* / *Corpus*, siendo *Profile* la

propietaria, y entre *Profile* y *Reason* / *Comment*, siendo las propietarias estas últimas.

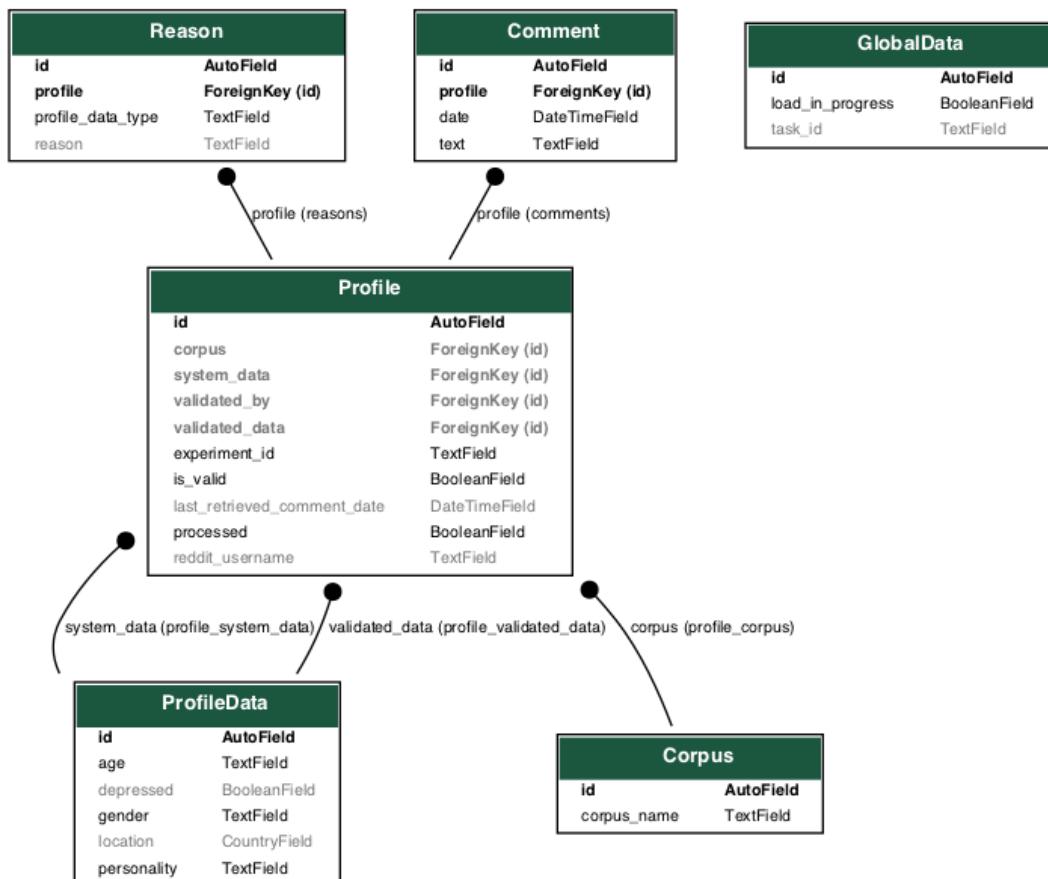


Figura 5.7.: Núcleo del diagrama entidad-relación

5.4. Modelo de datos

A continuación se describe cada entidad y sus atributos y relaciones en detalle. Se ha obviado el atributo *id* (identificador) dado que es trivial para el entendimiento de las entidades. También se ha decidido no documentar las entidades de la gestión de usuarios, ya que es código común en la mayoría de las aplicaciones y no supone el objetivo principal de este trabajo recayendo su gestión en las utilidades del *framework* de desarrollo.

- **User.** Esta entidad representa un usuario en el sistema. Sus propiedades son:

- *date_joined*: fecha y hora en la que su cuenta fue creada. Este campo es obligatorio

y se genera automáticamente cuando se crea la cuenta de un usuario.

- *email*: dirección de correo electrónico asociada al usuario.
 - *first_name*: nombre del usuario.
 - *is_active*: *flag* para indicar si el usuario está activo, y por tanto tiene permisos para realizar funciones en la aplicación, o no. Este campo es obligatorio.
 - *is_staff*: *flag* para indicar si el usuario tiene permisos para acceder a las funcionalidades del panel de administración o no. Este campo es obligatorio.
 - *is_superuser*: *flag* para indicar si el usuario tiene todos los permisos posibles sin asignárselos directamente. Este campo es obligatorio.
 - *last_login*: fecha y hora en la que el usuario hizo su último inicio de sesión.
 - *last_name*: apellido(s) del usuario.
 - *password*: contraseña del usuario con la que se le permite iniciar una sesión. Este campo es obligatorio.
 - *username*: nombre del usuario con el que va a poder iniciar una sesión. Este campo es obligatorio.
- **Profile**. Esta entidad representa un perfil, es decir, un usuario de redes sociales del cual nos interesa obtener su perfil (sus datos demográficos).
- *corpus*: *Corpus* al que pertenece el perfil.
 - *system_data*: *ProfileData* con los datos que ha extraído el sistema de forma automática.
 - *validated_by*: usuario que ha validado los datos de la persona usuaria de redes sociales.
 - *validated_data*: *ProfileData* con los datos corregidos o validados. Puede contener los mismos valores que *system_data* en el caso que el sistema haya realizado un perfilado perfecto.
 - *experiment_id*: *id* para reconocer al perfil. No es el *id* interno, si no uno externo para reconocer al experimento, sin tener que disponer de datos personales de este.

- *is_valid*: flag para indicar si el perfil ha sido validado o todavía está pendiente de revisión por un experto.
 - *last_retrieved_comment_date*: fecha y hora del comentario más reciente de un perfil. Sirve de guía para no persistir comentarios ya existentes.
 - *processed*: flag para indicar si los datos del perfil ya han sido procesados por el perfilador automático o no.
 - *reddit_username*: nombre de usuario del perfil en la red social Reddit, de la cual se cargan y procesan datos.
- **ProfileData**. Abarca los distintos campos del perfilado, tales como edad, género, etc. Y está relacionado con *Profile*. Esta tiene:
- *age*: representado mediante rangos de edad (12-20, 20-30, 30-65 y +65) para indicar el grupo de edad al que pertenece el perfil. El rango de edad es suficiente para conocer el perfil de la persona y su susceptibilidad a sufrir de trastornos psicológicos. Estos rangos de edad hacen referencia a etapas de la vida como adolescencia, adulto joven, adulto y vejez.
 - *depressed*: flag para indicar si el perfil padece de depresión o no.
 - *gender*: *male* o *female* en el caso de usuarios de redes sociales hombres o mujeres, respectivamente.
 - *location*: país de procedencia del perfil.
 - *personality*: personalidad de la persona a perfilar, se clasifica en *extraversion*, *agreeableness*, *conscientiousness*, *neuroticism* y *openness*.
- **Comment**. Representa un comentario de un *Profile* en una red social externa. Un *Profile* puede tener de cero a N comentarios.
- *profile*: perfil que ha realizado el comentario. Este campo es obligatorio.
 - *date*: fecha en la que se realizó el comentario. Este campo es obligatorio.
 - *text*: texto del comentario. Este campo es obligatorio.
- **Reason**. Representa el motivo por el que un campo del perfilado (de *ProfileData*) tiene un valor u otro. Puede ser desde un motivo de corrección por un experto o, ya que el

sistema de perfilado clasificó ese perfil en una categoría u otra y con que confianza.

- *profile*: perfil del cual se ha extraído o indicado el razonamiento.
 - *profile_data_type*: campo del perfilado del que se justifica su valor.
 - *reason*: campo de texto donde se recoge los motivos por los que es campo del perfilado de un usuario de redes sociales tiene un valor y no otro.
- **Corpus.** Esta entidad recoge los distintos corpus del sistema. Se puede entender como “categorías” para agrupar los distintos perfiles.
- *corpus_name*: nombre único y mandatario lo suficientemente explicativo como para reconocer la categoría o tipo de corpus que representa.
- **GlobalData.** Esta entidad engloba los datos compartidos entre todos los usuarios. En este caso, los atributos que se persisten son para indicar si ya hay una carga de Reddit en proceso y el *id* de esta.
- *load_in_progress*: *flag* que indica si hay una tarea en *background* ejecutándose, es decir, si se están cargando datos de Reddit en ese instante.
 - *task_id*: *id* de la tarea que se está ejecutando en *background*.

Capítulo 6

DISEÑO

En este capítulo se comentan las decisiones que se han tomado en cuanto al diseño de la aplicación y su arquitectura.

6.1. Maquetas

Se empezó creando bocetos y maquetas de cómo tendría que ser la interfaz del usuario. El objetivo es que el diseño de las interfaces sea lo más simple posible e intuitivo para que se maximice la usabilidad de la aplicación. Para esto, se intenta seguir las pautas de UI y UX, acrónimos de *User Interface* y *User Experience* respectivamente.

User Experience (UX) es la experiencia que los usuarios tienen con el sistema, que persigue mejorar la satisfacción de estos, mientras que *User Interface* (UI) hace referencia a la apariencia, la presentación y la interactividad del sistema, incluyendo los elementos visuales y patrones de interacción.

Las maquetas se fueron diseñando a medida que se avanzaba a lo largo de los distintos *sprints*.

Se pretende que las interfaces sean sencillas e intuitivas, para que el usuario sea capaz de realizar sus actividades fácilmente. También unificar los estilos para que los cambios de una interfaz a otra sea lo más suave posible.

Por este motivo, los formularios de inicio sesión (figura 6.1), cambio de contraseña (figura 6.2), recuperar contraseña (figura 6.3) y actualización de detalles de la cuenta (figura 6.4) tienen la

misma base y las diferencias son los datos que manejan, para así resultar familiar al usuario y sepa cómo enfrentarse a estas interfaces.

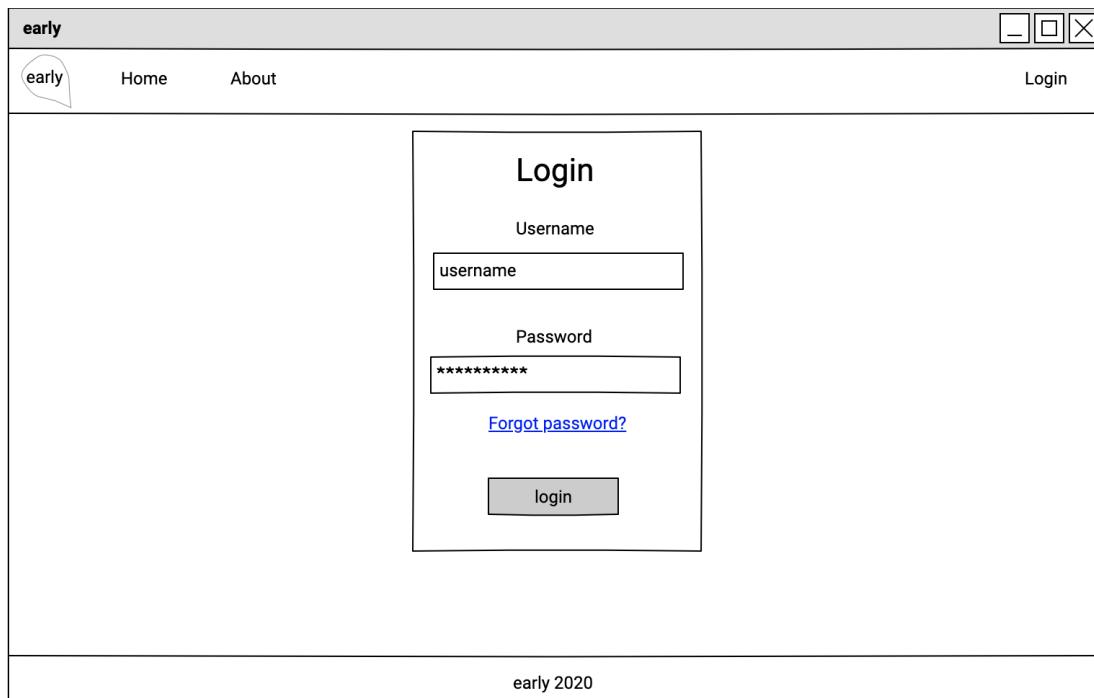
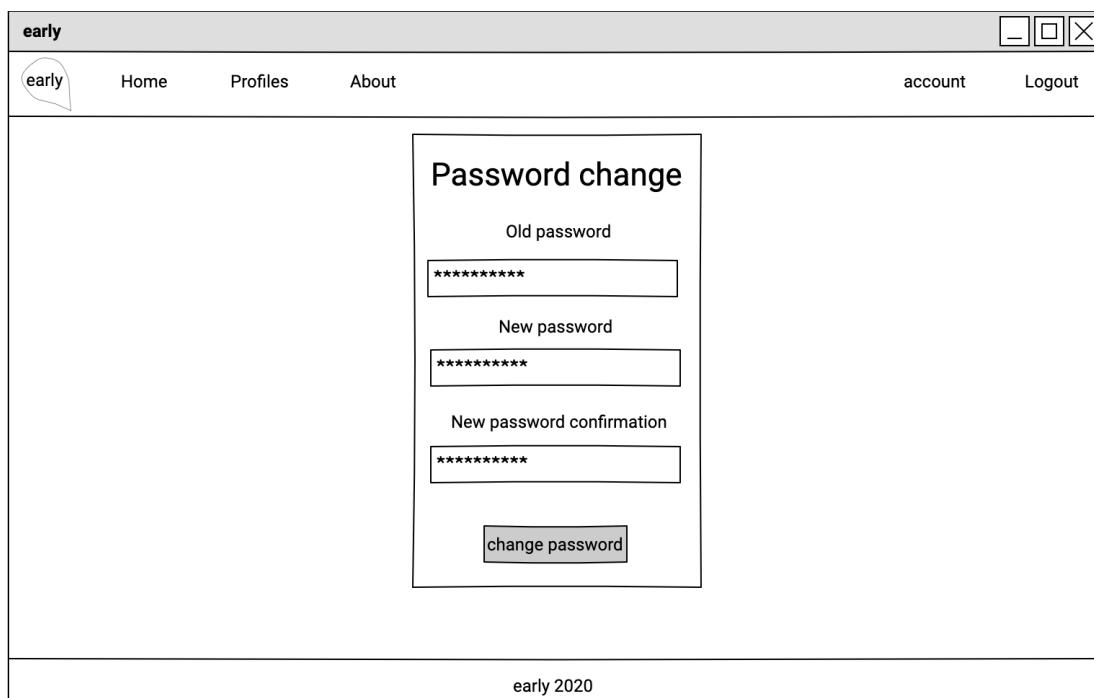


Figura 6.1.: Maqueta de la interfaz de login



early

Home Profiles About account Logout

>Password change

Old password

New password

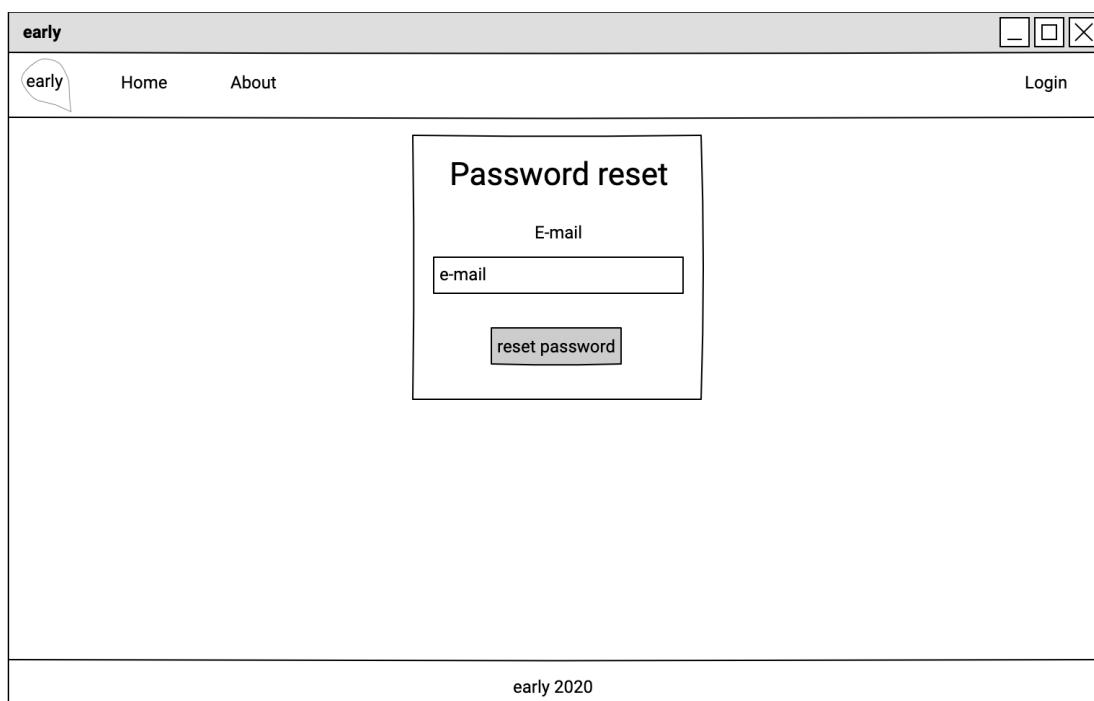
New password confirmation

change password

early 2020

This wireframe shows a 'Password change' form. It includes fields for 'Old password', 'New password', and 'New password confirmation', each represented by a series of asterisks. A 'change password' button is at the bottom. The form is centered on a page with a header containing 'early' and navigation links, and a footer indicating 'early 2020'.

Figura 6.2.: Maqueta de la interfaz de cambio de contraseña



early

Home About Login

Password reset

E-mail
e-mail

reset password

early 2020

This wireframe shows a 'Password reset' form. It has a single field for 'E-mail' (labeled 'e-mail') and a 'reset password' button. The form is centered on a page with a header containing 'early' and navigation links, and a footer indicating 'early 2020'.

Figura 6.3.: Maqueta de la interfaz de recuperación de la contraseña

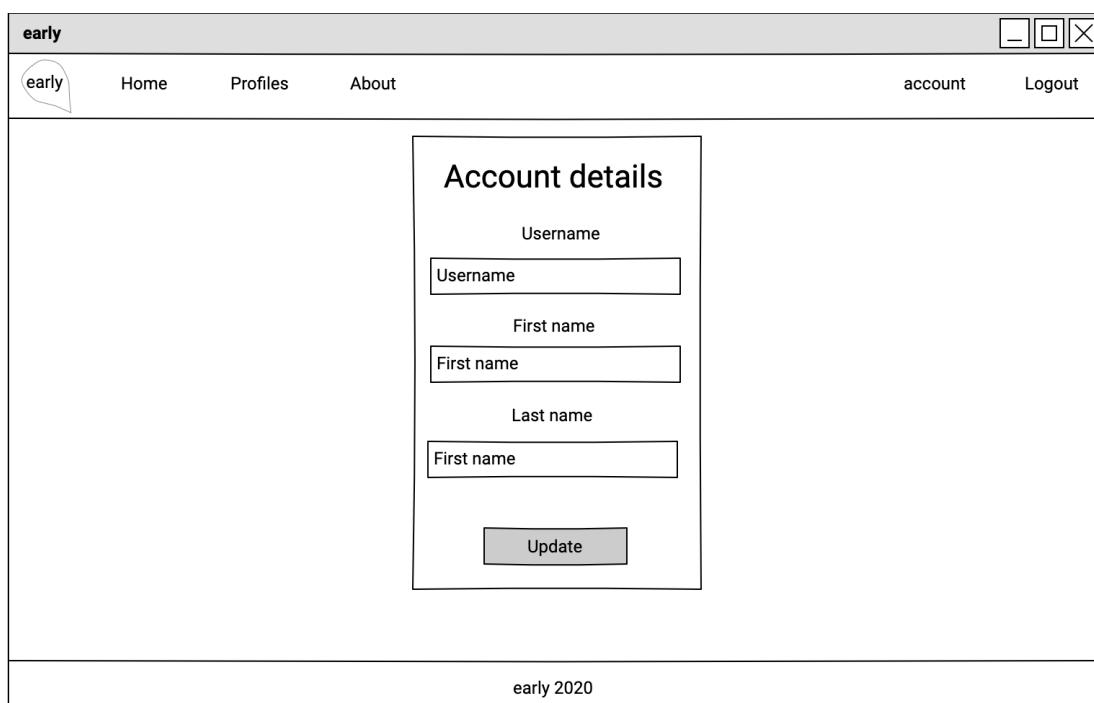


Figura 6.4.: Maqueta de la interfaz de actualización de detalles de la cuenta

Se desea que el usuario experto, de una forma visible y de fácil acceso, pueda localizar los perfiles, junto con sus datos más importantes, con iconos grandes, ajustándose el tamaño y disposición de estos al dispositivo que usa.

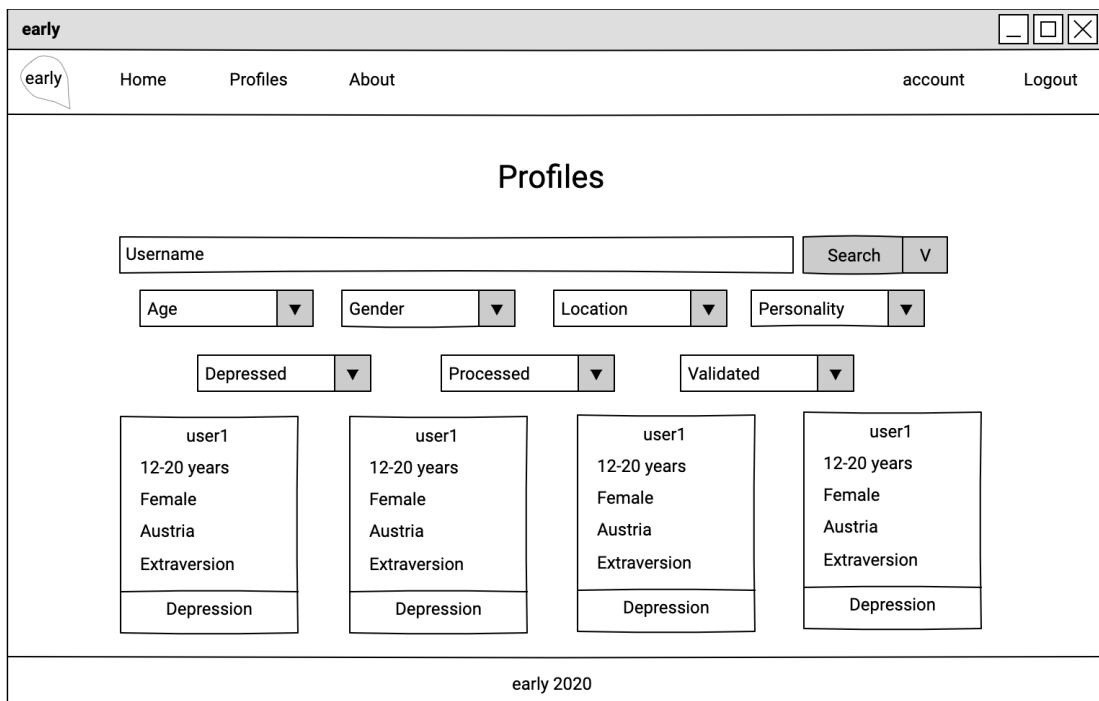


Figura 6.5.: Maqueta de la interfaz de los perfiles

Al mismo tiempo, se pretende que de un vistazo rápido, el usuario experto tenga una ligera idea sobre los perfiles. Para complementar esto, se ha facilitado un buscador para permitir filtrar los perfiles basándose en propiedades de estos y mejorar la experiencia del usuario para que se centre en aquellos perfiles que desea examinar.

Para la navegación al detalle del perfil, se ha hecho que toda la tarjeta con la información sea *clickable* y el usuario al pasar por encima de ella verá como la sombra y los bordes cambian para que destaque sobre el resto y sepa que al hacer *click* realizará una acción sobre ella, en este caso, acceder al detalle del perfil.

