Universidad Zaragoza
1542

Trabajo Fin de Máster

Aplicación de Técnicas de Minería de Textos para Apoyar la Búsqueda de Información en Contextos Médicos en Español

Application of Text Mining Techniques to Support the Search of Information in Medical Contexts in Spanish

Autor/es

Carlos Sánchez Coronas

Director/es

Sergio Ilarri Artigas
Carlos Tellería Orriols

Máster en Ingeniería Informática
Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
2019

AGRADECIMIENTOS

En primer lugar quiero agradecer a mis tutores, Sergio Ilarri y Carlos Tellería por su esfuerzo y su ayuda para guiarme en el desarrollo de este proyecto con sus sabios consejos y sugerencias.

En segundo lugar, dar las gracias a todo el equipo docente de la Universidad de Zaragoza, los cuales me han ayudado a adquirir competencias que me han hecho mejorar tanto a nivel profesional como personal durante el transcurso de este Máster.

Por otra parte, debo dar las gracias a todos mis amigos por estar siempre cuando los necesitaba para ayudarme a desconectar y relajarme. En especial, agradecer a Félix Angoso, médico interno residente en Calatayud y compañero de piso, quien me ha ayudado con temas más técnicos relacionados con la medicina debido a la naturaleza de mi trabajo.

Y por último, y no menos importante, agradecer enormemente a mis padres por su constante apoyo moral durante el transcurso del Máster y de toda mi formación, y sobre todo por su habilidad para calmarme cuando las cosas no iban bien.

Sin todos vosotros no habría conseguido llegar a ser lo que soy.

Gracias.

Resumen

Hoy en día, la minería de textos es uno de los campos de actuación clave dentro del aprendizaje automático debido a la gran cantidad de texto que se genera diariamente escrito en lenguaje natural, ya que es la forma que tenemos de comunicarnos y expresar nuestras ideas, haciendo a los ordenadores capaces de entender esta forma de comunicación y de extraer información de ella. Hay multitud de campos de actuación, como extracción de información, detección de palabras clave, análisis del sentimiento, clasificación de documentos, etc. Una de las grandes barreras de este sector es el idioma, debido a que las técnicas que se aplican son propias de un lenguaje y por tanto pueden no ser aplicables a otro idioma diferente. Dentro del marco de la medicina, también existen multitud de áreas de actuación, entre ellas, el procesamiento de historias o guías clínicas puede ser de especial interés para la búsqueda automática de información, ya que en medicina, es necesario tener acceso a la última evidencia científica para poder desarrollar la actividad.

Este trabajo se ha centrado en la aplicación de técnicas de minería de textos médicos en español con la idea de avanzar en la automatización de la búsqueda de esta evidencia científica. Durante su desarrollo se detalla el estudio del arte del sector y la implementación y evaluación de un detector de entidades médicas y los problemas encontrados al no contar con un conjunto de datos específico para esta tarea, y de un módulo cuyo objetivo es detectar la intención de *recomendación o sugerencia* en textos, ya que suelen ser las partes de las guías clínicas más importantes donde se resume toda la evidencia científica. A partir de este proyecto, se abren líneas de trabajo futuro en la búsqueda automática de la evidencia científica de la medicina en español.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Actualmente, la minería de textos engloba un amplio campo de trabajo con múltiples posibilidades de actuación [1]. Esto se debe a la gran cantidad de información no estructurada que se genera diariamente de formas muy diversas usando lenguaje natural (noticias, artículos, comentarios en redes sociales, historias clínicas, enciclopedias, etc.) y que no puede ser comprendida o, mejor dicho, procesada directamente por los ordenadores.

Uno de los grandes problemas de este sector de minería de textos es el idioma utilizado, ya que el vocabulario, la gramática, la sintaxis y la semántica son propios de cada lenguaje, por lo que normalmente los mismos procedimientos que se aplican a un idioma pueden no ser aplicables a otros idiomas. Debido a que el inglés es idioma universal y el más utilizado hoy en día, la mayoría de proyectos de investigación realizados han sido enfocados hacia textos escritos en inglés, por lo que las aplicaciones de técnicas de minería de textos en otros idiomas están menos avanzadas.

Dentro del contexto de la medicina, la aplicación de técnicas de minería de textos es especialmente útil. Las tareas de extracción de información y procesado de lenguaje natural pueden ser aplicadas a los informes de alta, evolutivos clínicos, guías de práctica clínica, artículos científicos, y en general a todos los textos médicos, ya sean clínicos o científicos, elaborados en lenguaje natural, para facilitar la búsqueda de información.

En la Sección 1.1 se explica la motivación del presente trabajo. En la Sección 1.2 se detallan los objetivos planteados en este proyecto. Finalmente, en la Sección 1.3 se muestra la estructura de la memoria.

1.1. Motivación

Uno de los problemas en la medicina, igual que en cualquier disciplina científico-técnica, radica en el hecho de que para desarrollar correctamente la actividad es preciso tener acceso a la última evidencia disponible y al estado del arte del conocimientos y los métodos.

Por otro lado, la eventual aplicación del aprendizaje automático a los sistemas de ayuda a la toma de decisiones clínica exigirá, inevitablemente, la capacidad de los sistemas para extraer y analizar información no estructurada y contextualizada de las historias o guías clínicas.