Además, se ha seguido la Ley de Miller [32], la cual defiende que de media una persona retiene en su memoria unos siete (más/menos dos) elementos. Por lo que, organizar el contenido en trozos más pequeños ayuda al usuario a procesar, entender y memorizar el contenido más fácilmente. Es por esto que se han agrupado los distintos datos demográficos de forma estructurada, destacando el nombre del campo. Y el motivo del valor de un campo se encuentra en otra zona de la interfaz, para que el usuario lo perciba tras haber procesado los datos más importantes del perfil.

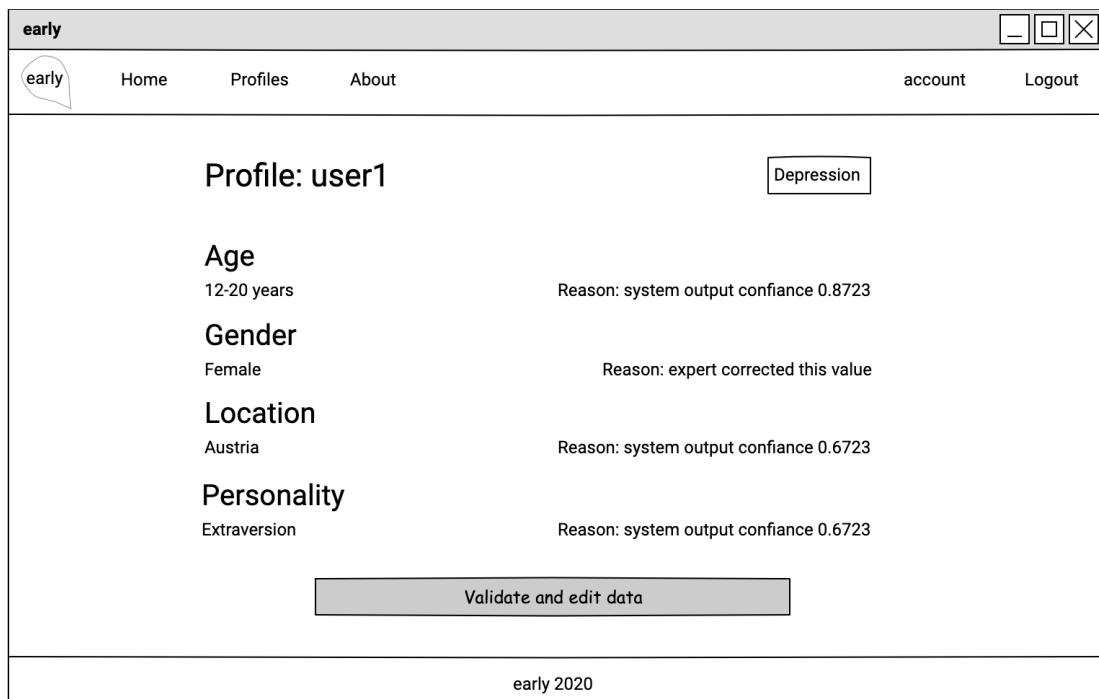


Figura 6.6.: Maqueta de la interfaz de detalle del perfil

Se persigue que los usuarios sean capaces de encontrar de una forma fácil y eficiente la funcionalidad proporcionada. Se ha seguido un diseño centrado en usuarios, por lo que se mostrará en el panel del administrador las acciones que se le permite realizar, siguiendo un diseño que le permite navegar sin esfuerzo y con iconos que representan cada acción.

Al mismo tiempo, este diseño ha seguido las pautas de la Ley de la Región Común [33]. Esta defiende que los elementos se suelen percibir en grupos si comparten un área con unos límites bien definidos. Por lo tanto, añadiendo bordes rodeando un elemento o grupos de elementos ayuda al usuario a percibir esa región común.

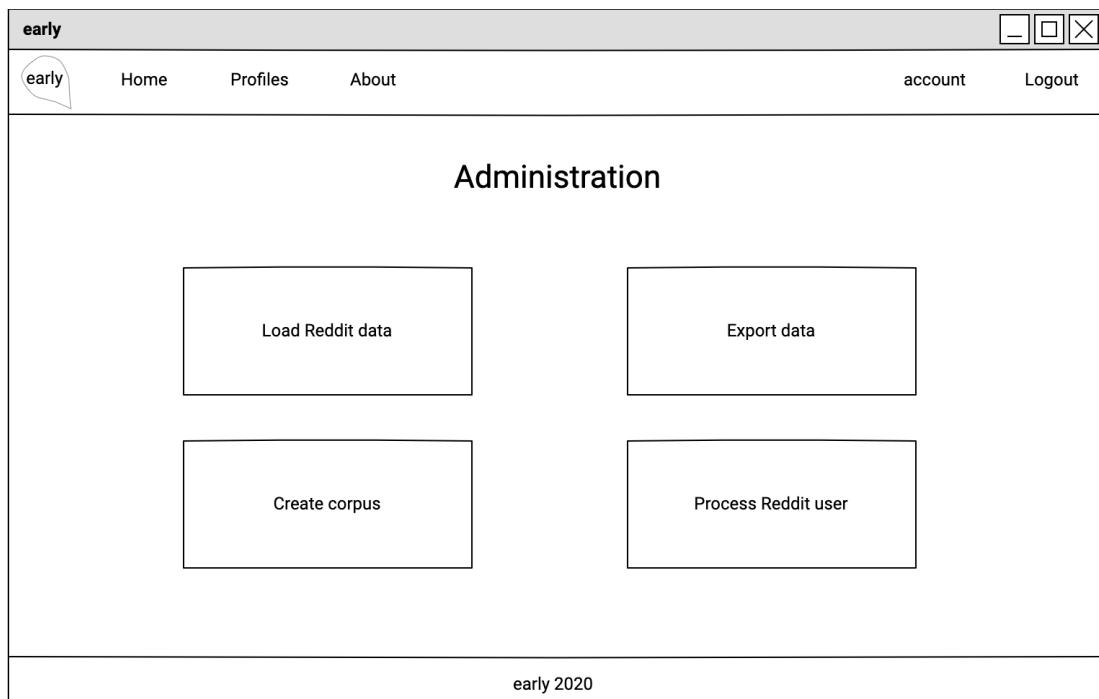


Figura 6.7.: Maqueta de la interfaz del panel de administración

Asimismo, se han agrupado los elementos, para que visualmente sea más fácil y cómodo para el usuario, ya que percibe ciertos elementos estructurados. Por lo tanto, las interfaces del panel de administración (figura 6.7) y las acciones del perfil de la cuenta (figura 6.8) siguen la misma estructura para que el usuario esté familiarizado con la aplicación.

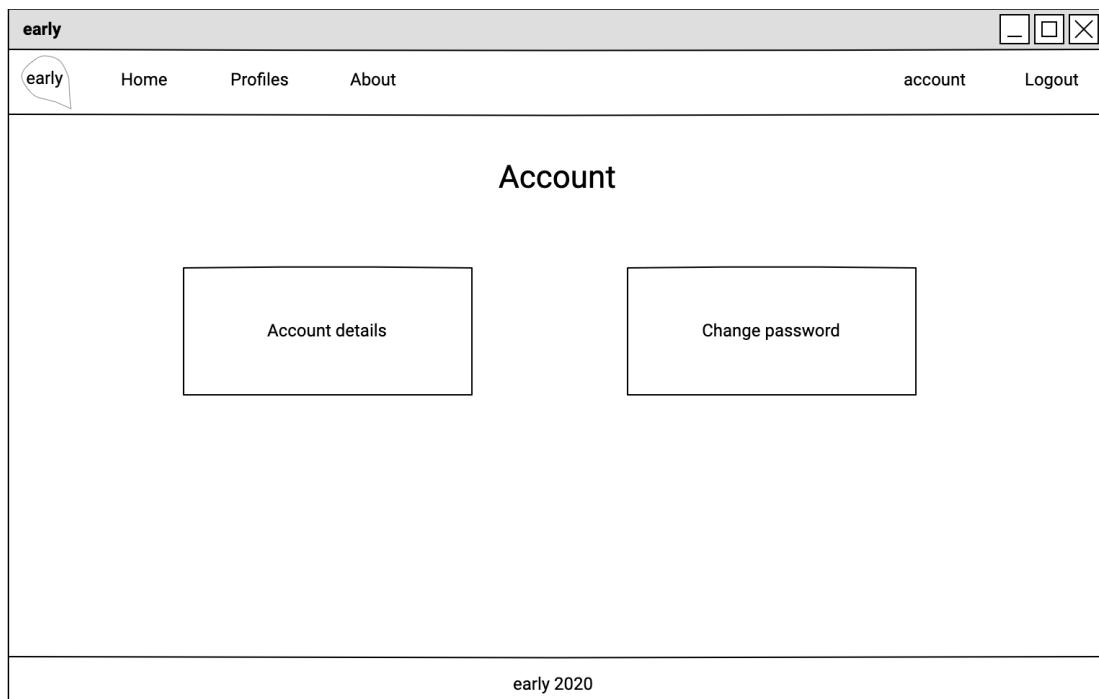


Figura 6.8.: Maqueta de la interfaz del panel de detalles de la cuenta

Las interfaces diseñadas se centran en el contenido que se quiere mostrar, no en la apariencia de esta. Por lo que, las distintas pantallas se han diseñado facilitando al usuario el acceso a funcionalidades e indicando el contenido con el que el usuario podrá interactuar.

Se han diseñado para tener una apariencia realista y que el usuario reconozca ciertas funciones.

Se ha intentado seguir la Ley de Fitts [34]. Esta ley de UX afirma que el tiempo requerido para alcanzar un elemento depende de la distancia al elemento, el ancho en la dirección del movimiento y el tamaño del elemento. Por lo tanto, cuanto mayor sea la distancia y menor el tamaño, más tardara. Para lograr esto, se han separando los elementos pequeños, dejando márgenes entre ellos, se han diseñando los botones o elementos *clickables* más grandes, para facilitarle la experiencia al usuario y dejando los elementos “sensibles” o destructivos alejados para evitar problemas.

Se ha optado por el uso de diálogos modales a la hora de editar contenidos (validar y editar el perfil -figura 6.9- y todas las acciones que puede realizar el administrador -figuras 6.10, 6.12, 6.13 y 6.14), para una mayor claridad y una correcta gestión, permitiendo al usuario añadir un nuevo corpus, cargar datos, exportar datos, etc. y seguir en la pantalla a partir de la que inicio el proceso.

early

Home Profiles About account Logout

Profile: user1

Age
12-20 years

Gender
Female

Location
Austria

Personality
Extraversion

Validate and edit data

Age
Gender
Location
Personality
Corpus
 Depression

Depression
Output confiance 0.8723
ert corrected this value
Output confiance 0.6723
Output confiance 0.6723

Save Close

early 2020

Figura 6.9.: Maqueta de la interfaz de validación y edición de un perfil

early

Home Profiles About account Logout

Load Reddit data

Create corpus

Load data

Subreddit
Number of users
Number of submissions
Number of comments per user
corpus ▾

Export data
Process Reddit user

Load data Close

early 2020

Figura 6.10.: Maqueta de la interfaz de carga de datos de Reddit

early

Profile: user1

Age: 12-20 years

Gender: Female

Location: Austria

Personality: Extraversion

Beck Depression Inventory

1. Sadness

I do not feel sad

I feel sad much of the time

I am sad all the time

I am so sad or unhappy

Context: I always feel sad

Output confiance 0.8723

2. Pessimism

Save Close

Depression

Output confiance 0.6723

Reason: system output confiance 0.6723

Validate and edit data

early 2020

Figura 6.11.: Maqueta de la interfaz del questionario de Beck

early

Load Reddit data

Create corpus

Export data

Export demographic data

Export corpus

All corpus

CSV

Export labeled data

All corpus

CSV

Export data Close

Export data

process Reddit user

early 2020

Figura 6.12.: Maqueta de la interfaz de exportación de datos

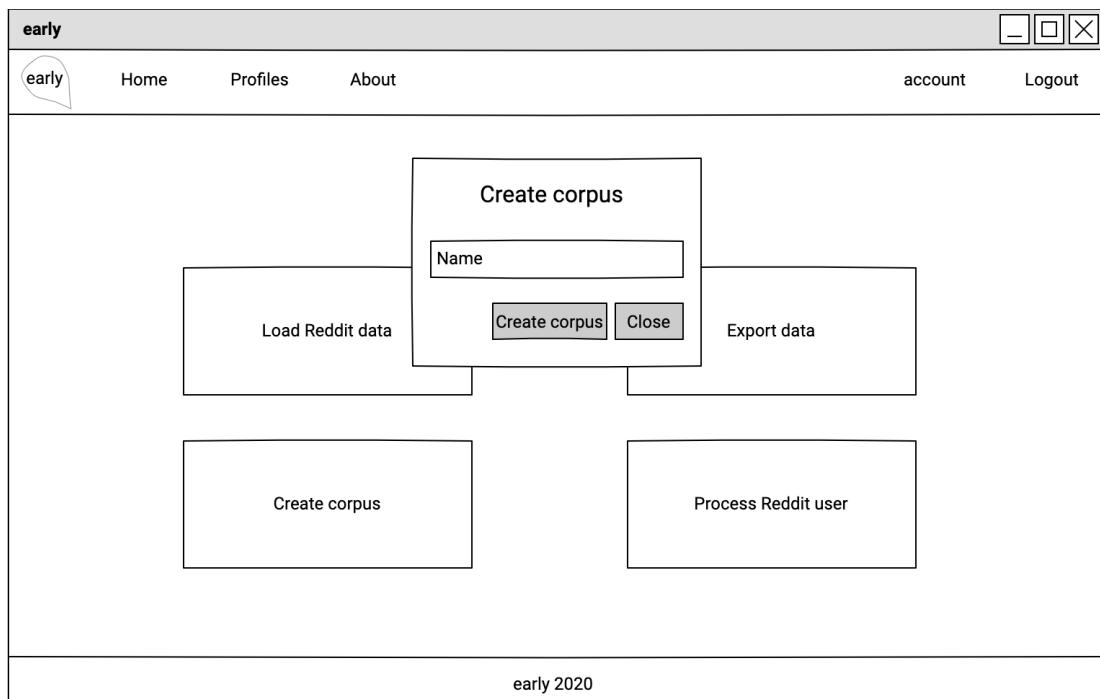


Figura 6.13.: Maqueta de la interfaz de creación de corpus

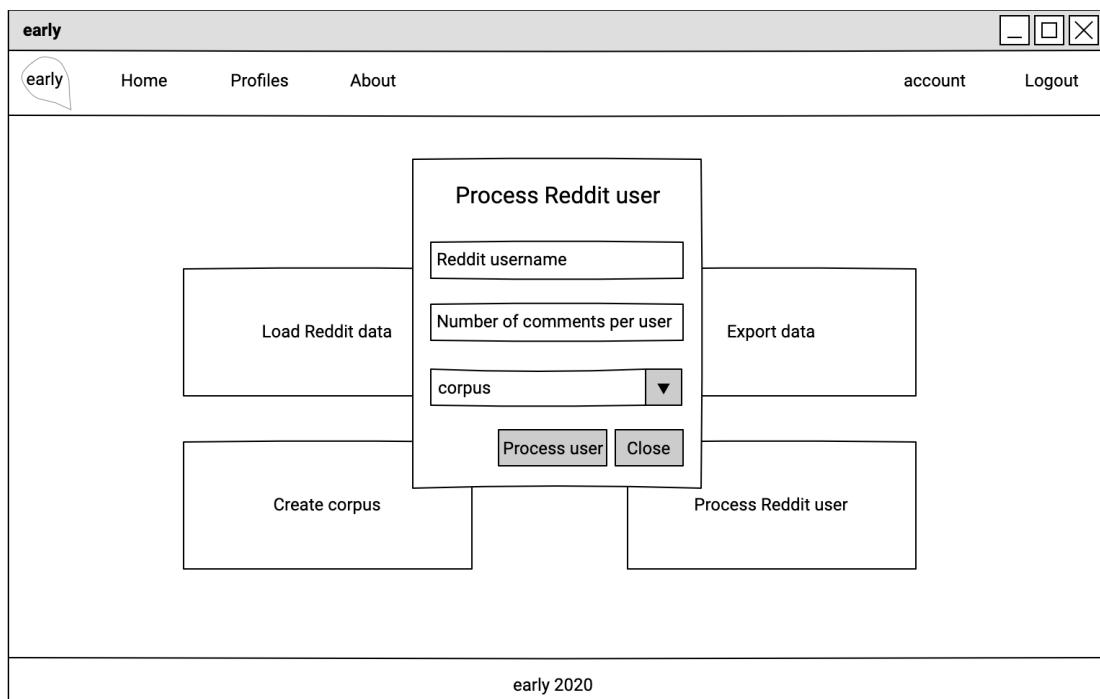


Figura 6.14.: Maqueta de la interfaz de procesamiento de un usuario

Al mismo tiempo, se ha tenido en cuenta la Ley de Jakob [35], la cual defiende que los usuarios prefieren trabajar en plataformas con una agrupación y distribución de elementos similar a la de otros sitios que ya conocen. Por esto, se ha decidido utilizar distintos elementos estándar como el menú de navegación, los diálogos modales, la búsqueda con filtros y tarjetas representando perfiles.

Se han realizado los distintos diseños teniendo en cuenta las últimas tendencias en diseño de interfaces web para que el usuario reconozca la aplicación como un sistema novedoso y lograr así que nuevos usuarios se interesen por esta aplicación. Combinado con el potencial del aprendizaje automático de los clasificadores, será un sistema muy interesante.

6.2. Diagramas de secuencia

Para un mejor entendimiento del flujo del sistema, se han realizado una serie de diagramas de secuencia para conocer en detalle el funcionamiento interno de las aplicaciones. La funcionalidad más compleja es la carga de datos de Reddit, ya que lanza el perfilador de forma interna. Se detallará esta funcionalidad y las relativas al usuario experto, hiladas todas en un único diagrama, simulando un conjunto de acciones que realizaría un usuario, pero no siendo obligatoria su ejecución en ese orden ni con todos los pasos. El resto de funcionalidades son los suficientemente estándar para no necesitar un diagrama de secuencia, por lo que no se han realizado. De esta forma, seguimos las pautas de las metodologías ágiles, donde se recomienda solo hacer aquella documentación necesaria y que realmente aporte valor al sistema.

6.2.1. Administrador: cargar datos

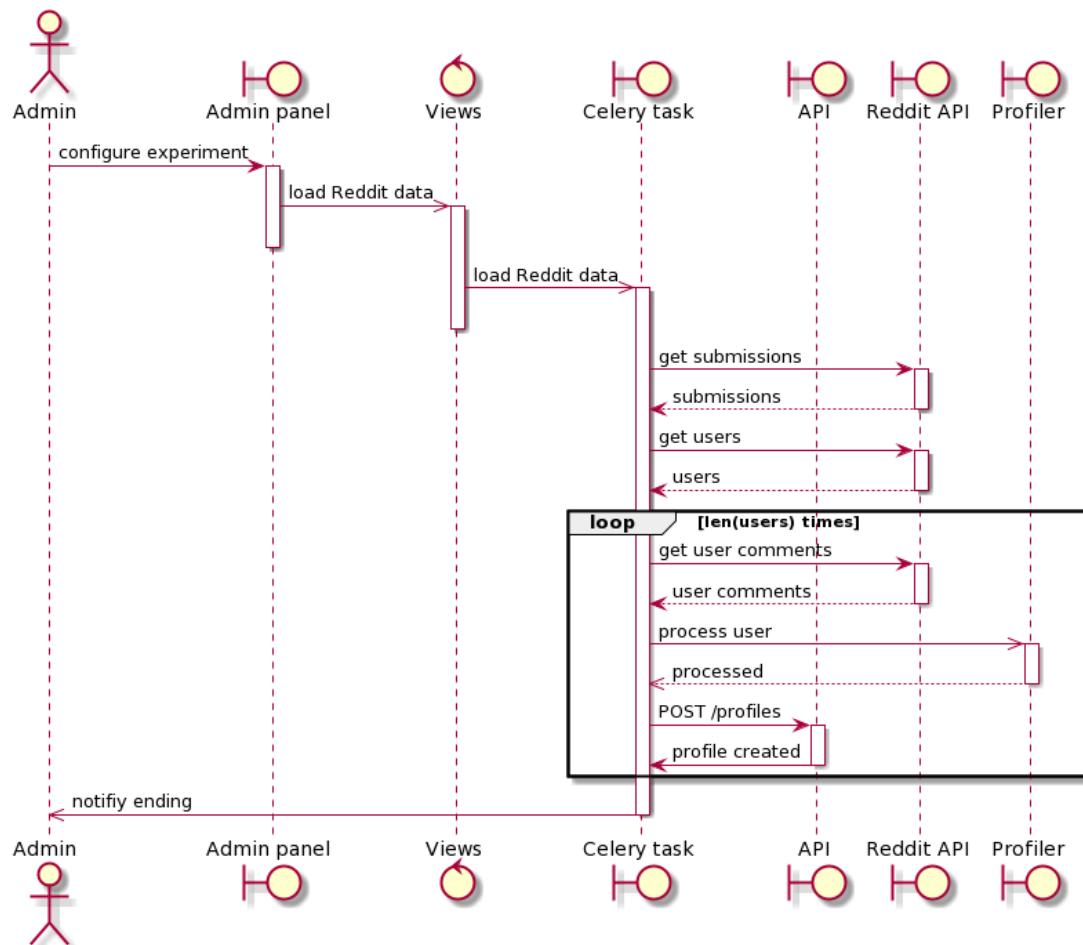


Figura 6.15.: Diagrama de secuencia de la carga de datos de Reddit

6.2.2. Experto: flujo completo

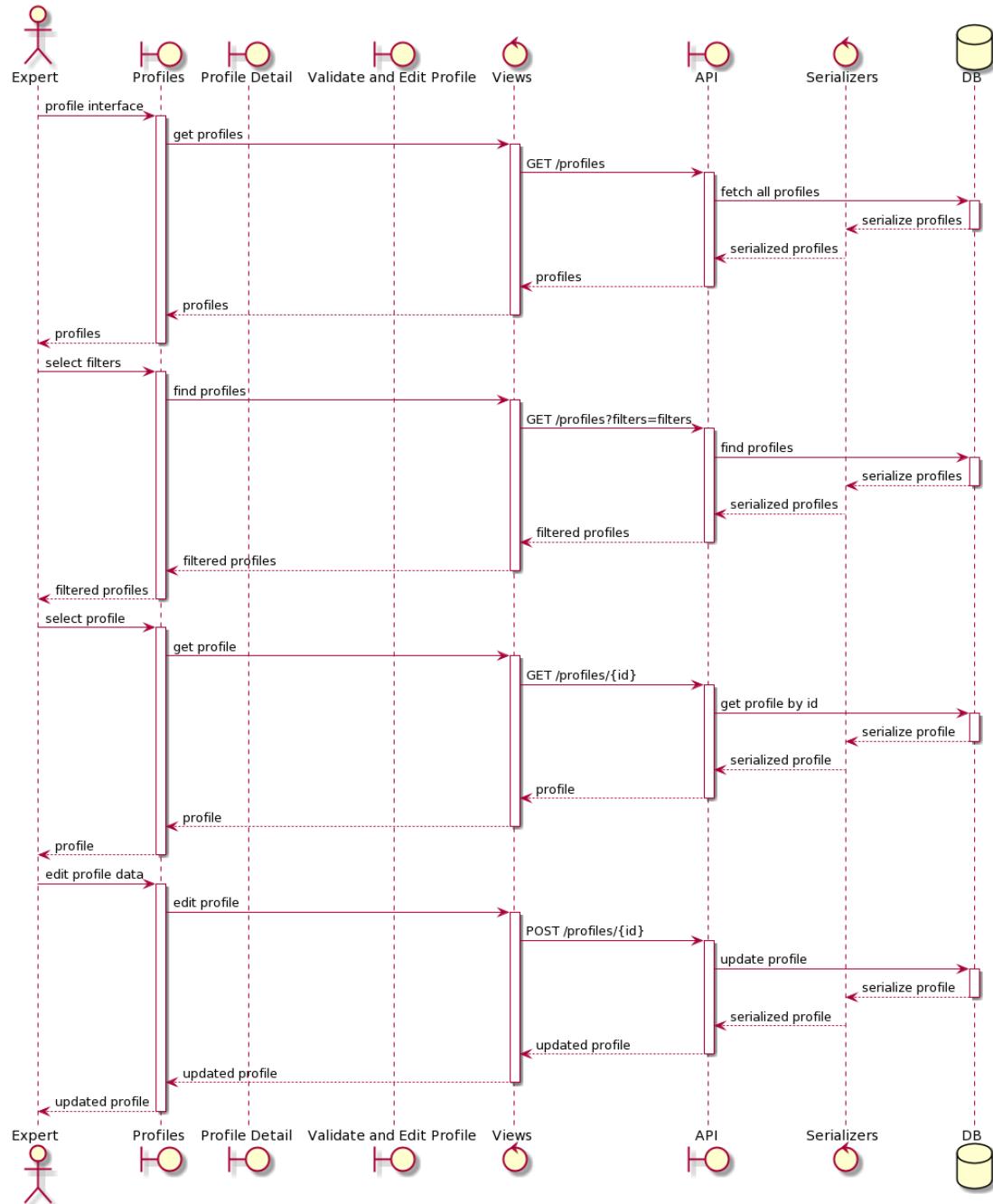


Figura 6.16.: Diagrama de secuencia de un flujo normal del usuario experto

6.3. Arquitectura global

El sistema sigue una arquitectura cliente-servidor, donde el usuario interacciona con los distintos elementos por medio del cliente, y el servidor se encarga de obtener los datos, prepararlos, realizar la lógica de negocio necesaria y enviarlos al cliente.

Al mismo tiempo, podemos diferenciar dos grandes componentes: la **aplicación web** y el **microservicio de perfilado (Profiler)**.

La aplicación web sigue una arquitectura cliente-servidor, se verá cada una de estas capas en detalle en la sección 6.3.1 del capítulo. También se consume la API de Reddit desde distintos componentes de la aplicación web y una API interna, que sigue una arquitectura REST por capas.

El microservicio de perfilado también sigue una arquitectura REST, formado por las capas web y distintos servicios (la lógica de negocio).

La figura 6.17 representa la arquitectura global del proyecto.

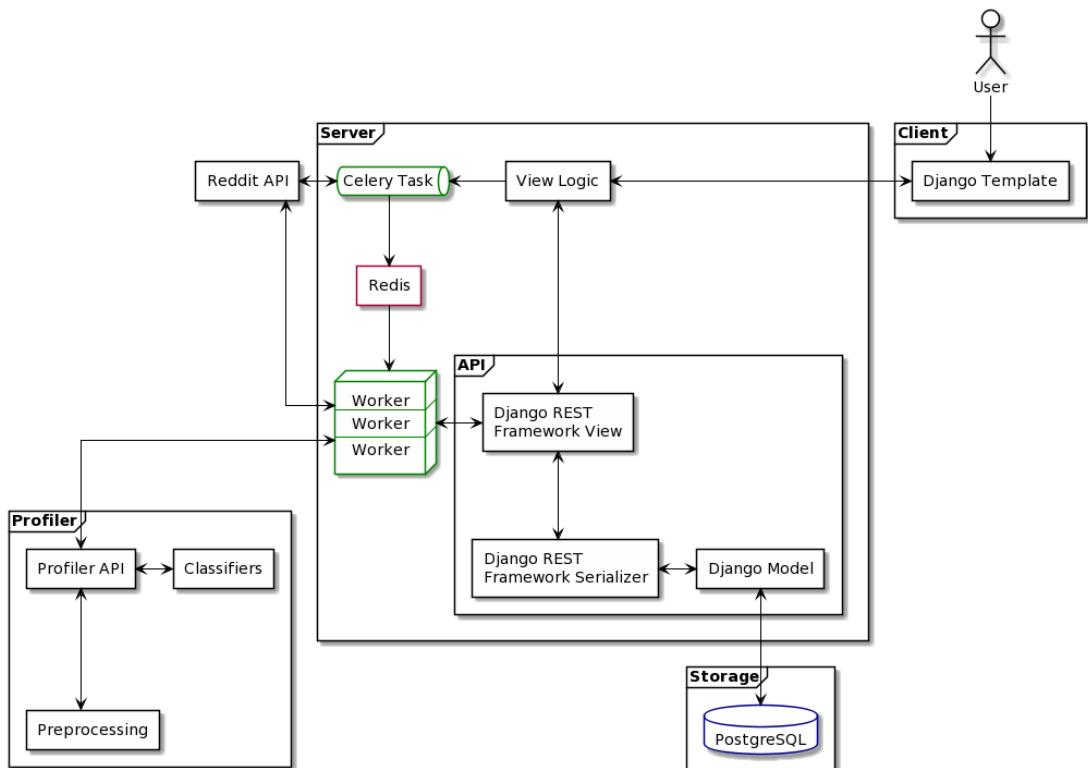


Figura 6.17.: Arquitectura global del proyecto

A continuación se explicará el detalle de los distintos componentes, siendo estos la aplicación web, el API interno de la aplicación web y el servicio del perfilador.

6.3.1. Aplicación web

Como ya se ha comentado, la aplicación web sigue una arquitectura cliente-servidor, donde el punto de interacción del usuario es el cliente, mediante *templates* de Django, y el servidor se encarga de realizar las distintas peticiones a las API que se consumen y, en otras capas, contiene lógica de negocio.

6.3.1.1. Cliente

El cliente de la aplicación es el punto de interacción del usuario. Está formado por las distintas pantallas, creados por medio de *templates* de Django.

Un *template* de Django contiene partes estáticas del HTML así como una sintaxis especiales para describir cómo insertar contenido dinámico. Django define una API estándar para cargar y renderizar *templates*. Utiliza el lenguaje de *templates* de Django para renderizar variables y sus valores, entre otros elementos.

6.3.1.2. Servidor

El servidor de nuestra aplicación web está formado por un fichero *views.py*, que es el controlador (se comunica con los *templates* y hace llamadas a otros servicios, componentes y API).

Métodos como actualizar perfil, buscar perfiles, etc. se comunican con la API interno para solicitar datos y realizar modificaciones. Por otro lado, funciones como cargar datos llaman a componentes internos, como las clases de *tasks.py*, para lanzar la tarea en segundo plano de Celery. Esta tarea, al mismo tiempo, se comunica con los métodos de *reddit.py* (que se encargan de hacer las llamadas a la API de Reddit), con la API interno y con el microservicio de perfilado.

6.3.1.3. Diagrama de clases

En la figura 6.18 se muestra el diagrama de clases que conforman el sistema.

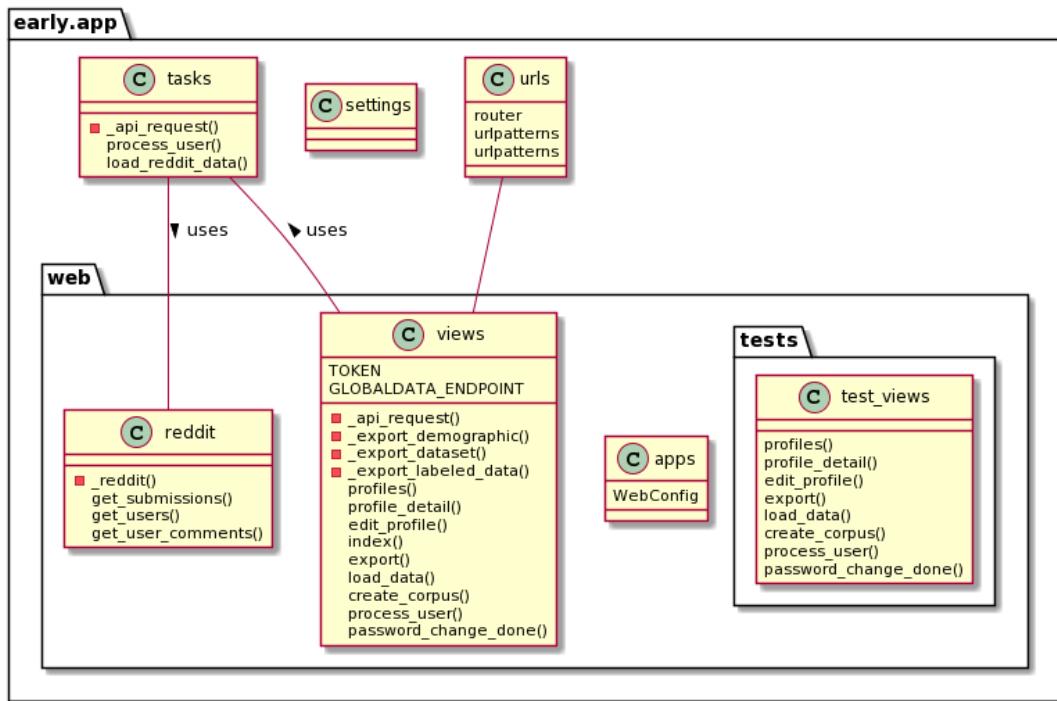


Figura 6.18.: Diagrama de clases de la aplicación web

6.3.1.4. Distribución en paquetes

La distribución en paquetes del lado del servidor de la aplicación se ha realizado por capas. El paquete base es `early.app`.

- `early.app.web`: Contiene los ficheros y clases principales de la API. Estos ficheros son:
 - `apps.py`: Este fichero sirve para incluir cualquier tipo de configuración para la aplicación.
 - `reddit.py`: Este fichero contiene las distintas funciones y métodos que realizan las llamadas a la API de Reddit. Es el punto de comunicación con esta API externo.
 - `urls.py`: Es el fichero de configuración de URL. Este nos permite mapear una URL a una función del fichero `views.py`.
 - `views.py`: Es el controlador. En este fichero se definen distintas clases o funciones, donde cada una de estas indica lo que pasa cuando cierta URL es accedida y una petición HTTP se realiza.

- *early.app.web.locale*: Contiene los distintos ficheros de internacionalización.
- *early.app.web.static*: Contiene ficheros donde se definen los estilos de la aplicación e imágenes que utiliza.
- *early.app.web.templates*: Son *templates* o ficheros HTML que definen la vista del cliente. Cuando una función en *views.py* renderiza un fichero HTML, le puede pasar objetos que puede mostrar en la interfaz. Con cada *template* se pueden tener ficheros CSS, JavaScript, imágenes, etc.
- *early.app.web.tests*: Contiene los distintos *tests*, de unidad y de integración del sistema.
- *settings.py*: Es el fichero central de configuración de proyectos Django. Permite configurar bases de datos, *templates*, *middleware*, internacionalización, *brokers*, etc.
- *tasks.py*: En este fichero se ha implementado las tareas en *background* con Celery, y distinta lógica para el correcto funcionamiento.

6.3.1.5. Internacionalización

La internacionalización es el proceso de diseñar software de manera tal que pueda adaptarse a diferentes idiomas y regiones sin la necesidad de realizar cambios de ingeniería ni en el código.

Para lograr la internacionalización, se ha utilizado el módulo de traducción ofrecido por Django. Esta librería, de código abierto, permite la internacionalización de aplicaciones Django de una forma muy sencilla.

En el fichero *settings.py*, se indican los idiomas en los que realizar la internacionalización y con dos comandos sencillos se realiza este proceso. Primero, se recogen todos los puntos donde se ha indicado que se trata de un texto a internacionalizar y se vuelca automáticamente en un fichero *.po*. Los ficheros *.po* se alojan en la carpeta de cada idioma dentro del directorio *locale*. Después, de forma manual, se indica la traducción en cada idioma, y finalmente con otro comando se compilan los mensajes para que se apliquen los cambios.

Por lo tanto, la aplicación soporta internacionalización en todos sus componentes, si bien de forma práctica solo se presentan las interfaces en inglés y castellano.

6.3.2. API interno

La API interna está implementado con Django REST Framework y está compuesta por las capas: *viewset*, *serializer* y *model* (las entidades).

6.3.2.1. Capa web (*viewset*)

Es la API REST de la aplicación. Contiene los controladores REST (fichero *View.py*) que atenderá las peticiones de los clientes y, utilizando la capa *serializer*, construirá las respuestas a las mismas.

La comunicación se realizará a través del paradigma REST, que es un estilo arquitectónico para construir aplicaciones distribuidas inspirado en las características de la web y que emplea directamente HTTP para la obtención de datos. A través de las siguientes peticiones HTTP podremos acceder a los distintos recursos.

- Los métodos de acceso especifican la acción que se quiere realizar sobre un recurso. De esta forma al usar GET se solicitará una representación del recurso pedido. POST crea una nueva representación de un recurso y PUT la modifica. También existe el método DELETE que elimina el recurso especificado.
- Dependiendo del método que se emplee, la ruta sobre la que se realiza la petición HTTP es distinta. Tanto PUT como DELETE se realizan sobre recursos específicos, por lo tanto la ruta será del estilo de: /recurso/{id_recurso}. POST se emplea habitualmente para crear un nuevo recurso de la forma /recurso. El método GET puede ser usado tanto en recursos colección como individuales.

Cuando se recibe una petición, el controlador indica la *query* que hay que realizar a la base de datos, el serializador a utilizar, y los permisos necesarios para realizar la consulta. Gracias a la capa del serializador, el modelo recuperará los datos de la base de datos, y los traducirá a un lenguaje que entienda el agente que realizó la petición.

6.3.2.2. Serializadores

Los serializadores permiten que los datos complejos, como conjuntos de consultas e instancias de modelos, se conviertan en tipos de datos nativos de Python que luego se pueden representar fácilmente en JSON, XML u otros tipos.

Los serializadores también proporcionan deserialización, lo que permite que los datos analizados se vuelvan a convertir en tipos complejos después de validar primero los datos entrantes.

Los serializadores en el marco REST de Django, proporcionan una clase *Serializer* que brinda una forma genérica y poderosa de controlar la salida de las respuestas. Asimismo una clase *ModelSerializer*, que proporciona un atajo útil para crear serializadores que tratan con instancias de modelos y conjuntos de consultas.

En los serializadores de Django REST Framework, se pueden sobreescribir los métodos de guardado, actualización, etc. para realizar acciones personalizadas. De esta forma logramos un mayor control sobre los datos que deseamos obtener o persistir.

6.3.2.3. Capa modelo

La capa modelo alberga las entidades persistentes. En el fichero *models.py* se definen todas las entidades, tipos, relaciones y demás que van a formar parte de nuestro sistema.

Gracias a Django, se puede hacer uso de sus *models* para indicar distintos tipos como enumerados (*TextChoices*), campos de texto (*TextField*), campos booleanos (*BooleanField*), relaciones (*ForeignKey*), etc. y se pueden indicar atributos adicionales como *constraints* de unicidad, *null*, longitud máxima, etc.

6.3.2.4. Diagrama de clases

En la figura 6.19 se muestra el diagrama de clases que conforman el sistema.

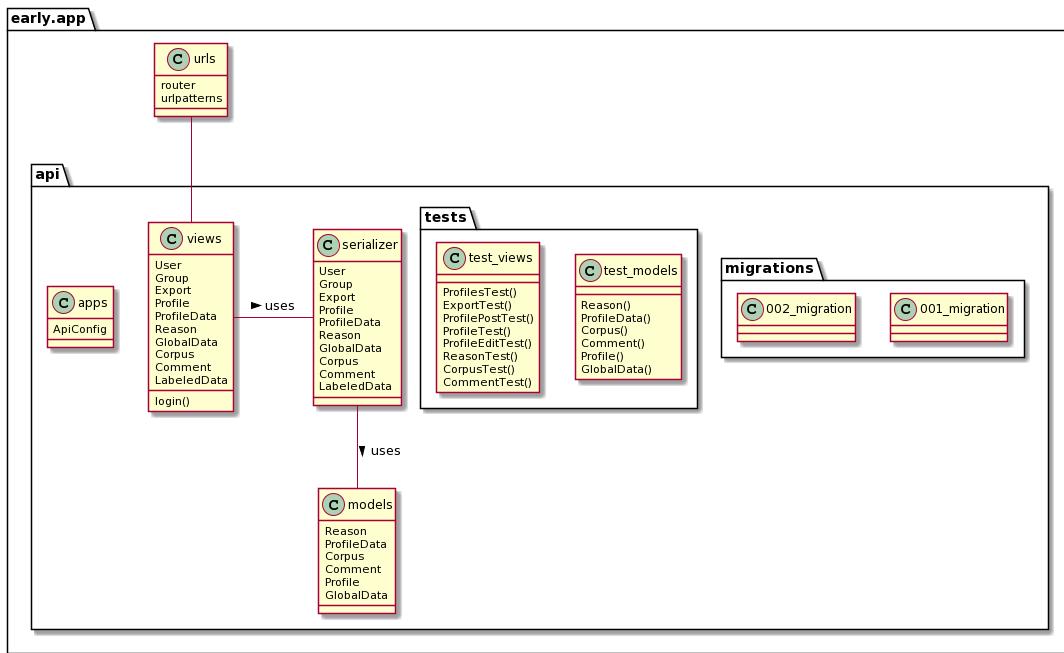


Figura 6.19.: Diagrama de clases del API interno

En el capítulo 5 se expuso el diagrama entidad-relación y se describieron todas las entidades en detalle.

6.3.2.5. Distribución en paquetes

La distribución en paquetes del API se ha realizado por capas. El paquete base es `early.app.api`.

- **`early.app.api`:** Contiene los ficheros y clases principales del API. Estos ficheros son:
 - `apps.py`: Este fichero es para incluir cualquier tipo de configuración para la aplicación.
 - `models`: Contiene todas las clases que representan entidades persistentes.
 - `serializers.py`: Contiene los distintos serializers para cada clase del modelo. En algún caso se sobreescreiben los métodos de guardado y actualización para que se adecuen a nuestras necesidades.
 - `views.py`: Es el controlador. En este fichero se definen las distintas funciones/clases (`ViewSet`). Estas definen lo que sucede cuando una URL es accedida, o se realiza

una llamada contra esta.

- *early.app.api.migrations*: Contiene los *scripts* de migraciones de la base de datos.
- *early.app.api.tests*: Contiene los distintos *tests*, de unidad y de integración del sistema.
- *urls.py*: En este fichero se definen las rutas que permiten realizar las distintas peticiones REST en el servicio.

6.3.3. Servicio perfilador

El microservicio de perfilado sigue una arquitectura REST. Se expone un punto de entrada (*profiler.py*), donde se definen las rutas y las distintas peticiones REST, y luego una serie de componentes o servicios, donde se realiza la lógica de negocio.

6.3.3.1. Capa web

Es la capa de entrada de peticiones. Recibe las distintas peticiones REST y realiza la acción correspondiente. Se apoya en los distintos componentes donde se define la lógica de negocio.

La base de los servicios y peticiones REST se ha explicado en la subsección 6.3.2.1

6.3.3.2. Lógica de negocio

En esta capa residen las distintas clases que dan valor al microservicio. Cada una realiza su labor adecuada (preprocesado, clasificación, etc.) y son un pilar fundamental para que, cuando se reciba una petición, se apliquen los procesos necesario para devolver la respuesta adecuada.

La lógica de negocio de este microservicio consiste en el preprocesado y cálculo de *features* a partir de los datos que se envían en la petición (comentarios de redes sociales) y la predicción de distintos aspectos del perfil.

6.3.3.3. Diagrama de clases

En la figura 6.20 podemos ver el diagrama de clases del microservicio de perfilado.

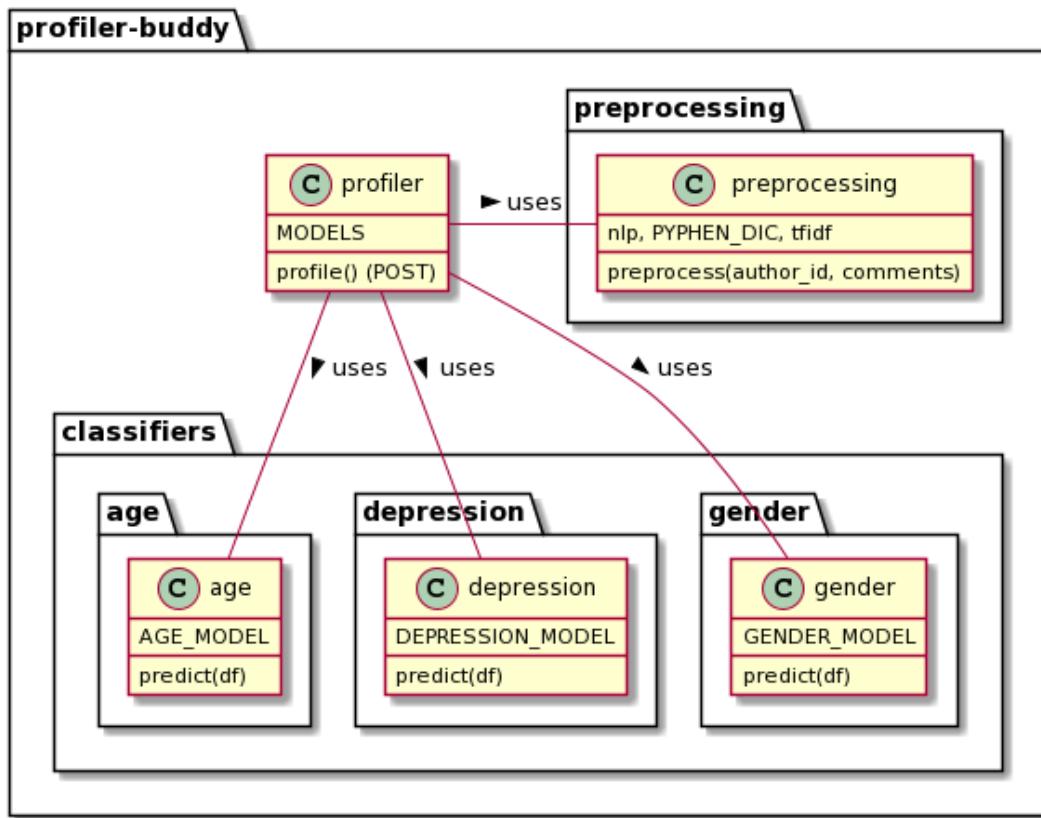


Figura 6.20.: Diagrama de clases del microservicio de perfilado

6.3.3.4. Distribución en paquetes

La distribución en paquetes del microservicio de perfilado se ha realizado por capas. El paquete base es *profiler-buddy*.

- *profilerbuddy*: Paquete principal, contiene directamente el punto de entrada del servicio, es decir, donde se reciben las peticiones REST (*profiler.py*).
- *profilerbuddy.classifiers*: Paquete que contiene los clasificadores de los distintos aspectos demográficos y el clasificador del perfil en depresión o no. Este paquete contiene a su vez un paquete por cada aspecto:
 - *profilerbuddy.classifiers.gender*: Abarca todos los ficheros necesarios y el modelo de clasificación para predecir el género de un perfil.
 - *profilerbuddy.classifiers.depression*: Contiene todos los ficheros necesarios y el mo-

delo de clasificación para predecir el nivel de depresión de un perfil.

- *preprocessing*: Paquete que contiene los ficheros necesarios para calcular las *features* que utilizarán los clasificadores para realizar su tarea.

Capítulo 7

PLANIFICACIÓN Y EVALUACIÓN DE COSTES

En este capítulo se exponen la planificación y el presupuesto inicial y los resultados finales en cuanto a estos aspectos.

7.1. Planificación inicial

Una vez conocidos los objetivos y herramientas de trabajo se realizará la planificación del proyecto. Para conseguirlo, se realiza un análisis previo donde se tendrán en cuenta los recursos y el tiempo disponible. Existe una gran limitación de recursos, ya que las tareas son realizadas por una única persona, que produce una sobrecarga en la planificación del proyecto. Habitualmente, se dispone de un equipo de más de una persona, existen tareas que se pueden realizar en paralelo, disminuyendo así en gran medida la duración del proyecto.

Para realizar la planificación del proyecto es necesario dividirlo en secciones o tareas con unos objetivos claros. La metodología que se sigue en el proyecto es Scrum, que divide el proyecto en los llamados *sprints*, que son bloques de tiempo donde se definen las tareas a realizar. Todos los diferentes *sprints* incluyen los mismos pasos:

- Creación del Sprint Backlog que contendrá las distintas historias de usuario a realizar.
- División de las distintas historias de usuario en tareas más manejables.

- Diseño de cada una de estas tareas.
- Implementación.
- Pruebas.

Todo el desarrollo de la aplicación se llevó a cabo basándonos en *sprints* que siempre se desarrollaron en un periodo tres semanas. Se estableció que cada *sprint* comprendería cuarenta puntos de historias, tomando como que un punto de historia es una hora, trabajando así

$$40 \times 1h = 40h$$

horas por *sprint* (casi catorce horas semanales).

El proyecto se realizaría en **18 sprints**, con un total de **720 horas** de trabajo.

7.2. Coste

Al realizar una evaluación de costes de un proyecto, se debe tener en cuenta tanto los gastos en personal, como también gastos en materiales y licencias.

El proyecto se lleva a cabo en un ordenador personal, se pone en marcha, una parte, en un servidor gratuito, y otra en una instancia de pago de AWS, y el coste de las licencias es 4 meses x 8,69€, ya que se ha utilizado Google Colab Pro y el resto son herramientas y tecnologías gratuitas. No se tienen en cuenta costes sociales ni indirectos.

Siendo el precio del ordenador personal 3.659,00€ y su tiempo usado 18 *sprints* de tres semanas de duración cada uno, es decir, 54 semanas. La amortización del equipo ofimático es la siguiente:

$$\frac{3,659,00\text{€}}{4\text{años} \times 12\text{meses/año}} \times 12\text{meses} = 914,75\text{€}$$

Licencias: 34,76€

Servidor Web: 9,20€

- Balanceador de carga: \$0.027 por hora (15 horas en total)
- EC2 AWS Fargate Memoria: \$0.00511 por hora (15 horas en total)

- EC2 AWS Fargate vCPU: \$0.04656 por hora (15 horas en total)
- EC2 \$0.4656 por hora (21 horas en total)

Este proyecto cuenta con tres recursos:

- Dos *product owner*: Patricia Martín Rodilla y Javier Parapar. Encargados de atender a las revisiones del proyecto, realizar evaluaciones, dar *feedback* y definir los requisitos del trabajo.
- Una analista programadora: Paloma Piot. Encargada de las tareas de análisis y diseño, como también de las tareas de desarrollo y realización de las pruebas del sistema.

Recurso	Coste	Tiempo de trabajo	Total
Product Owners	28,509€/h	72h	2.052,65€
Analista programadora	21,255€/h	720h	15.303,60€
Equipo ofimático	3.659,00€	12 meses	914,75€
Licencias y servidor web	43,96€	-	43,96€
			18.314,96€

El coste de los recursos humanos se ha calculado tomando el sueldo base promedio que indica glassdoor¹ a 5 de junio de 2021, sumando los costes de la seguridad social. Para el cálculo se ha tomado como base para ese sueldo una jornada de 40 horas semanales.

7.3. Seguimiento

Como se comentó en la sección de metodología, el proyecto se realizará en *sprints*. Se definieron un total de dieciocho *sprints*, con una duración fija de tres semanas, en el que se encuentran definidas las tareas a realizar. A continuación se detallan los *sprints* que se realizaron a lo largo del proyecto.

¹<https://www.glassdoor.es/>

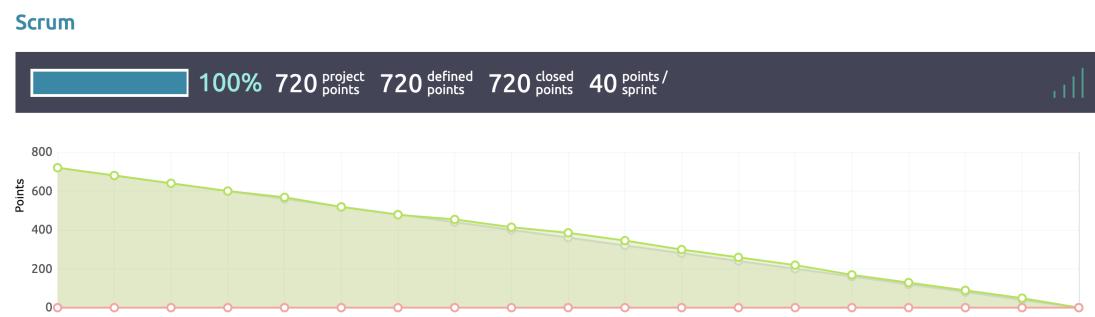


Figura 7.1.: Evolución del desarrollo del proyecto

7.3.1. Sprint 0 (20.01.2020 - 10.02.2020)

El *sprint* 0 se dedicó para una primera toma de contacto con el nuevo proyecto, poniendo orden a las ideas para su correcta ejecución.

Se realizaron las historias de usuario de:

- Estudio sobre el flujo del sistema
- Estudio sobre el modelo de datos
- Estudio sobre el estado del arte de la problemática a abarcar
- Parsear el conjunto de datos inicial
- Extracción del léxico relacionado con los campos del perfilado
- Extracción de expresiones que den pistas sobre los campos a perfilar

7.3.2. Sprint 1 (10.02.2020 - 02.03.2020)

En el primer *sprint* se abarcaron las historias de usuario de:

- Preprocesado de textos
- Arquitectura del sistema
- Posibles aproximaciones para identificar aspectos del perfilado

- Enriquecer listas de léxico

7.3.3. Sprint 2 (02.03.2020 - 23.03.2020)

El segundo *sprint* se dedicó en la realización de:

- Arquetipo aplicación web
- Perfilado inicial género

7.3.4. Sprint 3 (23.03.2020 - 13. 04.2020)

El tercer *sprint* se centró en:

- Nutrir el *matcher* de género
- Añadir sinónimos del léxico inicial, vectores de sinónimos y la similitud entre ellos
- Investigar si es posible obtener meta-information de los usuarios
- Maquetas aplicación

7.3.5. Sprint 4 (13.04.2020 - 04.05.2020)

En el cuarto *sprint* se realizó únicamente trabajo relacionado con la aplicación web:

- *Back-End* aplicación
- Configuración CI/CD y repositorio
- Gestión de usuarios y roles
- Inicio sesión y terminar sesión
- Cambiar contraseña

7.3.6. Sprint 5 (04.05.2020 - 25.05.2020)

El quinto *sprint* se empleó para añadir funcionalidades a la aplicación:

- Funcionalidades administrador
- Funcionalidades experto
- Ver detalle perfil
- Corregir datos perfil

7.3.7. Sprint 6 (25.05.2020 - 15.06.2020)

El sexto *sprint* se dedicó para investigar la API de Reddit e implementar las *queries* a realizar.

7.3.8. Sprint 7 (15.06.2020 - 06.07.2020)

El séptimo *sprint* se centró en aplicar una estrategia combinada de filtrado de léxico y algún algoritmo de clasificación en una colección anotada de datos.

7.3.9. Sprint 8 (06.07.2020 - 27.07.2020)

En el octavo *sprint*, se conectaron las *queries* implementadas en el *sprint* 6 y se realizó un fichero de léxico con trazabilidad a cada pregunta del formulario de depresión.

7.3.10. Sprint 9 (27.07.2020 - 17.08.2020)

En el *sprint* 9, se probó Snorkel² como posible opción para los modelos y se añadieron nuevas *features* al clasificador de género.

7.3.11. Sprint 10 (17.08.2020 - 07.09.2020)

El décimo *sprint* abarcó las tareas de:

²<https://www.snorkel.org/>

- Integrar una estrategia no supervisada al clasificador de género
- Recuperar el contexto del léxico de depresión entre los textos de Reddit
- Añadir enumerados de rango de edades y personalidad a la aplicación
- Exportar datos en formato CSV y JSON

7.3.12. Sprint 11 (07.09.2020 - 28.09.2020)

Durante la realización del *sprint* 11, se integraron los clasificadores en la aplicación, se redactó la tarea de clasificación de género y se añadieron *tests* de unidad, integración, aceptación y carga.

7.3.13. Sprint 12 (28.09.2020 - 19.10.2020)

El *sprint* 12 se centró en implementar la búsqueda o filtrado de perfiles y obtener colecciones de datos etiquetados para la tarea de edad.

7.3.14. Sprint 13 (19.10.2020 - 09.11.2020)

En el décimo tercer *sprint* se normalizaron los valores numéricos de las *features* del clasificador de género, se añadieron *topics* LDA y se volvió a entrenar el modelo de género.

7.3.15. Sprint 14 (09.11.2020 - 30.11.2020)

En el décimo cuarto *sprint* se amplió la integración con Reddit, se añadió la exportación de individuos y comentarios y se incluyó la funcionalidad de procesar un usuario por su nombre de Reddit.

7.3.16. Sprint 15 (30.11.2020 - 21.12.2020)

En el décimo quinto *sprint* se rediseñó la aplicación, se añadió el microservicio de perfilado (se desacopló de la aplicación) y se implementó la funcionalidad de ver y editar detalles de la cuenta de usuario.

7.3.17. Sprint 16 (21.12.2020 - 11.01.2021)

El *sprint* 16 se centró en realizar una primera aproximación e investigación en lo relacionado con la tarea de auto-completar el formulario de depresión.

7.3.18. Sprint 17 (11.01.2021 - 01.02.2021)

El *sprint* 17 se centró en ampliar e implementar en mayor profundidad el modelo de auto-completar el formulario de depresión, se redactó esta tarea para la memoria, se integró el modelo en el perfilador y se finalizó la redacción de la memoria.

Capítulo 8

PRUEBAS Y RESULTADOS

En este capítulo se exponen las pruebas realizadas a lo largo del proyecto, los problemas conocidos y los resultados extraídos de estas.

Las pruebas en cualquier proyecto software es una parte muy importante de su desarrollo. Las metodologías de pruebas están orientadas a probar la existencia de errores y no la ausencia de ellos, se persigue, por tanto, detectar el mayor número de fallos posibles. El diseño de pruebas permite conocer de forma objetiva e independiente información sobre la calidad del software y el riesgo en el uso habitual de la aplicación.

8.1. Pruebas

Se realizaron pruebas de unidad, integración y de aceptación sobre la aplicación desarrollada. También se complementó esta sección con una batería de pruebas de carga (o rendimiento).

8.1.1. Pruebas de unidad

Las pruebas de unidad suelen ser pruebas automatizadas que garantizan que una sección de la aplicación cumple con su diseño y se comporta según lo previsto.

Este tipo de pruebas son de vital importancia ya que su correcto desarrollo ahorra tiempo y dinero en el ciclo de desarrollo de un sistema.

Las pruebas unitarias nos ayudan a corregir errores en las primeras etapas del desarrollo, y pueden servir como documentación del proyecto. Al mismo tiempo, si se realiza una refactorización del código o se cambia algún aspecto interno, pero mantenimiento la funcionalidad, nos sirven para detectar si nuestros cambios se han realizado correctamente o no.

Se han realizado pruebas unitarias en los modelos de la API interno y en los distintos métodos de las clases de los servicios.

En el siguiente fragmento de código (8.1) se puede apreciar el formato de los *tests* de unidad.

```
class ProfileDataModelTest(TestCase):
    @classmethod
    def setUpTestData(cls):
        ProfileData.objects.create(
            age="20-30", gender="Female", location='AU',
            personality="Agreeableness", depressed=False)

    def test_profile_data(self):
        data = ProfileData.objects.get(id=1)
        self.assertIsInstance(data.age, str), True)
        self.assertIsInstance(data.gender, str), True)
        self.assertEqual(data.location.name, "Australia")
        self.assertIsInstance(data.personality, str), True)
        self.assertIsInstance(data.depressed, bool), True)
```

Listing 8.1: Test de unidad

8.1.2. Pruebas de integración

Las pruebas de integración constituyen una fase en la que los distintos componentes del sistema se combinan y se prueban como un conjunto. Se realiza este tipo de pruebas para evaluar el cumplimiento de un sistema o componente con requisitos funcionales especificados.

Los casos de prueba se suelen centrar principalmente en las interfaces y flujo de datos entre los módulos. Por este motivo, se han hecho pruebas de integración para cada tipo de petición HTTP disponible (tanto en la API interno, en el microservicio de perfilado y las funciones del controlador de la aplicación web).

En el siguiente fragmento de código (8.2) se puede apreciar el formato de los tests de integra-

ción contra los distintos *endpoints*.

```
class ProfileEditTestCase(APITestCase):
    def setUp(self):
        self.username = 'john_doe'
        self.password = 'foobar'
        self.user = User.objects.create(
            username=self.username, password=self.password)
        self.client.force_authenticate(user=self.user)

    def test_put_profile(self):
        profile = self.client.post(PROFILES_ENDPOINT, BODY, format='json')
        data = json.loads(str(profile.content, encoding='utf8'))
        edition = {
            "validated_data": {
                "age": "20-30",
                "gender": "Female",
                "location": "Sweden",
                "personality": "Agreeableness",
                "depressed": True
            },
            "validated_by": {"username": "me"}
        }
        response = self.client.put(
            PROFILES_ENDPOINT + str(data['id']) + '/', edition, format='json')
        updated_data = json.loads(str(response.content, encoding='utf8'))
        self.assertEqual(response.status_code, 200)
        self.assertEqual(updated_data['is_valid'], True)
```

Listing 8.2: Test de integración

8.1.3. Pruebas de aceptación automatizadas

Las pruebas de aceptación son un tipo de pruebas que determinan si los requisitos del sistema se cumplen. Evalúan si los requisitos de negocio se satisfacen o no.

Se han realizado este tipo de pruebas bajo la metodología *Behaviour-driven development* (BDD). BDD defiende que en un primer momento se crean las pruebas, y más adelante la funcionalidad que haga que estas pruebas sean exitosas.

Los escenarios en BDD indican cómo debe comportarse una funcionalidad en particular según los parámetros de entrada proporcionados a la prueba. Cada caso de prueba se basa en una historia de usuario escrita en un lenguaje muy sencillo, específico de dominio (Gherkin). Con Gherkin podemos implementar los distintos escenarios de automatización de pruebas de Selenium.

Las ventajas de utilizar BDD son:

- Garantiza que todas las partes interesadas del proyecto estén alineadas en cuanto a la funcionalidad esperada y se trabaje de forma colectiva para mejorar el sistema.
- Como las pruebas están escritas en Gherkin, cada miembro del equipo puede participar en la creación de estas.
- Las pruebas de BDD son más reutilizables y modulares, ya que solo cambian si hay algún cambio en los requisitos de negocio.

En el siguiente fragmento de código 8.3 podemos ver cómo se definen los distintos escenarios de pruebas.

Feature: Profiles actions

Scenario: Access the profiles page

```
Given a logged user
And an existing profile
When I load the profiles page
Then I get a list of profiles
```

Scenario: Access the profile detail page

```
Given a logged user
And an existing profile
When I load the profile detail page
Then I get the profile details
```

Scenario: Validate profile

```
Given a logged user
And an existing profile
When I validate an existing profile
Then I am redirected to the success edit page
```

```
Scenario: Validate invalid profile
  Given a logged user
    And an existing profile
  When I validate with wrong values an existing profile
  Then I get a submission error
```

Listing 8.3: Escenarios escritos en Gherkin

Estos se ejecutarán iniciando un navegador y realizando las acciones descritas en los *tests* de Selenium y Behave.

Como podemos observar en el siguiente fragmento de código Python (código 8.4). Con la anotación `@given` estamos indicando a qué parte del escenario nos estamos refiriendo. Se ejecutará dicha función solo cuando se esté ejecutando esa parte de la prueba y si el texto escrito corresponde totalmente.

En nuestro caso, cuando el escenario “Validate profile” empiece su ejecución, lo primero que realizará será la ejecución del “Given”, por lo que se buscará entre las anotaciones `@given`, aquellas que se indican en el escenario (es decir, “a logged user” y “an existing profile”). De esta forma ya tendremos nuestras precondiciones satisfechas.

```
@given('a logged user')
def step_impl(context):
    u = UserFactory(username='foo', email='foo@example.com')
    u.set_password('bar')
    u.save()

    # Log user
    br = context.browser
    br.get(context.base_url + '/account/login/')
    br.find_element_by_name('username').send_keys('foo')
    br.find_element_by_name('password').send_keys('bar')
    br.find_element_by_name('submit').click()

@given('an existing profile')
def step_impl(context):
    context.experiment = ProfileFactory()
    context.experiment.save()
```

Listing 8.4: Funciones que simulan un usuario logeado y la creación de un perfil

A continuación, se ejecutará la parte del “when”, es decir, la funcionalidad que queremos probar (“I validate an existing profile”). En el código que se puede ver a continuación (código 8.5) se puede apreciar los pasos a realizar para ejecutar la funcionalidad de validar un usuario.

```
@when('I validate an existing profile')
def step_impl(context):
    br = context.browser
    br.get(context.base_url + '/profiles/1/')

    assert br.find_element_by_class_name('badge-pill').text == 'Not processed'

    # Open validation modal
    br.find_element_by_class_name('btn-outline-primary').click()

    time.sleep(1)

    # Fill validation form and submit it (valid version)
    br.find_element_by_name('gender').send_keys('Female')
    br.find_element_by_name('depressed').click()
    br.find_element_by_name('submit').click()
```

Listing 8.5: Función que realiza la validación de un perfil en el navegador

Finalmente, la anotación @then (código 8.6) albergará todos los pasos y las condiciones que se deben cumplir para que el caso de prueba se supere con éxito.

```
@then('I am redirected to the success edit page')
def step_impl(context):
    br = context.browser

    # Checks success status
    assert br.current_url.endswith('/profiles/1/edit/')
    assert br.find_element_by_tag_name(
        'h1').text == "Profile: behave_test_1"
    assert br.find_element_by_class_name(
        'fa-check-circle') is not None
    assert br.find_element_by_class_name(
        'badge-pill').text == 'Depression'
```

Listing 8.6: Función que realiza la validación de un perfil en el navegador

Se han implementado estas pruebas para todas las funcionalidades de la aplicación.

Para la ejecución automatizada de las pruebas, se ha utilizado el *WebDriver* de Selenium, el cual configura un navegador que se inicia antes de la ejecución de estas. Por lo tanto, el primer paso que se realiza es instanciar un controlador de un navegador (por ejemplo, de Chrome o de Firefox). Luego, este controlador del navegador recibirá las distintas peticiones HTTP y las enviará al navegador real, siendo en este punto donde tiene lugar la interacción con los elementos de cualquier operación.

8.1.4. Pruebas de rendimiento

Las pruebas de rendimiento son las pruebas que se realizan para determinar lo rápido que un sistema realiza una tarea en condiciones particulares de trabajo. También puede servir para validar y verificar otros atributos de la calidad del sistema, tales como la escalabilidad, fiabilidad y uso de los recursos.

Pueden servir para diferentes propósitos. Por ejemplo, pueden demostrar que el sistema cumple los criterios de rendimiento. También, pueden comparar dos sistemas para encontrar cuál de ellos funciona mejor o pueden medir que partes del sistema o de carga de trabajo provocan que el conjunto rinda mal.

Se han realizado las pruebas de rendimiento con la ayuda de Locust.

A continuación se muestra un ejemplo de una prueba de rendimiento, en la que se han cargado hasta 100 usuarios concurrentes, añadiendo 5 cada segundo, y se han ejecutado de forma aleatoria las peticiones de obtener todos los perfiles y obtener el detalle de un perfil. Se ha decidido este escenario, ya que 100 usuarios lanzando peticiones contra la base de datos en el mismo instante de tiempo se considera un pico de usuarios en el sistema actual. No se prevé que tantos usuarios realicen una operación contra la base de datos en el mismo instante. Además, la configuración actual, debido a la limitación de recursos y su despliegue en una plataforma gratuita, admite como máximo 100 conexiones a la base de datos de forma concurrente. Consideramos que son suficiente para el sistema actual, pero en un entorno real de producción se debería desplegar la base de datos en un entorno que admita más conexiones.

En la siguiente imagen 8.1 podemos ver una tabla de resultado de las peticiones realizadas y

los tiempos que ha tardado en responder.

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)
POST	/api/login	100	1	140	180	147	105	259
GET	/api/profiles	2	0	313	340	326	313	338
GET	/api/profiles/3070	40	1	2000	5400	2301	112	5803
GET	/api/profiles/3071	32	3	1500	4800	1984	113	5266
GET	/api/profiles/3072	24	0	1800	5000	2091	115	5319
GET	/api/profiles/3073	19	0	1600	3900	1884	114	5807
GET	/api/profiles/3074	18	0	1700	4800	2295	114	5477
GET	/api/profiles/3075	12	0	2300	6300	2681	121	6776
GET	/api/profiles/3076	10	0	1100	5400	1942	113	5395
GET	/api/profiles/3077	8	0	800	3700	1660	143	3660
GET	/api/profiles/3078	7	0	2000	7000	2872	172	7035
GET	/api/profiles/3079	5	0	1800	3600	1906	205	3551
Aggregated		277	5	340	4200	1417	105	7035

Figura 8.1.: Estadísticas de las peticiones de las pruebas de carga

Podemos observar, que hay 5 fallos en total, ya que se deben a peticiones que se realizaron justo al llegar a los 100 usuarios concurrentes, y la base de datos rechazó dichas conexiones.

En las siguientes gráficas (figura 8.2 y 8.3) se puede observar una gráfica de este prueba de carga realizada.

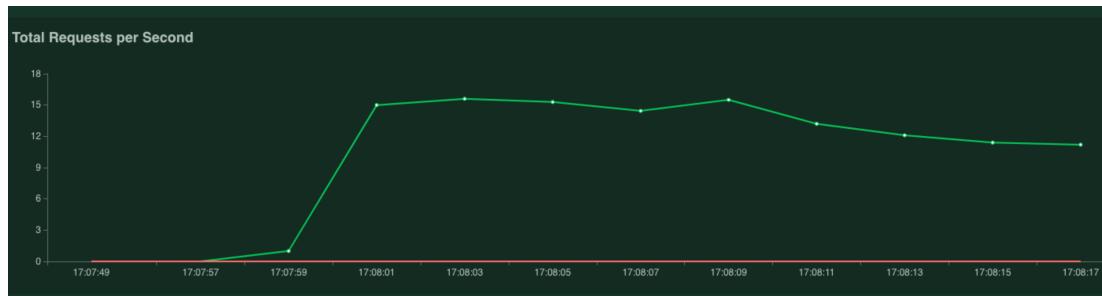


Figura 8.2.: Gráfica de las peticiones por segundo realizadas



Figura 8.3.: Gráficas del tiempo de respuesta y el número de usuarios en cada instante de tiempo

Podemos observar como cada vez que hay más usuarios concurrentes, el tiempo de respuesta va aumentando, algo totalmente normal, pero en ningún momento llega a valores que comprometan el rendimiento de la aplicación.

Capítulo 9

SOLUCIÓN IMPLEMENTADA

En este capítulo se expone la solución implementada y se explica el funcionamiento de cada una de las opciones que ofrece. El objetivo es que esta sección sirva de guía de uso y describa las características principales de la aplicación desarrollada. Se mostrará el funcionamiento normal del sistema, apoyado en capturas de la aplicación para ofrecer un soporte visual que facilite su comprensión.

9.1. Página de inicio

La primera toma de contacto con la aplicación tiene lugar con la página de inicio. En esta, se muestra, con el acompañamiento de unas ilustraciones, un breve resumen sobre la aplicación (el núcleo de esta, técnicas que emplea, etc.). Podemos ver el aspecto de esta interfaz en las figuras 9.1 y 9.2. También se puede apreciar que se ha proporcionado una barra de navegación para que el usuario tenga en un punto claro las distintas pantallas a las que puede navegar.

9.1. Página de inicio

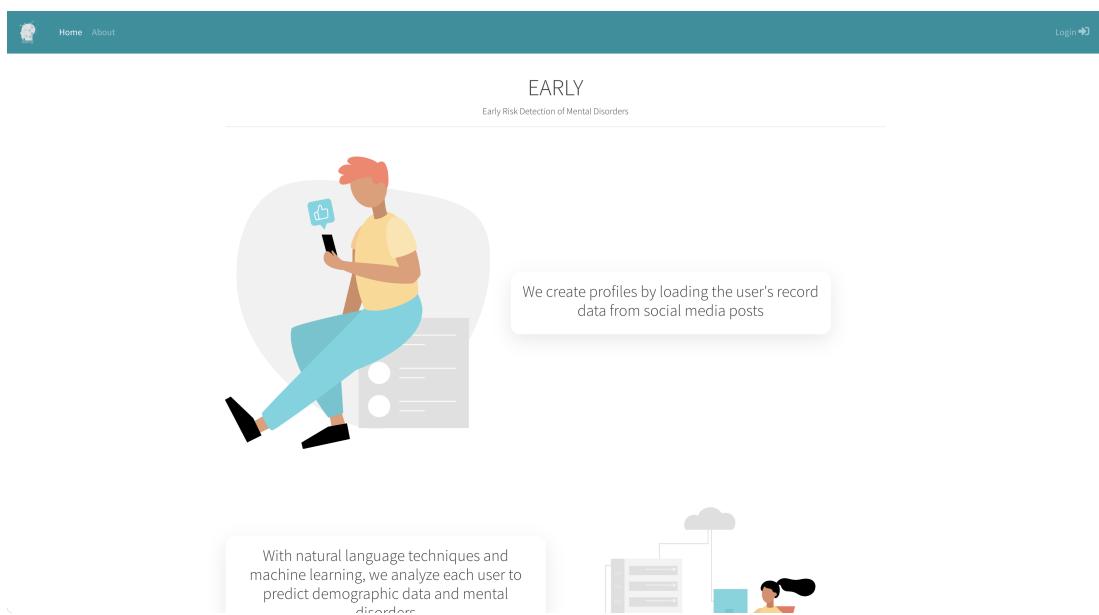


Figura 9.1.: Interfaz de inicio (primera parte)

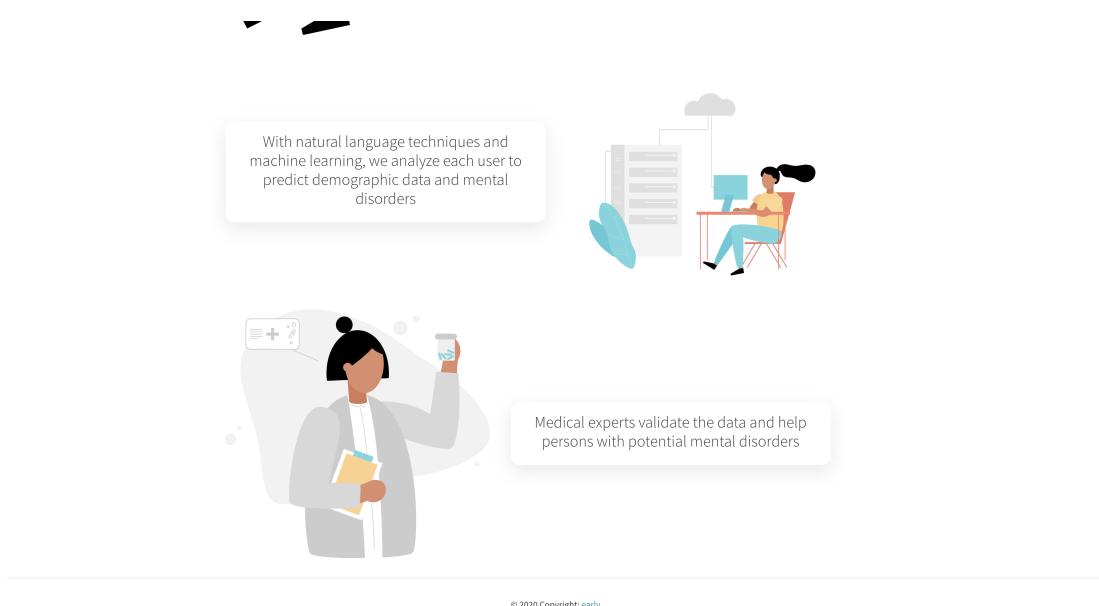


Figura 9.2.: Interfaz de inicio (segunda parte)

9.2. Funcionalidades de gestión de usuarios

9.2.1. Iniciar sesión

En la parte superior derecha de la barra de navegación, podemos acceder al inicio de sesión. Esta interfaz, con un diseño de interfaz minimalista, permite a un usuario acceder a más funcionalidades si proporciona su nombre de usuario y contraseña. En la figura 9.3 podemos ver el aspecto de esta interfaz. Se puede apreciar que existe un enlace para recuperar la contraseña, en la siguiente subsección explicaremos esta funcionalidad.

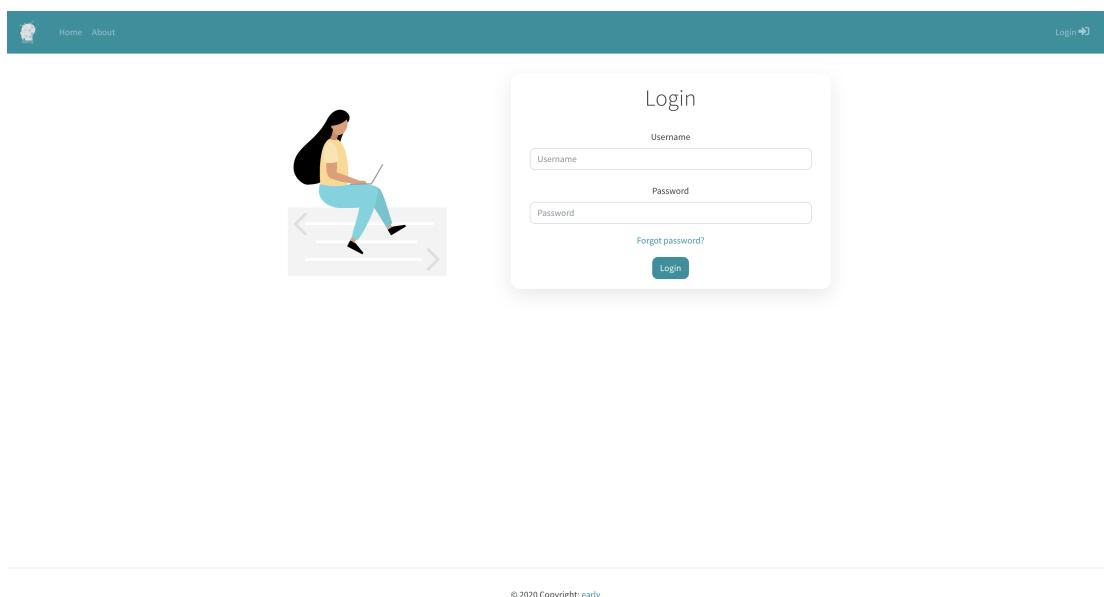


Figura 9.3.: Interfaz de login

9.2.2. Recordar contraseña

En el caso de que el usuario no recuerde su contraseña, podrá resetearla por medio de la siguiente interfaz (figura 9.4). Solamente necesita indicar su correo electrónico asociado, y ya se le proporcionará un enlace donde podrá crear una nueva.

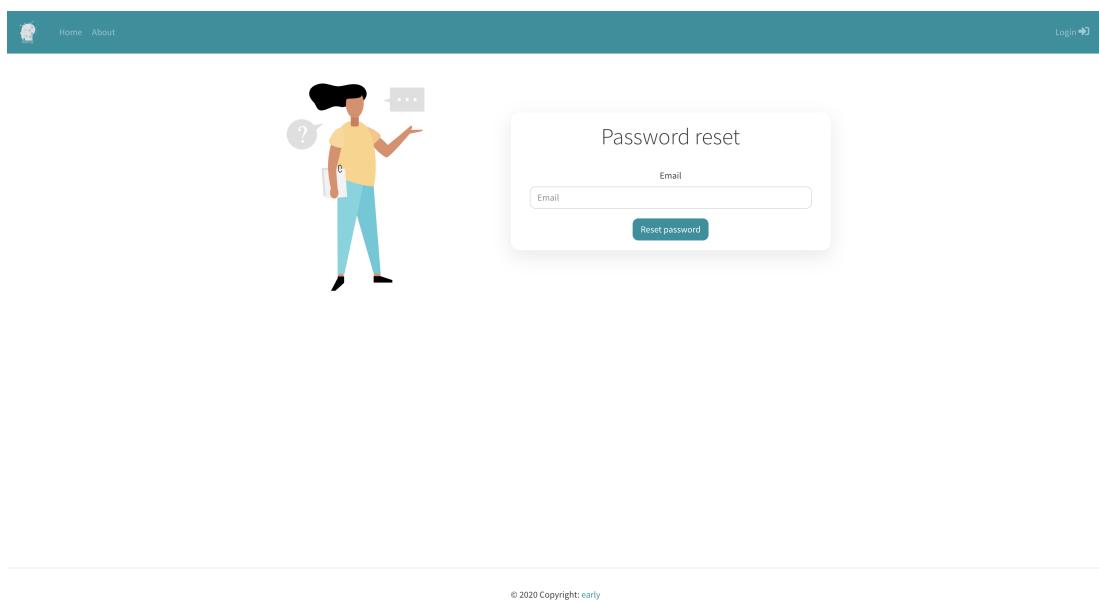


Figura 9.4.: Interfaz de recordar contraseña

9.2.3. Cerrar sesión

Una vez que el usuario haya iniciado sesión, puede cerrarla haciendo *click* en la parte superior derecha de la barra de navegación, y será redirigido a la página de inicio.

9.2.4. Cuenta

Una vez iniciada sesión, el usuario podrá acceder al panel de su cuenta de usuario para realizar acciones como ver y editar sus detalles, y cambiar su contraseña. El panel mencionado tiene el aspecto de la siguiente figura 9.5.

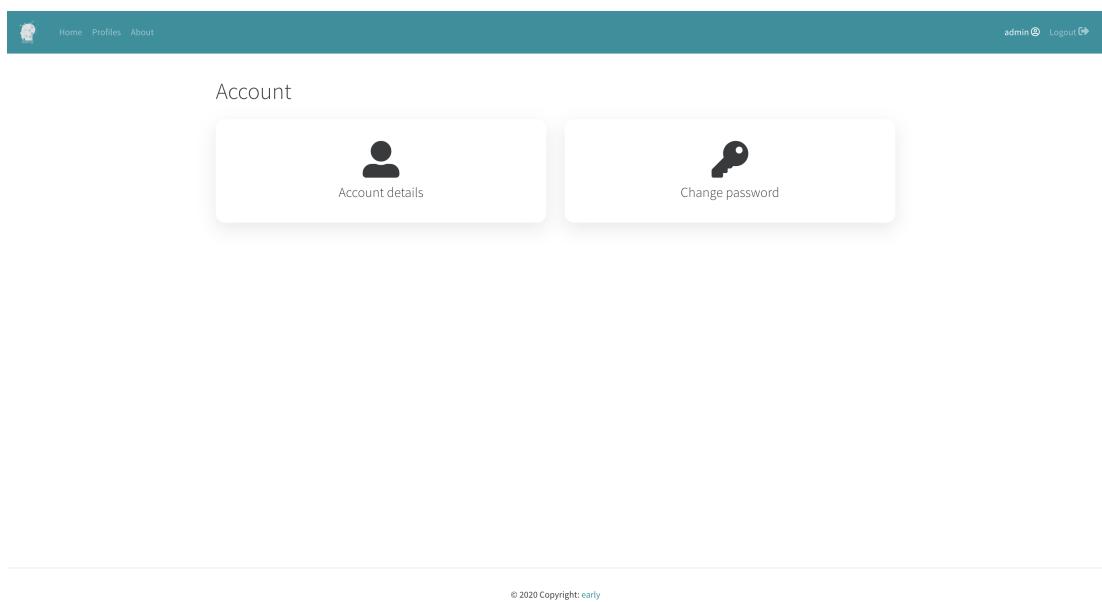


Figura 9.5.: Panel de la cuenta de usuario

9.2.4.1. Editar detalles de la cuenta

La primera de las acciones que permite el panel de la cuenta del usuario es editar o ver los detalles. En la figura 9.6 podemos ver el aspecto de esta interfaz y su similitud con el inicio de sesión, recuperar contraseña y cambiar contraseña. Se permite modificar el nombre, apellido y correo electrónico del usuario.

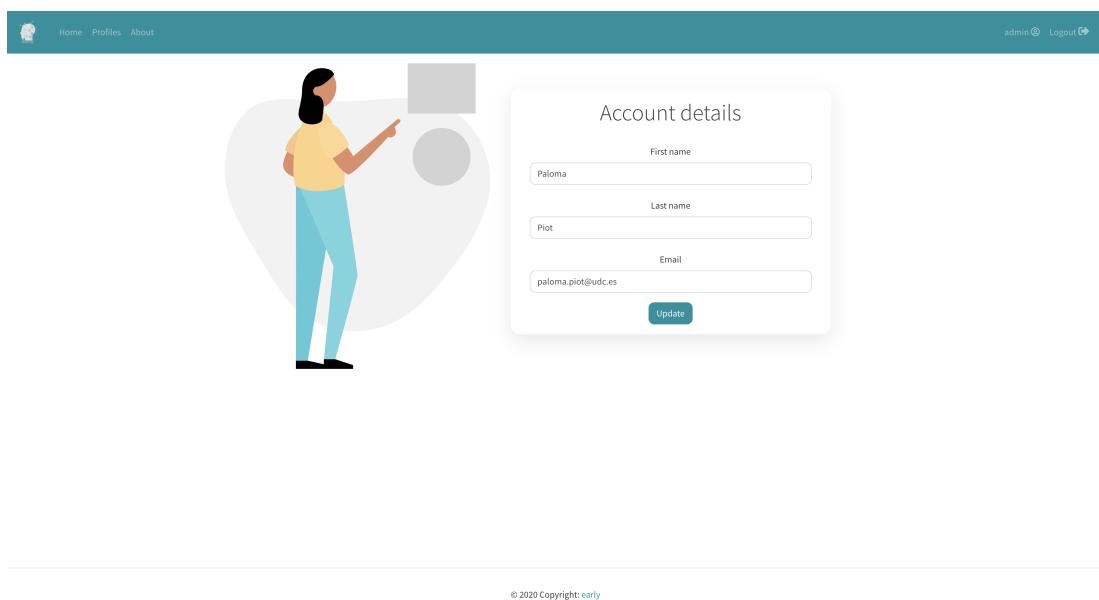


Figura 9.6.: Interfaz de actualización de los detalles de la cuenta

9.2.4.2. Cambiar contraseña

La otra acción de la gestión de la cuenta del usuario es cambiar la contraseña. Podemos ver en la siguiente figura 9.7 el aspecto de esta interfaz, donde se pide la contraseña actual y escribir dos veces la nueva, por motivos de seguridad.

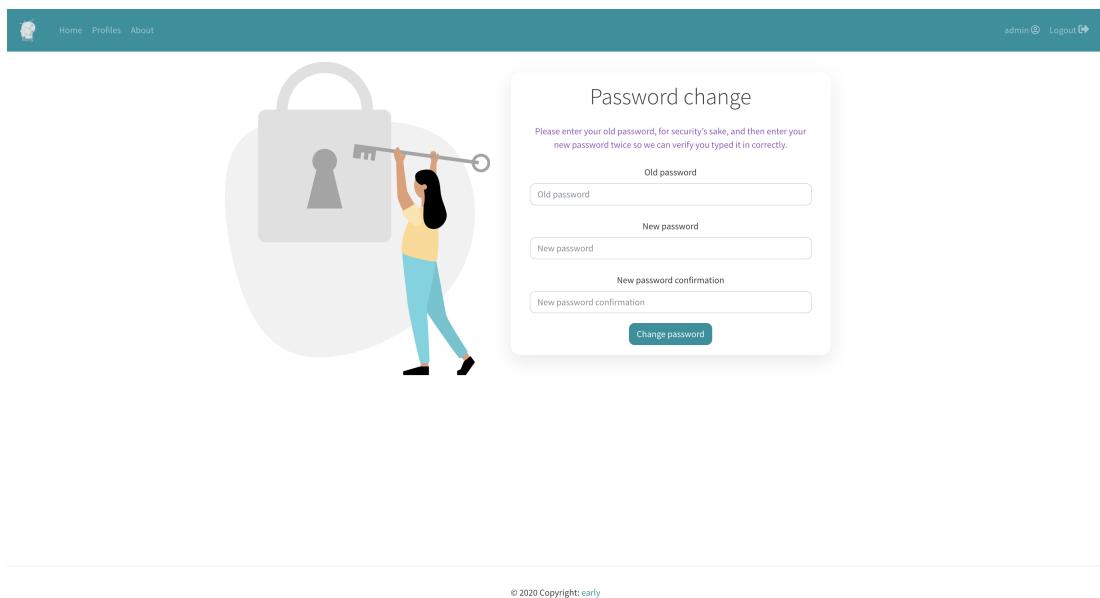


Figura 9.7.: Interfaz de cambio de la contraseña

9.3. Funcionalidades de gestión de perfiles

9.3.1. Ver perfiles

Para que un usuario pueda visualizar los perfiles, simplemente tiene que hacer *click* en el propio apartado de la barra de navegación y al entrar en esta página ya verá, de forma paginada, el conjunto de todos los perfiles, ordenados de forma alfabética. Cada perfil está contenido en una carta y muestra la información principal para que el usuario experto, de un vistazo rápido, pueda tener una ligera idea de su contenido. Además, se ha acompañado cada dato de un ícono, para que la asociación sea más rápida. También, para indicar si un usuario padece depresión o no, se han utilizado colores (rojo - depresión, azul - no depresión, violeta - no procesado). En la siguiente figura 9.8 podemos ver el aspecto de esta interfaz.

Figura 9.8.: Interfaz de perfiles

9.3.2. Buscar perfiles

En la misma interfaz que la anterior sección (ver perfiles), se puede realizar una búsqueda sobre estos. En la parte superior de la pantalla, se puede ver un *input* de texto, donde se permite buscar por el nombre de usuario del perfil (en su red social externa), y, si se despliega la flecha del buscador, se visualizará una búsqueda avanzada, donde se recuperarán aquellos perfiles que coincidan con los criterios de búsqueda establecidos.

Cada campo de filtrado funciona con los otros como una operación *AND*, mientras que si seleccionan varios valores de un mismo campo, se aplica la operación *OR*.

En la siguiente captura de pantalla 9.9, podemos ver los filtros establecidos y el resultado de la búsqueda realizada.

Figura 9.9.: Interfaz de perfiles con filtrado

9.3.3. Ver detalle perfil

Si hacemos *click* en la tarjeta de un perfil, accedemos a su detalle. En esta interfaz veremos exclusivamente sus datos. Además de los campos mostrados en el listado, se verán otros datos como corpus al que está asociado, razonamiento o motivo por el que un campo tiene un valor, y, si es el caso, quién validó el perfil.

En la interfaz de la figura 9.10, podemos ver el aspecto de esta pantalla. También podemos apreciar que se permite validar y editar los datos del perfil, y ver los detalles del formulario de depresión funcionalidades que se explicarán a continuación.

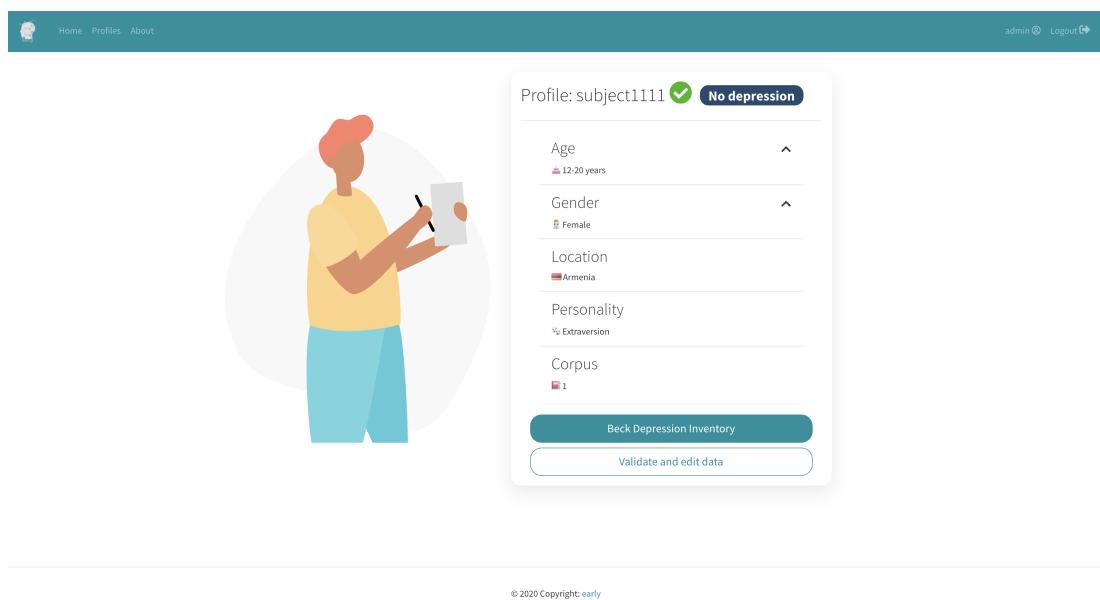


Figura 9.10.: Interfaz de detalle de un perfil

9.3.4. Validar perfil

Si accedemos al diálogo modal de validar y editar el perfil, permitimos al usuario corregir o añadir datos sobre el perfil. De esta forma tendremos una versión más correcta del usuario. Podemos cambiar su rango de edad, género, asociarlo a otros corpus, etc.

También podemos observar que, para el campo depresión, aparece un texto que indica el nivel de depresión que el sistema ha predicho. Este nivel se calcula a partir de la suma de las puntuaciones del inventario de depresión, existiendo cuatro posibles categorías o niveles de depresión.

En la figura 9.11 podemos ver la apariencia de este modal de edición.

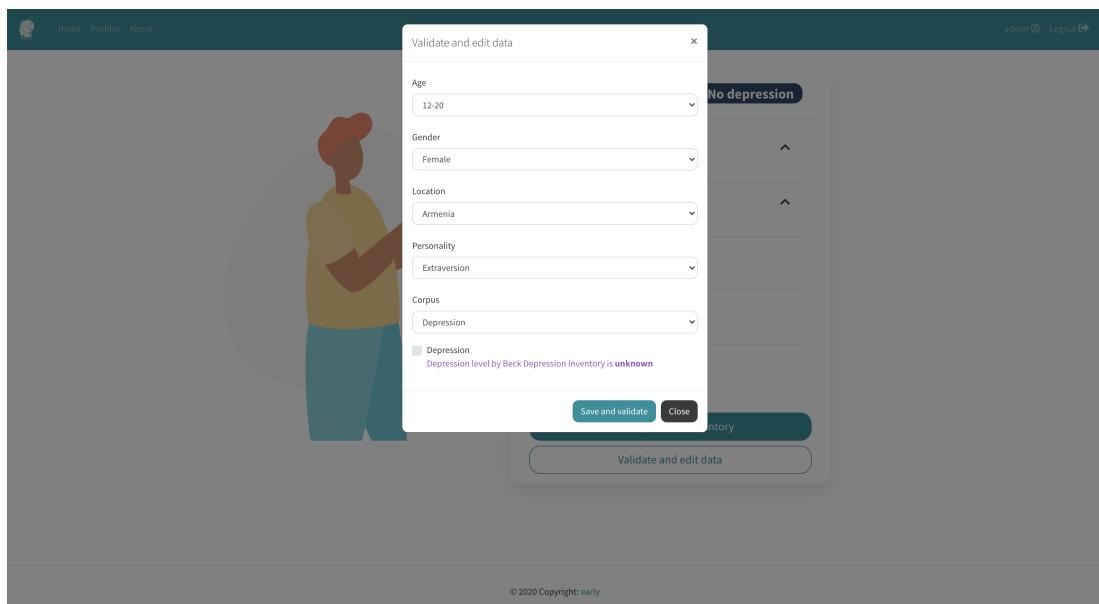


Figura 9.11.: Modal para validar y editar un perfil

9.3.5. Ver y corregir formulario de depresión

Para realizar una revisión del inventario de depresión de Beck (BDI) y sus respuestas autocompletadas, podemos simplemente hacer *click* en el botón que indica esto, y aparecerá un modal. Este modal representa las 21 preguntas del cuestionario, con la respuesta seleccionada que ha predicho el sistema, y el contexto por el cual el sistema ha marcado una respuesta u otra.

En la figura 9.12 podemos ver la apariencia de este modal.

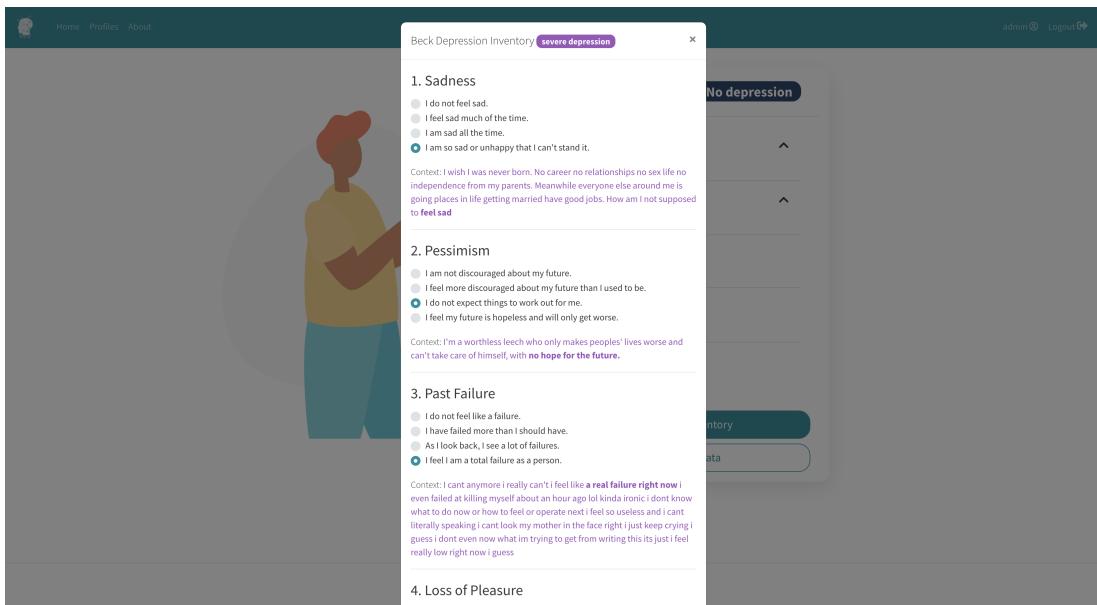


Figura 9.12.: Modal para ver y editar el cuestionario de depresión

9.4. Funcionalidades de gestión de datos

9.4.1. Panel de administración

En el caso de un usuario administrador, tendrá esta sección disponible en la interfaz de *home*. En ella, se disponen cuatro elementos con las funcionalidades principales que puede realizar: cargar datos de Reddit, exportar datos, crear corpus y procesar un usuario de Reddit.

En la siguiente captura de pantalla 9.13 podemos ver el panel del administrador y apreciar su similitud de diseño con el panel de la cuenta de usuario.

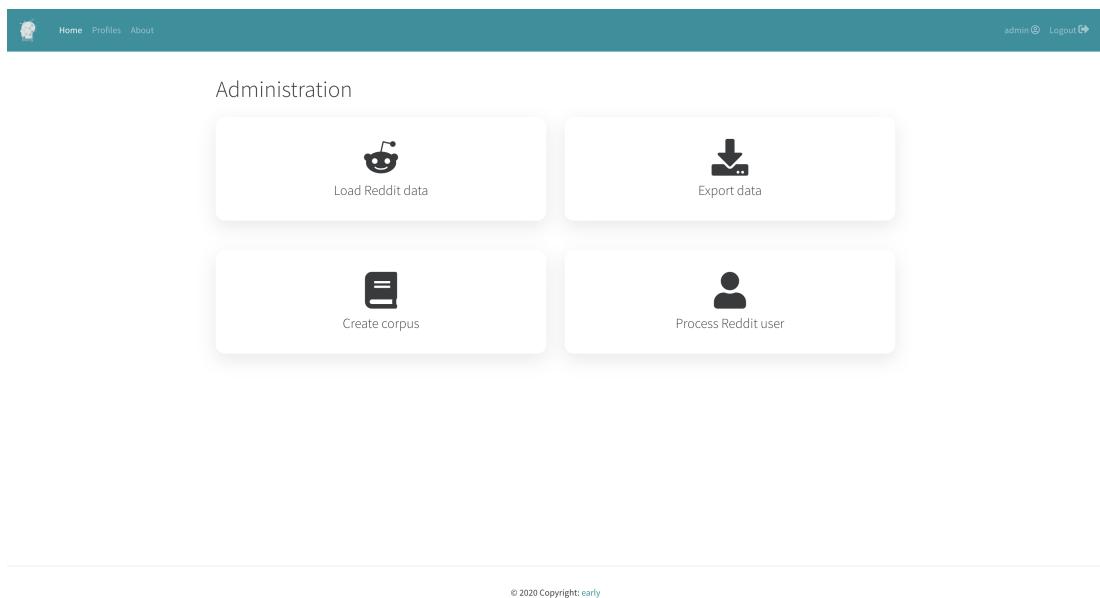


Figura 9.13.: Interfaz del panel de administración

9.4.2. Obtener datos de Reddit

Al hacer *click* en *Obtener datos de Reddit*, se abrirá un modal que nos permitirá configurar nuestro experimento indicando el subreddit del que obtener los datos, número de usuarios a obtener, en cuantos hilos buscar, comentarios por usuario y el corpus al que asociar los perfiles de este experimento. Podemos ver esta funcionalidad en la figura 9.14.

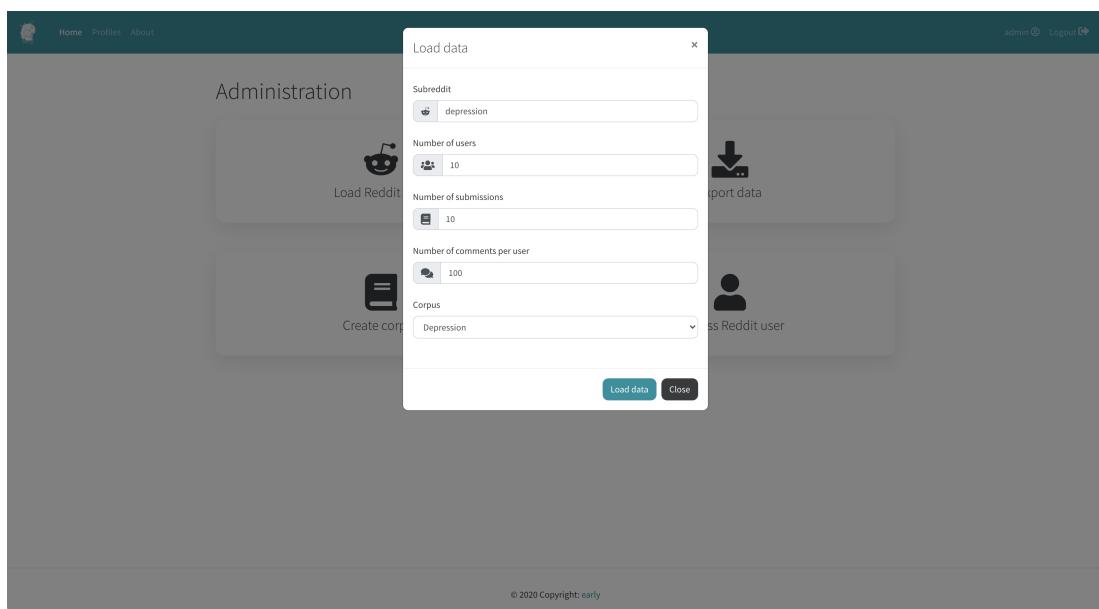


Figura 9.14.: Modal que permite configurar experimentos de carga de datos

9.4.3. Exportar datos

Esta funcionalidad también permite personalización por medio de un diálogo modal. Se permite exportar o bien los datos demográficos seleccionados, o bien el corpus seleccionado o bien datos etiquetados (corpus donde cada usuario tiene etiquetados sus datos demográficos). Se permite realizar la exportación tanto en formato JSON como CSV, y se permite indicar que etiquetas de datos demográficos exportar. Podemos ver el aspecto de este modal en la imagen 9.15.

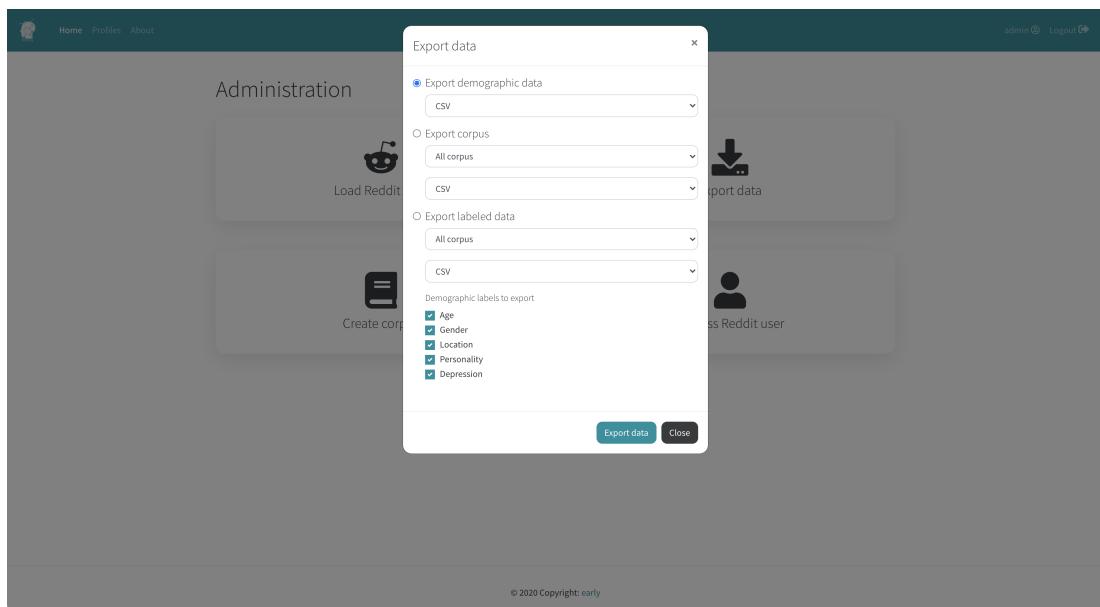


Figura 9.15.: Modal que permite personalizar la exportación de datos

9.4.4. Crear corpus

Para crear un corpus, simplemente debemos proporcionar un nombre para este. En la siguiente figura 9.16 podemos ver como, por medio de un modal, podemos realizar esta acción.

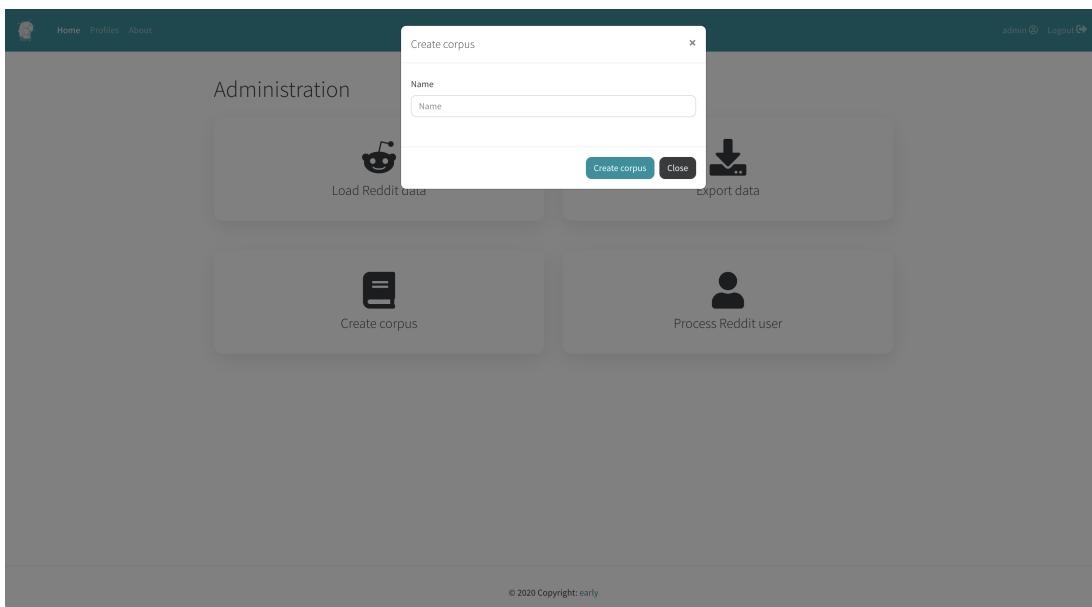


Figura 9.16.: Modal que permite crear un corpus

9.4.5. Procesar usuario de Reddit

Finalmente, permitimos procesar y perfilar a un usuario directamente (de forma independiente a la carga de datos que lanza el perfilado) indicando el nombre de este en Reddit, el número de comentarios que queremos obtener y el corpus al que añadirlo. Esta funcionalidad se ve reflejada en la captura de pantalla de la imagen 9.17.

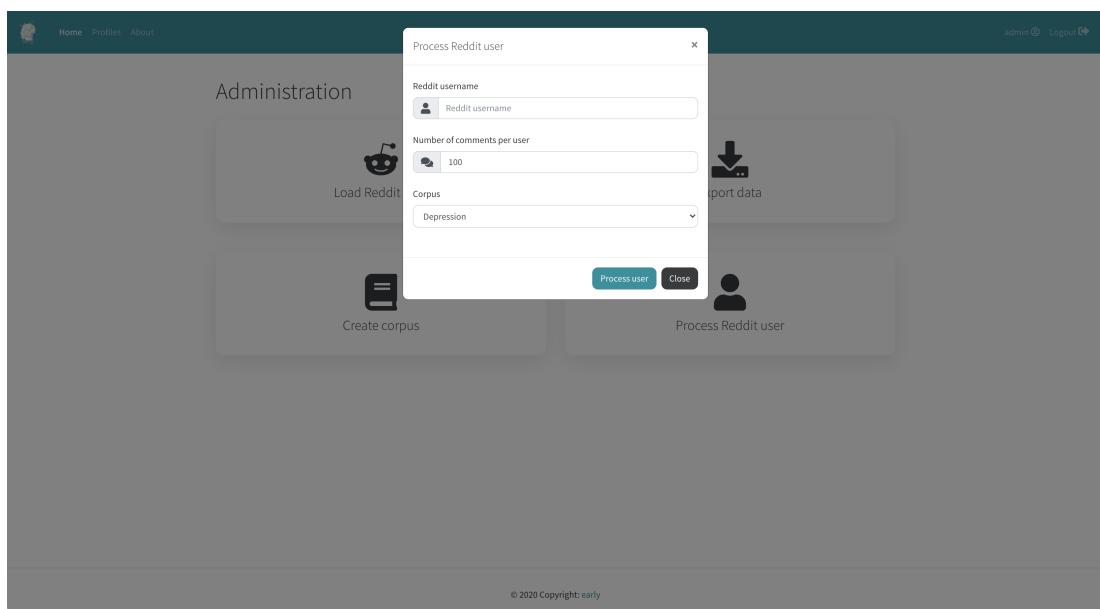


Figura 9.17.: Pantalla que permite procesar un usuario de Reddit

9.5. Interfaces en dispositivo móvil

A continuación se muestran una batería de interfaces para visualizar el aspecto de la aplicación en un dispositivo móvil.

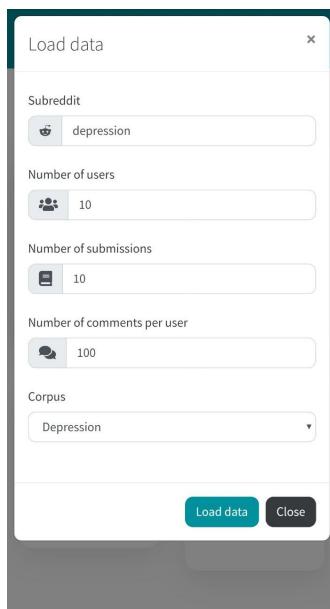


Figura 9.18.: Pantalla que permite cargar datos de Reddit

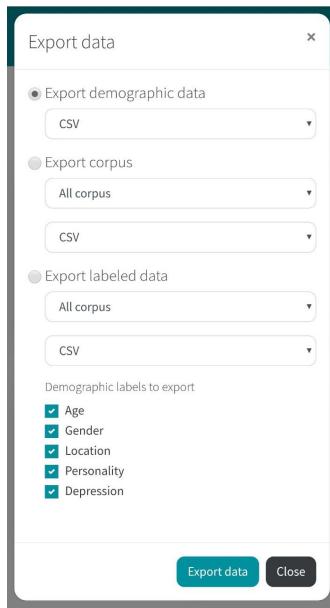


Figura 9.19.: Pantalla que permite exportar datos

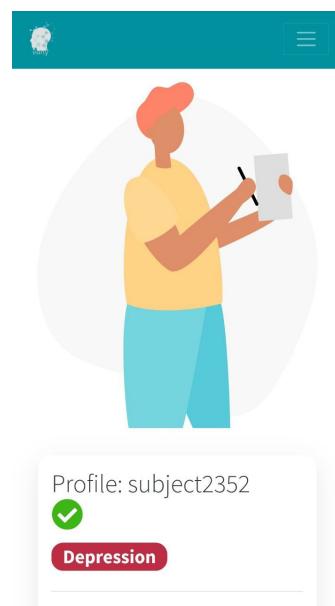


Figura 9.20.: Pantalla de detalle de un perfil

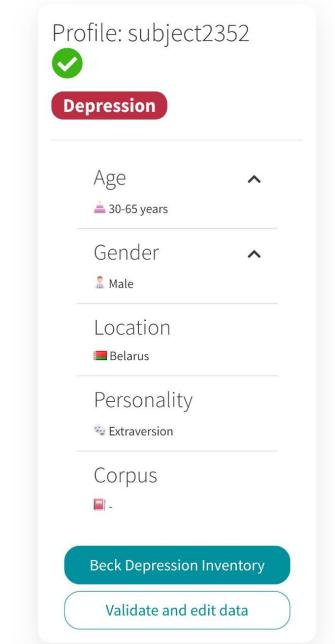


Figura 9.21.: Pantalla de detalle de un perfil

Beck Depression Inventory

severe depression

1. Sadness

- I do not feel sad.
- I feel sad much of the time.
- I am sad all the time.
- I am so sad or unhappy that I can't stand it.

Context: I wish I was never born. No career no relationships no sex life no independence from my parents. Meanwhile everyone else around me is going places in life getting married have good jobs. How am I not supposed to **feel sad**

2. Pessimism

- I am not discouraged about my future.
- I feel more discouraged about my future than I used to be.
- I do not expect things to work out for me.
- I feel my future is hopeless and will only get worse.

Context: I'm a worthless leech who only makes peoples' lives worse and can't take care of himself, with **no hope for the future**.

3. Past Failure

- I do not feel like a failure.
- I have failed more than I should have.

Figura 9.22.: Pantalla de detalle del cuestionario de depresión

Validate and edit data

Age: 12-20

Gender: Female

Location: Armenia

Personality: Extraversion

Corpus: Depression

Depression
Depression level by Beck Depression Inventory is **severe depression**

Save and validate **Close**

Figura 9.23.: Pantalla de edición de los datos de un perfil

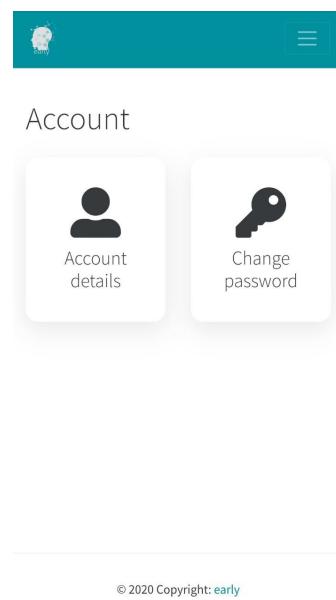


Figura 9.24.: Pantalla del menú de la cuenta del usuario

Capítulo 10

EVALUACIÓN

En este capítulo realizaremos una evaluación y un análisis del resultado de las distintas tareas realizadas en el presente trabajo. Se discutirán temas de rendimiento, la funcionalidad que abarca, los experimentos realizados y los resultados obtenidos, especialmente en lo tocante a las tareas de clasificación y predicción.

10.1. Tarea de clasificación de género

10.1.1. Funcionalidad

La funcionalidad tras la tarea de clasificación de género es etiquetar a un usuario o perfil en una de las dos categorías existentes para el género: masculino o femenino (*male* o *female*).

Para esto, el sistema, a partir de un conjunto de datos no estructurados (comentarios extraídos de redes sociales) deberá analizarlos y establecer esta etiqueta.

El estudio de esta tarea abarca dos fases. La primera donde se realiza un estudio de las *features* mencionadas en la sección 2.1, analizando su relevancia en los distintos modelos y una segunda fase donde se lanzan los experimentos con distintos modelos de aprendizaje y distintas configuraciones de las *features*.

Se puede observar el diagrama de trabajo en esta tarea en la siguiente figura 10.1.

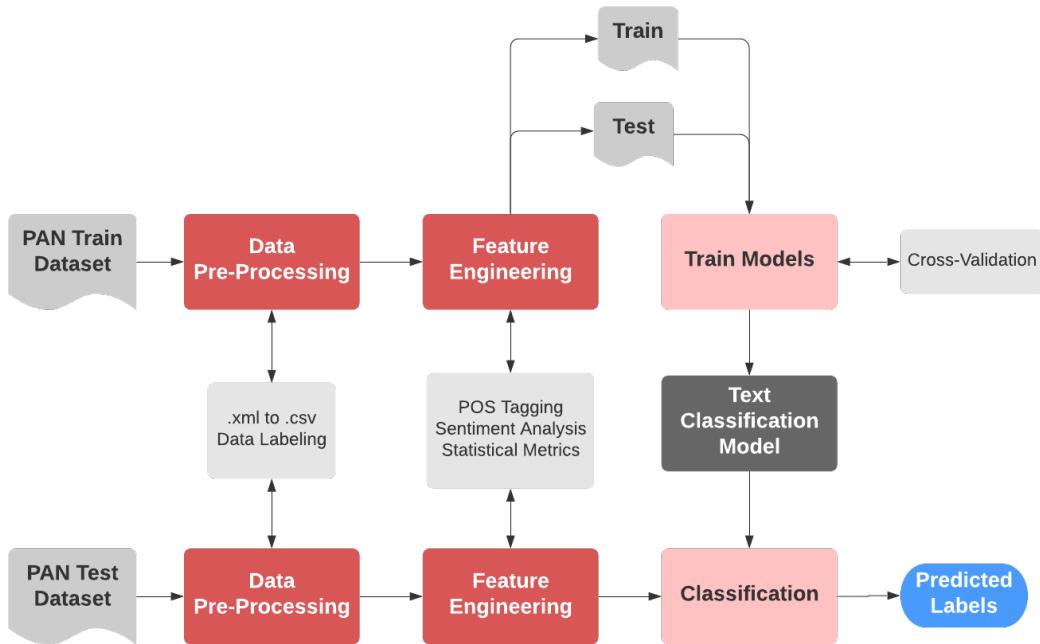


Figura 10.1.: Diagrama de la tarea de clasificación de género. *Publicado en Piot-Perez-Abadín et al. [1]*

10.1.2. Experimentos

Se examinaron distintos clasificadores y, teniendo en cuenta la naturaleza binaria de esta tarea, se han tenido en cuenta distintas aproximaciones utilizando los siguientes algoritmos. Para obtener el mejor valor, se ha realizado *10-fold cross-validation* y se han ajustado de forma manual los hiperparámetros.

10.1.2.1. k-fold cross-validation

Cross-validation [36] es una técnica que se utiliza para evaluar modelos predictivos, partiendo el *dataset* original en datos de entrenamiento y de prueba del modelo. El objetivo es tener un conjunto de datos para probar el modelo y demostrar que un modelo específico se adapta a un conjunto de datos independiente.

En **k-fold cross-validation**, el conjunto de datos original se partitiona de forma aleatoria en k subconjuntos del mismo tamaño.

En nuestro modelo, se realizaron diez pliegues de *cross-validation*, utilizando particiones diferentes y luego se tomó un promedio de los resultados. Se separaron los datos, dedicando el 0.8 para el entrenamiento y el 0.2 restante para pruebas. Esto es una separación habitual, ya que se dispone de un *dataset* exclusivo para la validación por lo que este conjunto de datos es exclusivo para el entrenamiento y las pruebas, siendo la validación sobre otro y, además, esta división sigue el principio de Pareto [37].

La ventaja de este método es que todas las observaciones se usan tanto para entrenamiento como validación y cada observación se usa para validar una vez exactamente.

Finalmente, se realizaron los experimentos con los siguientes clasificadores:

- *Random Forest*, 500 estimadores. Como se trata de una tarea sofisticada, se decidió experimentar con Random Forest para ser capaces de aprender con límites no lineales y tratar de obtener una precisión mayor que con un algoritmo lineal [38].
- *Adaptive Boosting*, estimador base Decision Tree, 500 estimadores, algoritmo SAMME, profundidad máxima 9. La idea tras los métodos de refuerzo es entrenar predictores de forma secuencial, cada uno intentando corregir los errores de su predecesor en cada iteración, para que, el siguiente clasificador, se construya basándose en el error de clasificación del anterior. Varios participantes de las competiciones de PAN han intentado AdaBoost en la tarea de reconocimiento de *bots*, por lo que se ha decidido probar el rendimiento de AdaBoost en esta tarea de clasificación de género [17].
- *LightGBM*, 500 iteraciones, profundidad máxima 7, tasa de aprendizaje 0.15, *boosting* *gbdt*, métrica binaria *logloss*, mínimo de datos en hoja 600, fracción de *bagging* 0.8, fracción de *feature* 0.8. Se ha decidido utilizar LightGBM, ya que es un *framework* de refuerzo gradiente que utiliza algoritmos de aprendizaje basados en árboles. Se centra en la precisión del resultado y es uno de los algoritmos que lidera las clasificaciones de las competiciones [39].

Teniendo en cuenta estos algoritmos, se han realizado distintas combinaciones de experimentos para encontrar el clasificador más efectivo en la tarea de perfilado de género. Se ha evaluado su precisión para saber cuál es el más adecuado.

Como se mencionó en la sección 2.1, se han utilizado los mencionados conjuntos de datos de PAN y se ha probado la precisión de nuestro modelo en el conjunto de datos de prueba de PAN de la competición “PAN Author Profiling 2019”.

Teniendo en cuenta la combinación de *features*, hemos probado los modelos:

1. Con todas las *features* sociolingüísticas sin la información de los *topic*.
2. Con todas las *features* iniciales, más los *topic*.
3. Eliminando las *features* con cero relevancia en el modelo.
4. Eliminando las *features* con menor relevancia en el modelo.
5. Incluyendo solo la mitad de *features* más importantes.

10.1.3. Resultado

En la siguiente tabla 10.1, se pueden ver los resultados obtenidos tras lanzar los experimentos mencionados en el apartado anterior.

Cuadro 10.1.: Precisión del clasificador de género

Model + features	Author	Celebrity	Author + Celebrity
All features and topics			
RandomForestClassifier	0.6030	0.5470	0.7174
AdaBoostClassifier	0.5538	0.5114	0.7470
LightGBM	0.5818	0.6121	0.7735
All features without topics			
RandomForestClassifier	0.6621	0.5174	0.6780
AdaBoostClassifier	0.6523	0.5197	0.6780
LightGBM	0.6598	0.5795	0.7152
Without less important features			
RandomForestClassifier	0.6720	0.5303	0.6985
AdaBoostClassifier	0.6598	0.5083	0.7258
LightGBM	0.6803	0.5068	0.7561
Top half important features			
RandomForestClassifier	0.6530	0.5523	0.6583
AdaBoostClassifier	0.6545	0.5871	0.6795
LightGBM	0.6538	0.5985	0.7008

Es importante resaltar que los experimentos realizados nos permiten realizar un análisis en cada *feature*, su relevancia y los efectos en el modelo. La siguiente gráfica 10.2 muestra la relevancia de cada *feature* en el modelo final. Podemos apreciar como los emojis de amor, los signos de exclamación, los emojis de afecto, y los artículos, junto con un *topic* son las *features* más importantes.

Esta métrica de relevancia nos muestra el número de veces que una *feature* se utiliza en el modelo, cuando más alto el número, más importancia tiene.

Otras *features* como la similitud del coseno, las referencias a uno mismo, la legibilidad, el uso de número, las interjecciones, y los adjetivos están entre la mitad de las *features* más relevantes.

Determinantes, emojis de mono, ratio de palabras, y la longitud media de las palabras, tiene un impacto bajo, pero, de todas formas, su contribución sigue siendo de relevancia, ya que, el clasificador empeora si estas se eliminan.

Por otro lado, hay *features* que son buenas candidatas para eliminar, como, por ejemplo, corchetes, media de número de líneas del texto o conjunciones, ya que no tienen un impacto en nuestro modelo. Si se eliminan todas las *features* con cero importancia, se mantiene la misma precisión en nuestro modelo.

Comparando los resultados, la mejor aproximación que se ha obtenido es utilizando el algoritmo LightGBM con los *topics* LDA y manteniendo todas las *features*. Se ha obtenido una precisión de **0.7735**, y este resultado es directamente comparable con los obtenidos en la tarea de la competición “PAN Author Profiling 2019”. En esta competición los tres mejores resultados rondan una precisión de **0.84**. A pesar de que nuestro resultado en precisión es inferior, supone una aportación novedosa a los clasificadores de género ya que se realizó una aproximación puramente lingüística, al contrario de los resultados que rondan esta cifra [16, 17, 18]. Estos, por lo general, han seguido una aproximación de *n-grams* y *word embeddings*. Por lo tanto, debido a las aproximaciones diferentes tomadas, existe esa diferencia de precisión, no obstante, como ya se ha mencionado, este trabajo supone una aportación beneficiosa [1].

En los gráficos 10.3 y 10.4 se puede ver el impacto de la salida del modelo resultante. Podemos ver que es posible conocer los efectos de cada *feature* basándonos en la categoría a predecir (“male” y “female” respectivamente).

Estas gráficas están formadas por todos los puntos de los datos de entrenamiento y demuestran:

- La importancia de las *features*, ya que las variables están clasificadas en orden descendente, de más importante a menos.

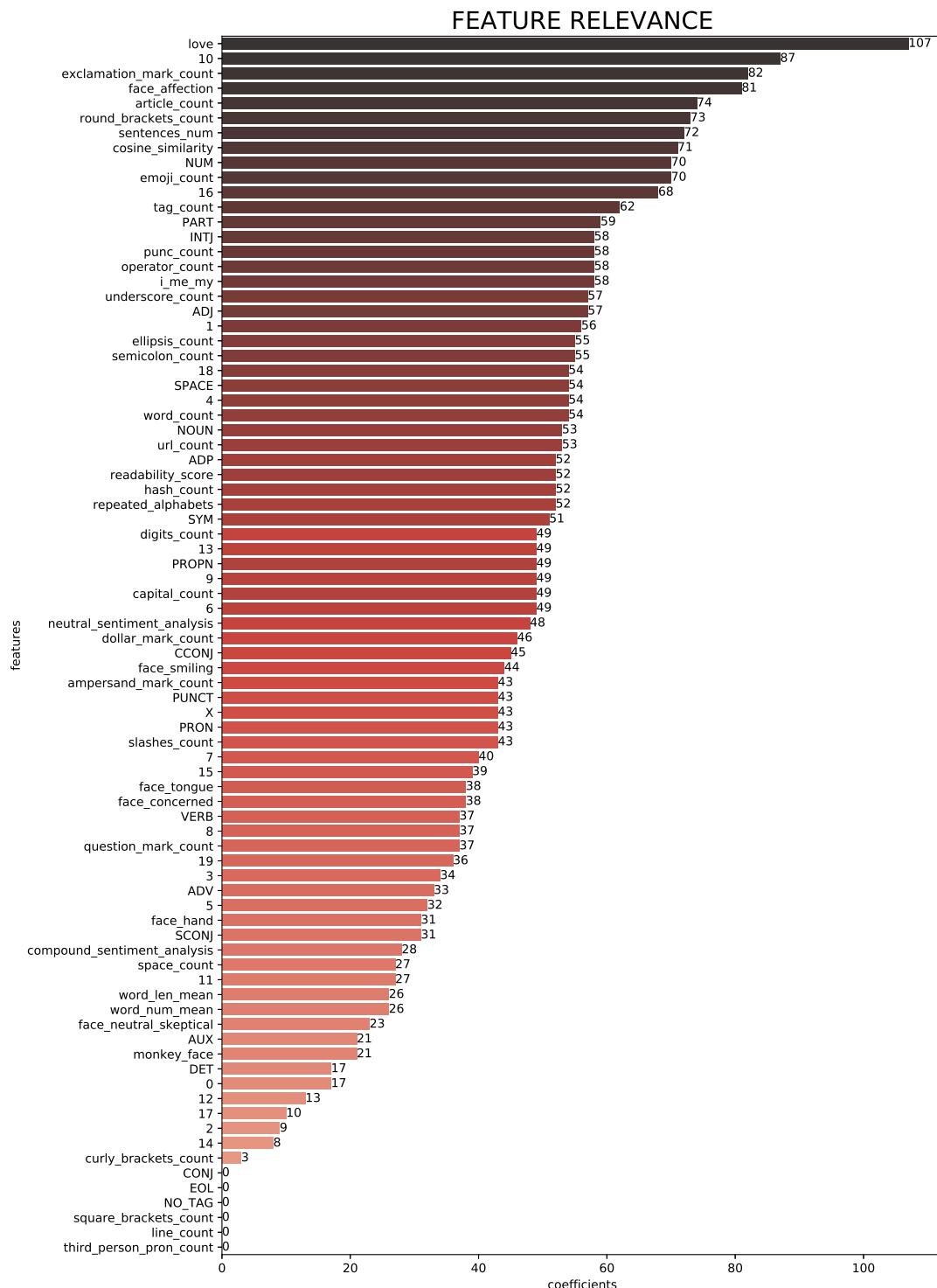


Figura 10.2.: Relevancia de las *features* en nuestro modelo. *Publicado en Piot-Perez-Abadin et al. [1]*

- El impacto. El eje X muestra si el efecto de ese valor está asociado con una predicción mayor o menor.
- Valor original. El color muestra si el valor de esa variable es alto (en rojo) o bajo (en azul) para esa observación.
- Correlación. Por ejemplo, un valor alto de “articles” tiene un impacto alto y positivo en la categoría “male”. El “alto” proviene del color rojo, y el impacto “positivo” se muestra en el eje X. De manera similar, diremos que la *feature* “love” está correlacionada negativamente con la variable objetivo.

En otras palabras, para los hombres, para la *feature* “articles”, cuanto mayor es el valor, el modelo tiende a clasificar los datos como “male”. Por otro lado, para la *feature* “love”, cuanto menor es el valor, el modelo tiene a clasificar el perfil como “male”.

Sabiendo esto, podemos ver que cuantos más artículos, *tags*, número de emojis en general y el particular emojis de cara sonriente y lengua, el número de frases, símbolos de puntuación, incluyendo puntos suspensivos, el número de determinantes y palabras más largas, el modelo tiende a clasificar el perfil como “male”. Esto confirma trabajos anteriores donde se había concluido que artículos, palabras largas, emojis de caras sonrientes son indicadores del género masculino [6]. El presente trabajo confirma esto y lo amplía incluyendo más *features* socio-lingüísticas.

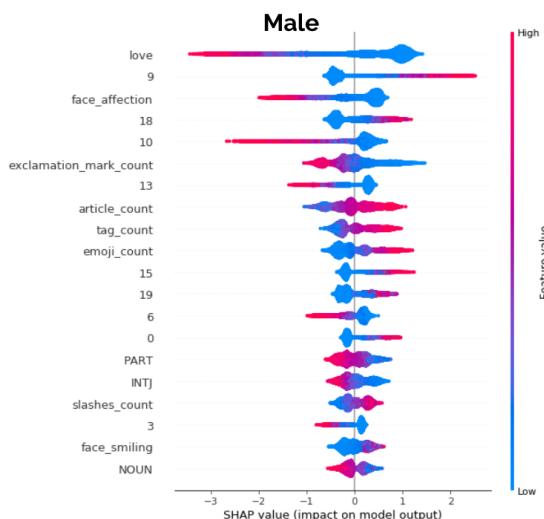


Figura 10.3.: Impacto de las *features* en la categoría de clasificación “male”.

En el caso de las mujeres, emojis de amor, cara afectiva, cara de preocupación y emojis de

mono, entre signos de exclamación, interjecciones, sustantivos, caracteres repetidos, pronomombres, URL, números de palabras y autoreferencias, son rasgos que indican que el perfil es “female”. Trabajos anteriores también han identificado que la secuencia de signos de exclamación, emojis de amor, caracteres repetidos y pronomombres son indicativos del género femenino [6]. Ahora nuestro trabajo también llega a esas conclusiones y ofrece una lista más extensa de características sociolingüísticas por género.

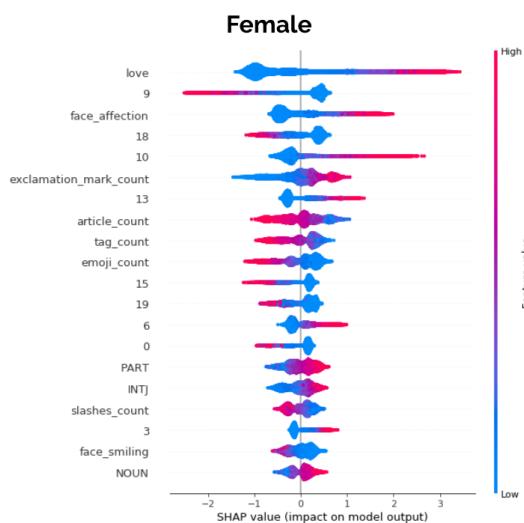


Figura 10.4.: Impacto de las *features* en la categoría de clasificación “female”.

Se puede remarcar que la combinación de los dos conjuntos de datos disponibles supone una precisión mayor, ya que el modelo es capaz de generalizar mejor. Por ello, el resultado con ambos *datasets* es ligeramente superior que el resto de experimentos.

10.2. Tarea de medición de la gravedad de los signos de depresión

10.2.1. Funcionalidad

En esta tarea se persigue dar una respuesta automática a cada pregunta del “Beck Depression Inventory”, para así conocer el nivel de depresión de un usuario de forma automática a partir de sus publicaciones en redes sociales.

Para lograr esto, nuestro sistema, a partir de un conjunto de datos no estructurados (comentarios extraídos de redes sociales) deberá analizarlos y establecer la respuesta más adecuada a cada pregunta.

En este trabajo se abarcan las fases de estudio de posibles soluciones para dar respuesta a esta problemática y la fase de experimentos, donde una vez escogida la solución, se diseñará el modelo que realice el trabajo.

10.2.2. Experimentos

Teniendo en cuenta la naturaleza de la tarea, en la cual se pretende de encontrar la mejor respuesta a un conjunto de preguntas entre un grupo de respuestas dado, se ha decidido realizar una aproximación de *Question Answering* (QA) [40]. QA es una disciplina que tiene como objetivo construir sistemas que, de forma automática, respondan preguntas planteadas por humanos. Sabiendo esto, esta aproximación es bastante acertada para nuestra tarea, ya que queremos dar respuesta a las preguntas del cuestionario, extrayendo esta información de la evidencia de las publicaciones en redes usuarios de la persona.

Para adecuar nuestra información e implementar esta aproximación hemos tenido que:

1. Reformular cada pregunta del cuestionario. Esto ha sido necesario, ya que, cada pregunta, viene formulada por una o varias palabras sobre el contexto de esta. Para poder aplicar esta técnica, hemos convertido contextos del estilo “Tristeza” a “¿Estás triste?”.
2. Aplicar el modelo para buscar la parte del texto de las publicaciones donde se responde a la pregunta.
3. Encontrar cuál de las respuestas de cada pregunta tiene un significado más parecido a la respuesta encontrada por el sistema en las publicaciones del usuario.

10.2.2.1. QA

Los modelos de *Question Answering* tienen como objetivo buscar la respuesta a una pregunta, dada esta y un fragmento de texto. Existen distintas aproximaciones, tanto supervisadas como no supervisadas, para obtener esto. Para poder realizar la primera, se necesitan muchos datos anotados para que, así, el modelo aprenda de ellos, y sepa determinar las respuestas.

SQuAD [41] es un conjunto de datos, que contiene miles de preguntas y fragmentos de texto

extraídos de artículos de Wikipedia. Ha sido entrenado con distintos temas como geología, el sistema inmune, los colegios privados, la crisis del petróleo de 1973, la normativa de la unión europea, etc.

Las características principales de SQuAD son:

- Es un *dataset* cerrado, donde la respuesta a una pregunta siempre forma parte del contexto (en la primera versión) y se encuentra de forma continua en el fragmento.
- El problema se simplifica buscando el índice de inicio y el de final para cada contexto.
- La mayor parte de las respuestas tienen una longitud igual o menor a cuatro palabras.

Se ha utilizado el dataset de SQuAD como base para construir nuestro modelo.

10.2.2.2. BERT

BERT (*Bidirectional Encoder Representations from Transformers*) es un modelo de *deep learning* basado en transformadores, donde cada elemento de salida está conectado a cada elemento de entrada, y los pesos entre ellos se calculan dinámicamente con base en su conexión [42]. BERT nos ofrece un mejor entendimiento de las palabras y frases en un contexto.

- *Bidirectional*: utiliza el contexto de la parte derecha e izquierda cuando analiza una palabra. Esto define el proceso de entrenamiento. Para predecir las siguientes palabras, BERT accede a todo el contexto.
- *Encoder Representations*: es un sistema de modelado del lenguaje, pre-entrenado con datos no etiquetados, al que se le puede aplicar *fine-tuning* (aplicar pequeños ajustes para obtener un resultado mejor).
- *from Transformers*: esto significa que está basado en un algoritmo NLP muy potente (*Transformers*), el cual define la arquitectura de BERT.

BERT es uno de los algoritmos para clasificación de texto más potentes hasta el momento, siendo los campos de tareas de QA y similitud de frases, donde más destaca.

Para tareas de *Question Answering*, el sistema recibe una pregunta y un fragmento de texto, y tiene que marcar la respuesta en este fragmento. Nuestra entrada de datos se procesa de la siguiente forma antes de ser procesada por el modelo:

- *Token embeddings*: se añade un token [CLS] a los tokens de entrada al principio de la pregunta y el token [SEP] se inserta al final de la pregunta y antes del fragmento de texto.
- *Segment embeddings*: es un marcador que indica a qué frase pertenece cada token. En la siguiente imagen 10.5 se puede ver que todos los tokens marcados como “A” pertenecen a la pregunta y los marcados como “B” al fragmento de texto.

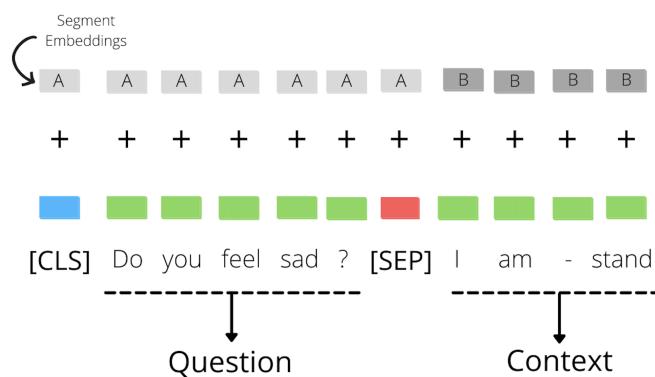


Figura 10.5.: *Segment embeddings* para la tarea de *Question Answering*

Para ajustar BERT y así construir un sistema de *Question Answering*, nuestro modelo se entrena con dos vectores a mayores, que marcan el inicio y el final de la respuesta. La probabilidad de que cada palabra sea el inicio de la respuesta se calcula como el producto escalar entre el *embedding* final de la palabra y el vector de inicio, seguido del *softmax* entre todas las palabras. La palabra con la probabilidad más alta, es la más conveniente. Y lo mismo ocurre para la palabra que marca el final de la respuesta.

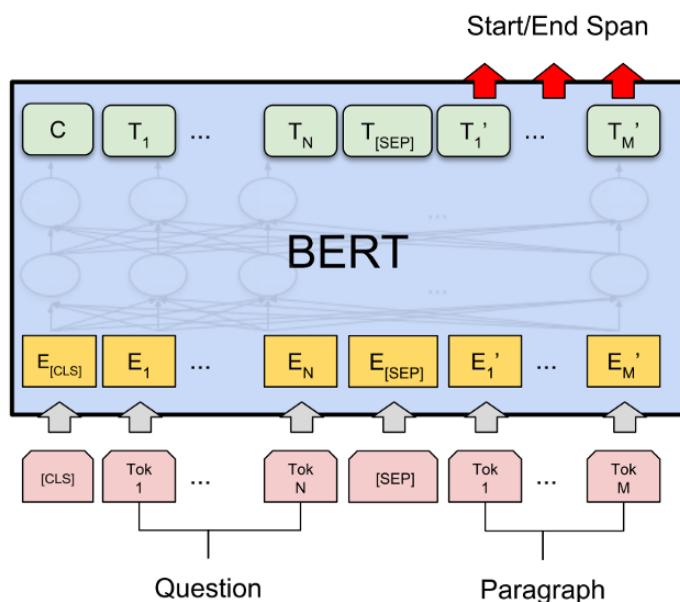


Figura 10.6.: Proceso de *fine-tuning* para BERT

10.2.2.3. Modelo implementado

Nuestro modelo ha utilizado el algoritmo de BERT aplicado en el conjunto de datos SQuAD (*Stanford Question Answering Dataset*).

Para ello, como entrada nuestro modelo tiene la pregunta y el fragmento de texto, separados por el token [SEP], y como salida obtendremos un *span* que contiene la respuesta. Esto se representa como una predicción del token que marca el inicio de la respuesta y el que marca el final. Sobre este modelo, se han realizado ajustes (*fine-tuning*) para obtener un resultado mejor. En la imagen 10.6, se puede ver el proceso interno de BERT para realizar esta tarea.

Aplicado a nuestra problemática, podemos ver, en la imagen 10.7, el resultado tras aplicar el modelo ajustado a una de las preguntas del inventario de Beck.

WHAT DO YOU THINK ABOUT YOUR FUTURE?

'Ever since I first got suicidal thoughts I know I don't see a point in living. Everyone has dreams they work towards, I have dreams I don't want to work towards. I have no doubt I could have a good life if I wanted but I why should I work for a future that has **no meaning** to me? Living is treated like such a status quo, nobody ever stops and asks themselves "Why shouldn't I just die?", you have to try and combat and bla bla bla. But why should I fight depression? Even normal life has hardships and those hardships make it not worth it. If I can't have life the way I want it then why should I have it at all? In a few months, I'll have final exams that I have to study for, if I want to continue living that is. I have the energy and I could absolutely do it since I came back to normal life. But every time I think about it, I get a thought "Why do you want to study for future if you don't see a point in it?" and then I do nothing I either get myself together or die, simple. If you have any answers beside "You'll have to find your own meaning in life" because I do have many ideas on who I could be. Problem is, I don't want to be anyone at all.'

Figura 10.7.: Ejemplo de pregunta, fragmento de texto y respuesta extraída

Una vez nuestro modelo de BERT nos devuelve ese texto como respuesta (“*no meaning*”, en el ejemplo de la figura), tenemos que encontrar, para esa pregunta (“*What do you think about your future?*”), reformulada para el contexto de Beck “*Pessimism*”), cuál es la respuesta del cuestionario que más se adecúa (“*0. I am not discouraged about my future*”, “*1. I feel more discouraged about my future than I used to be*”, “*2. I do not expect things to work out for me*”, “*3. I feel my future is hopeless and will only get worse*”).

Para realizar este proceso, hemos utilizado la API de Gensim para descargar vectores de palabras pre-entrenados, y así, para cada respuesta del formulario, ver qué respuesta extraída es la más similar.

Este proceso se realiza calculando la *Word Mover's Distance* (WMD) [43] entre los documentos (nuestra respuesta del texto y cada respuesta de la pregunta). Cuanto menor sea esta distancia, más similitud hay entre las respuestas. En la siguiente figura 10.8 podemos ver un ejemplo gráfico del funcionamiento de esta técnica.

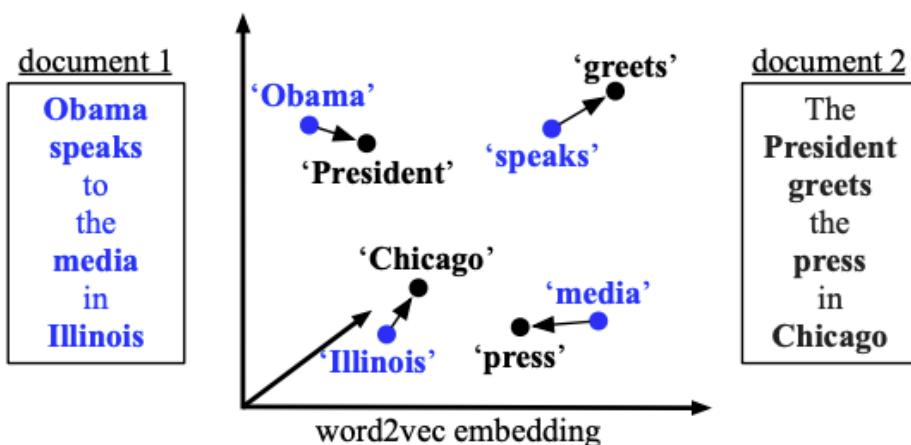


Figura 10.8.: Ejemplo de *Word Mover's Distance* entre dos documentos

10.2.3. Resultados

A continuación, se exponen los resultados obtenidos. Los resultados de las métricas explicadas en la sección 2.2 son directamente comparables con los obtenidos en la competición de eRisk de 2019 [23], ya que se ha utilizado el mismo *dataset* de validación.

Cuadro 10.2.: Resultados tarea de medición de la gravedad de los signos de depresión

AHR	ACR	ADODL	DCHR
27.85 %	70.63 %	76.58 %	25 %

Para realizar esta tarea, no se ha utilizado ningún conjunto de datos específicos del ámbito psicológico, sino que se ha delegado el trabajo al algoritmo y a los *datasets* ya entrenados, aplicando *fine-tuning* para obtener los mejores resultados posibles. Teniendo esto en cuenta, y la dificultad intrínseca de la tarea, se explicarán los resultados obtenidos.

Cuadro 10.3.: Resultados de la tarea con un modelo aleatorio

AHR	ACR	ADODL	DCHR
23.98 %	58.55 %	77.78 %	35.55 %

En términos de **AHR**, hemos logrado un 27.85 % de precisión, siendo los mejores resultados

de 2019 41.41 %, 40.71 % y 38.33 % y el resultado de un modelo aleatorio 23.08 %. Podemos decir que, se trata de una tarea muy compleja, donde existen bastantes factores a tener en cuenta como: que el usuario no haya expuesto la problemática de la pregunta en sus publicaciones, el sistema no sea capaz de encontrar la respuesta para el contexto, o no se acierte con la respuesta entre las cuatro opciones (siendo las diferencias entre las respuestas matices de cantidad o medida en la que afecta cada contexto). No obstante, se ha obtenido un resultado superior a un algoritmo aleatorio, por lo que podemos decir que nuestro modelo entiende el contexto y el análisis de las publicaciones es útil para extraer síntomas relacionados con la depresión.

La puntuación de la **ACR** muestra como se ha logrado obtener más de un 70 % (70.63 %), una puntuación muy cercana al máximo obtenido en 2019 (71.27 %) y lo que supone la segunda mejor marca entre todos los participantes. Esta métrica penaliza grandes distancias entre la respuesta correcta y la extraída por el sistema, por lo tanto, retomando el matiz de la complejidad de *mappear* la respuesta extraída con la respuesta correcta, podemos decir que el sistema hace un trabajo aceptable para encontrar, por lo general, el nivel de malestar que sufre un usuario en cada síntoma de depresión.

En lo tocante a la estimación del nivel de depresión, a pesar de obtener una **ADODL** del 76.58 %, nuestro modelo no es capaz de acertar el nivel exacto de depresión, ya que solo se logra un 25 % de **DCHR**, por lo que es posible mejorar en este campo, tal vez, utilizando una aproximación distinta en la que se maximice la precisión del nivel de depresión del usuario (*minimal depression, mild depression, moderate depression y severe depression*). Es decir, en vez de sumar los resultados de las preguntas individuales y ver a qué nivel de depresión corresponde, aplicar una estrategia supervisada, donde se entrenen colecciones de datos etiquetadas con un nivel de depresión y se intente predecir este nivel en datos nuevos.

En conclusión, estos experimentos son un indicativo de que es posible extraer de forma automática síntomas relacionados con la depresión a partir de publicaciones en redes sociales, aunque todavía se está lejos de obtener un sistema lo suficientemente válido para su uso por profesionales en el campo.

También es de interés conocer en qué contextos sobre la depresión es más común que los usuarios publiquen en sus redes sociales. Distintos subreddits indican que el suicidio es un signo del que muchos usuarios hablan en redes sociales, ya que existen hilos exclusivamente para tratar este tema. Por otro lado, es común que se compartan ideas sobre sentimientos de culpabilidad, tristeza o pesimismo sobre el futuro. Y siendo temas como el nerviosismo o concentración más difíciles de encontrar.

Capítulo 11

CONCLUSIONES

En el primer capítulo se presentó el objetivo de este proyecto, que era el *desarrollo de una plataforma para el diagnóstico psicológico a partir de análisis de textos*. En este capítulo final se realizará una evaluación crítica sobre el trabajo realizado y una propuesta de las líneas de mejora futuras que podrían servir de continuación del proyecto.

En la primera sección se explicarán las conclusiones del Trabajo de Fin de Máster una vez llevado a cabo y después se hablará sobre las líneas futuras del proyecto.

11.1. Trabajo realizado

Una vez terminado el proceso de desarrollo del trabajo se cumplieron los objetivos expuestos en la primera parte de esta memoria.

Se pusieron en práctica metodologías de trabajo propias del desarrollo ágil de software y, mediante ellas, se resolvió el problema planteado. Se creó, de esta forma, una plataforma para el diagnóstico psicológico a partir del análisis de textos. Como objetivos conseguidos consideramos:

- Se realiza un perfil demográfico automático a partir de comentarios o publicaciones de redes sociales.
- Se automatiza la tarea de carga de datos al consumir la API de Reddit, que desencadena la tarea del perfilado.

- Se permite la creación y exportación de colecciones de datos para que investigadores dispongan de *datasets* para sus trabajos.
- Se autocumplimenta el formulario de depresión para así estimar el nivel de depresión del usuario.
- Se ha implementado una plataforma web que permite a expertos corregir y validar los datos extraídos por el sistema.
- La plataforma web abarca todas las funcionalidades mencionadas en la sección de objetivos.

11.2. Lecciones aprendidas

A raíz del desarrollo de este proyecto se adquirieron conocimientos muy importantes en cuanto a las tecnologías mencionadas en el capítulo 3: se realizó una primera toma de contacto y se obtuvo un buen nivel con Django y Django REST Framework. También se utilizaron tecnologías como Celery y Redis, distintos *frameworks* de pruebas como Behave o Locust, de vital importancia para asegurar la calidad en nuestro sistema.

En lo tocante al procesado de lenguaje natural, se realizó una fase de investigación sobre las tareas de perfilado y se obtuvo un buen nivel en el uso de librerías como spaCy o scikit-learn. Al mismo tiempo, se trabajó con distintos algoritmos de aprendizaje automático y se evaluó el rendimiento y resultado de estos.

Los *frameworks* y tecnologías mencionadas se emplean ampliamente en los sistemas software actuales y, por lo tanto, son cada vez más importantes en el mercado laboral.

Se descubrieron los beneficios y las complicaciones del uso de otras tecnologías como OpenAPI y Docker, las cuales ya se habían utilizado con anterioridad, pero este trabajo sirvió para ampliar el conocimiento sobre ellas.

Por último, resaltar que el uso de GitLab y LaTeX, en un proyecto real, afianzó los conocimientos que se tenían sobre ellos. Sumando la implementación de un ciclo de CI/CD, de vital importancia en proyectos reales y lo que supone una gran ventaja a la otra de automatizar procesos

También se pusieron en práctica conocimientos sobre la planificación y los métodos de trabajo de proyectos de ingeniería informática. Mejorando el producto en gran medida gracias a en-

frentarse a los problemas que surgieron durante el desarrollo de este sistema. Por otro lado, se profundizó en la metodología Scrum, que permite trabajar de forma cómoda y rápida y que es tan importante en el desarrollo de software a día de hoy.

11.3. Líneas futuras

El sistema implementado cumple con los objetivos propuestos, pero como todo software desarrollado es natural que requiera evolucionar o ampliar las funcionalidades.

Las líneas futuras con respecto a las tareas de clasificación son:

- Realizar un estudio más exhaustivo sobre las *features* sociolingüísticas y su peso en el habla según distintos factores como el género y la edad. Esto sería una parte de investigación muy interesante, pero donde el conocimiento lingüístico es de vital importancia, y se necesitaría el apoyo de un experto en esta materia.
- Incluir en el perfilado más campos demográficos como edad, personalidad, nivel económico, nivel de estudios, etc.
- Realizar los entrenamientos con más conjuntos de datos, lo más diversos posibles, para que los modelos generalicen lo máximo posible y ofrezcan mejores resultados.
- Probar distintas aproximaciones sobre los modelos (ej. *n-grams* en la tarea de clasificación de género).
- Llevar a cabo distintas aproximaciones para la tarea de auto completar el formulario de depresión, e intentar obtener mejores resultados, sobre todo en lo tocante a la estimación del nivel de depresión.

En cuanto a trabajo futuro con respecto a la aplicación web, existen múltiples posibilidades, entre las que se destacan:

- Mejorar el sistema de despliegado de la aplicación, para que la infraestructura en la que esté ejecutando tenga los recursos suficientes para un sistema en producción. Actualmente la aplicación está desplegada y se puede utilizar, pero con limitaciones, ya que se están utilizando servidores gratuitos.
 - Asociar los perfiles a un grupo, para que solo sean visibles por un conjunto de expertos. Estaría bien que los perfiles estuvieran agrupados por centro o por entidad de expertos,
-

para que solo estos pudieran acceder a ellos y modificarlos. Ahora mismo cualquier usuario experto puede ver cualquier perfil disponible en la aplicación.

- Permitir integrar la historia médica de un paciente y que, tras indicar su nombre de usuario en redes sociales, se cargue su historial de publicaciones y así predecir su nivel de depresión.
- Recuperar datos de Twitter. Actualmente se ha escogido la red social de Reddit para crear colecciones, pero sería interesante la integración con Twitter o con otras redes sociales en el futuro.
- Incluir más idiomas en la internacionalización. En este momento la aplicación está disponible en inglés y español, pero podría ampliarse a más idiomas para así llegar a más público.

Apéndice A

Glosario de términos

NLP (*Natural Language Processing*). Es un subcampo de la lingüística, informática e inteligencia artificial que se ocupa de las interacciones entre las computadoras y el lenguaje humano. En particular, cómo programar las computadoras para procesar y analizar grandes cantidades de datos del lenguaje natural.

ML (*Machine Learning*). Es un método de análisis de datos que automatiza la construcción de modelos analíticos.

POS Tagging (*Part-of Speech Tagging*). En lingüística, es el proceso de marcar una palabra en un texto como correspondiente a una parte particular del discurso, basado tanto en su definición como en su contexto.

NER (*Named-entity recognition*). Es una subtarea de extracción de información que busca ubicar y clasificar entidades de un texto no estructurado en categorías predefinidas como nombres propios, organizaciones, cantidades, valores monetarios, etc.

SVM (*Support Vector Machine*). Modelo de aprendizaje automático supervisado que utiliza algoritmos de clasificación para problemas de clasificación de dos grupos (binarios).

DCNN (*Deep Convolutional Neural Network*). Red neuronal que consta de muchas capas de redes neuronales.

CI/CD (*Continuous Integration/Continuous Delivery*). Método para entregar aplicaciones frecuentemente mediante la automatización en las etapas de desarrollo de la aplicación. Suele incluir integración continua, entrega continua e implementación continua.

HTTP (*Hypertext Transfer Protocol*). Es un protocolo de capa de aplicación para sistemas de

información distribuidos, colaborativos e hipermedia.

REST (*Representational State Transfer*). Estilo arquitectónico de software que utiliza un subconjunto de HTTP.

API (*Application Programming Interface*). Software intermediario que permite que dos aplicaciones se comuniquen entre sí.

URL (*Uniform Resource Locator*). Dirección de un recurso único en la web.

HTML (*Hypertext Markup Language*). Es el lenguaje de marcado estándar para documentos, diseñado para mostrarse en un navegador web.

BDD (*Behaviour-Driven Development*). Es un proceso ágil que consiste en definir el comportamiento de una funcionalidad a través de ejemplos en texto plano.

TDD (*Test-Driven Development*). Es un proceso de desarrollo de software donde se convierten los requisitos en casos de prueba antes de que se desarrolle el software.

QA (*Quality Assurance*). Es un proceso que asegura que una organización entregue los mejores productos o servicios posibles.

E2E (*End-to-End*). En *testing* de software, es un método de pruebas que implica probar el flujo de trabajo de una aplicación de principio a fin.

VM (*Virtual Machine*). Una máquina virtual es la emulación del sistema de un ordenador.

LDA (*Latent Dirichlet Allocation*). En NLP, LDA, es un modelo estadístico generativo que permite explicar conjuntos de observaciones por grupos no observados que explican por qué algunas partes de los datos son similares.

Corpus. En lingüística, un corpus es una colección de datos lingüísticos, ya sean textos escritos, como una transcripción de un discurso grabado.

Corpora. En lingüística, corpora es una colección de textos, completa y autónoma.

Lexicon. Vocabulario de una persona, lenguaje o rama de conocimiento.

Documento. Una unidad de recuperación, generalmente texto. Puede ser un párrafo, una página web, un artículo, un libro completo, etc.

Documentos. Conjunto de documentos.

QA (*Question Answering*). Un sistema QA es una forma de recuperación de información que debe ser capaz de recuperar respuestas a preguntas planteadas en lenguaje natural.

Bibliografía

- [1] P. Piot-Perez-Abadin., P. Martin-Rodilla., and J. Parapar., “Experimental analysis of the relevance of features and effects on gender classification models for social media author profiling,” in *Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE*, pp. 103–113, INSTICC, SciTePress, 2021.
- [2] OMS, “Organización mundial de la salud: Depresión.” <https://www.who.int/es/news-room/fact-sheets/detail/depression>, 2020.
Accedido el 16/02/2020.
- [3] UNICEF, “Responding to the mental health and psychosocial impact of covid-19 on children and families.” <https://www.unicef.org/media/83951/file/MHPSS-UNICEF-Learning-brief.pdf>, 2020.
Accedido el 17/12/2020.
- [4] PAN, “Pan: series of scientific events and shared tasks on digital text forensics and stylometry.” <https://pan.webis.de/shared-tasks.html>, 2020.
Accedido el 02/02/2020.
- [5] D. E. Losada, F. Crestani, and J. Parapar, “Early detection of risks on the internet: An exploratory campaign,” in *Advances in Information Retrieval - 41st European Conference on IR Research, ECIR 2019, Cologne, Germany, April 14-18, 2019, Proceedings, Part II* (L. Azzopardi, B. Stein, N. Fuhr, P. Mayr, C. Hauff, and D. Hiemstra, eds.), vol. 11438 of *Lecture Notes in Computer Science*, pp. 259–266, Springer, 2019.
- [6] H. A. Schwartz, J. C. Eichstaedt, M. L. Kern, L. Dziurzynski, S. M. Ramones, M. Agrawal, A. Shah, M. Kosinski, D. Stillwell, M. E. Seligman, and L. H. Ungar, “Personality, Gender, and Age in the Language of Social Media: The Open-Vocabulary Approach,” *PLoS ONE*, 2013.

- [7] A. Sboev, T. Litvinova, D. Gudovskikh, R. Rybka, and I. Moloshnikov, “Machine Learning Models of Text Categorization by Author Gender Using Topic-independent Features,” in *Procedia Computer Science*, 2016.
- [8] B. Liu, “Sentiment analysis and opinion mining,” *Synthesis Lectures on Human Language Technologies*, 2012.
- [9] J. Coates, *Women, men and language: A sociolinguistic account of gender differences in language, third edition*. Routledge, 2015.
- [10] G. Jackson-Koku, “Beck depression inventory,” 2016.
- [11] F. Orlandi, J. Breslin, and A. Passant, “Aggregated, interoperable and multi-domain user profiles for the social web,” in *ACM International Conference Proceeding Series*, 2012.
- [12] OMS, “Gender and women’s mental health.” https://www.who.int/mental_health/prevention/genderwomen/en/, 2020.
Accedido el 18/07/2020.
- [13] Albert, Paul R., “Why is depression more prevalent in women?,” 2015.
- [14] e. a. Czeisler MÉ; Lane RI; Petrosky E, “Mental health, substance use, and suicidal ideation during the covid-19 pandemic.,” June 24–30, 2020.
MMWR Morb Mortal Wkly Rep 2020;69:1049–1057. DOI: <http://dx.doi.org/10.15585/mmwr.mm6932a1externalicon>.
- [15] Asociación Americana de Psiquiatría, “Manual diagnóstico y estadístico de trastornos mentales (4^a ed.),” 2000.
- [16] A. I. Valencia, H. G. Adorno, C. S. Rhodes, and G. F. Pineda, “Bots and gender identification based on stylometry of tweet minimal structure and n-grams model notebook for PAN at CLEF 2019,” in *CEUR Workshop Proceedings*, 2019.
- [17] A. Bacciu, M. L. Morgia, A. Mei, E. N. Nemmi, V. Neri, and J. Stefa, “Bot and gender detection of twitter accounts using distortion and LSA notebook for PAN at CLEF 2019,” *CEUR Workshop Proceedings*, vol. 2380, no. July, 2019.
- [18] D. Y. Espinosa, H. Gómez-Adorno, and G. Sidorov, “Bots and gender profiling using character bigrams notebook for PAN at CLEF 2019,” in *CEUR Workshop Proceedings*, 2019.

- [19] Y. Joo and I. Hwang, “Author profiling on social media: An ensemble learning model using various features notebook for PAN at CLEF 2019,” in *CEUR Workshop Proceedings*, 2019.
 - [20] E. R. D. Weren, A. U. Kauer, L. Mizusaki, V. P. Moreira, J. P. M. D. Oliveira, and L. K. Wives, “Examining Multiple Features for Author Profiling,” *Journal of Information and Data Management*, 2014.
 - [21] C. Llamas, “Sociolinguistics,” in *The Routledge Handbook of Applied Linguistics*, Routledge, 2011.
 - [22] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent Dirichlet allocation,” *Journal of Machine Learning Research*, 2003.
 - [23] D. E. Losada, F. Crestani, and J. Parapar, “Overview of eRisk at CLEF 2019 Early Risk Prediction on the Internet (extended overview),” in *CEUR Workshop Proceedings*, 2019.
 - [24] A. T. Beck, C. H. Ward, M. Mendelson, J. Mock, and J. Erbaugh, “An Inventory for Measuring Depression,” *Archives of General Psychiatry*, 1961.
 - [25] D. E. Losada, F. Crestani, and J. Parapar, “Overview of eRisk 2020: Early Risk Prediction on the Internet,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2020.
 - [26] S. G. Burdisso, M. Errecalde, and M. Montes-Y-Gómez, “UNSL at Erisk 2019: A Unified Approach for Anorexia, Self-harm and Depression Detection in Social Media,” in *CEUR Workshop Proceedings*, 2019.
 - [27] K. Schwaber and J. Sutherland, “The Scrum Guide,” in *Software in 30 Days*, Wiley, 2015.
 - [28] S. W. Ambler, “User Stories: An Agile Introduction,” *AgileModeling*, 2013.
 - [29] Atlassian, “Gitflow Workflow — Atlassian Git Tutorial,” 2017.
 - [30] GitLab, “GitLab Continuous Integration & Deployment,” 2018.
 - [31] C. Solís and X. Wang, “A study of the characteristics of behaviour driven development,” in *Proceedings - 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2011*, 2011.
 - [32] G. Miller, “The magical number seven plus minus two,” *Psych. Rev.*, 1956.
 - [33] S. E. Palmer, “Common region: A new principle of perceptual grouping,” *Cognitive Psychology*, 1992.
-

- [34] P. M. Fitts, “The information capacity of the human motor system in controlling the amplitude of movement,” *Journal of Experimental Psychology*, 1954.
- [35] NNgroup, “Jakob’s Law of Internet User Experience.,” 2017.
- [36] J. Brownlee, “A Gentle Introduction to k-fold Cross-Validation,” 2019.
- [37] W. van der Aalst., “On the pareto principle in process mining, task mining, and robotic process automation,” in *Proceedings of the 9th International Conference on Data Science, Technology and Applications - DATA*, pp. 5–12, INSTICC, SciTePress, 2020.
- [38] K. Kirasich, T. . Smith, and B. Sadler, “Random Forest vs Logistic Regression: Binary Classification for Heterogeneous Datasets,” *SMU Data Science Review*, 2018.
- [39] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, “LightGBM: A Highly Efficient Gradient Boosting Decision Tree,” in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, pp. 3146–3154, Curran Associates, Inc., 2017.
- [40] S. J. Semnani and M. Pandey, “Revisiting the open-domain question answering pipeline,” 2020.
- [41] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, “SQuad: 100,000+ questions for machine comprehension of text,” in *EMNLP 2016 - Conference on Empirical Methods in Natural Language Processing, Proceedings*, 2016.
- [42] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bi-directional transformers for language understanding,” in *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 2019.
- [43] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger, “From word embeddings to document distances,” in *32nd International Conference on Machine Learning, ICML 2015*, 2015.