Uno de los posibles inconvenientes se da cuando el tiempo para encontrar la respuesta específica a una consulta puede alargarse demasiado, ya sea porque las guías clínicas suelen ser muy extensas, porque los resultados del buscador de Internet no son muy precisos, o porque no se tiene disponible a una segunda persona que solvente la situación.

Por tanto, sería interesante la elaboración de algún tipo de herramienta que pudiese reducir dicho tiempo y hacer más eficaz este proceso de búsqueda. Una posible solución consistiría en facilitar una aplicación parecida a los buscadores pero específica para este dominio médico, acelerando e incrementando la especificidad de la búsqueda de este tipo de información, de manera que la respuesta sea lo mas precisa posible en función de la consulta.

Una de las fuentes de las que se puede obtener esta información son las guías de práctica clínica [29]. En ellas se recogen un número importante de recomendaciones y evidencias científicas de cara a la actuación clínica en diversas situaciones, valorando el riesgo/beneficio de las mismas. Todos los datos que las forman se ven sometidos a rigurosas revisiones sistemáticas para mantenerse actualizados conforme pasa el tiempo.

Por otro lado, utilizando diversos documentos de contenido sanitario, tal y como son las historias clínicas, se podría adquirir bastante información relevante de cara a la práctica médica. Como es lógico, y para respetar la Ley Orgánica de Protección de Datos y Garantía de los Derechos Digitales 3/2018 y Reglamento General de Protección de Datos (EU 2016/679), estos documentos deben estar completamente anonimizados, con el objetivo de garantizar la confidencialidad del paciente.

Por último, hoy en día se está dando a conocer un nuevo término a la hora de hablar del entorno sanitario: la humanización de la medicina [24]. Dicho término hace referencia a la búsqueda de la atención centrada en la persona, tanto a nivel individual como a nivel socio-familiar. Este concepto surge con el objetivo de no despersonalizar a la persona enferma a la hora de ser tratada por personal sanitario, algo que suele ocurrir cuando un individuo se enfrenta a una tecnificación que no entra dentro de sus conocimientos. Por todo ello, no se puede pasar por alto la importancia que tendría solucionar estos inconvenientes de cara a la calidad de la atención al paciente ya que podría verse incrementada indirectamente al verse reforzado el conocimiento de la evidencia científica más fácilmente por el equipo sanitario.

1.2. Objetivos

El objetivo principal de este trabajo radica en avanzar en el desarrollo de una aplicación que, en su estado final, sea capaz de, a partir de una búsqueda realizada por un médico escrita en lenguaje natural, devolver extractos de guías clínicas con la información que mejor resuelva la consulta en cuestión.

Uno de los objetivos más interesantes de esta área de trabajo es el desarrollo de una herramienta capaz de etiquetar automáticamente términos médicos, a partir de la cual se pueda obtener un conjunto de textos médicos etiquetados en un tiempo mucho más reducido que si se realizara de forma manual, ya que este proceso requiere bastante tiempo y dedicación de profesionales de la salud para conseguir un *gold standard* de calidad. Con esto será posible aplicar distintas técnicas de minería de textos para poder satisfacer las búsquedas solicitadas.

Otro punto importante es poder identificar qué párrafos de estas guías son los más útiles de cara a la práctica clínica, es decir, los más relevantes a la hora de generar los resultados de la búsqueda en estas guías clínicas. Este tipo de textos suelen ser *recomendaciones*, en las que explican cómo actuar ante cierto problema, qué medicamento es mejor para un caso concreto, o *evidencias científicas* de la evolución de tratamientos o enfermedades. Para ello es necesario investigar técnicas de minería de textos que logren clasificar los diferentes párrafos entre recomendaciones y no recomendaciones.

Para llegar a cumplir estos objetivos, la fases a seguir durante el desarrollo del proyecto son las siguientes:

1. Estudio del estado del arte sobre la minería de textos en entornos médicos.
2. Selección de uno o varios conjuntos de datos con información no estructurada en español relacionados con el ámbito de la salud.
3. Identificación de problemas de interés, valorando su relevancia y complejidad
4. Selección de un conjunto de tareas a abordar para su análisis en mayor profundidad: extracción de palabras clave y detección de la intención de recomendación en fragmentos de texto.
5. Evaluación experimental de las técnicas aplicadas.
6. Extracción de conclusiones y elaboración de la documentación, a lo largo de desarrollo del trabajo.

Todos estos puntos se verán desarrollados a lo largo de esta memoria, detallando el proceso realizado para llegar a ellos.

1.3. Estructura de la memoria

De cara a la estructuración de la memoria de este trabajo, se ha optado por dividir las diferentes áreas en Capítulos. En el Capítulo 2, se resume el estado

del arte del tema a tratar. Inicialmente se habla sobre la minería de textos a nivel general, para continuar tratando dicho tema de forma más enfocada al ámbito médico y al castellano en sí. En el Capítulo 3, se describe cómo se ha planificado el proyecto así como las herramientas utilizadas durante su desarrollo. En los Capítulos 4, 5 y 6 se detalla minuciosamente el desarrollo de los dos módulos descritos en los objetivos del trabajo: el detector de entidades médicas y el detector de recomendaciones. Finalmente, en el Capítulo 7, se explican las conclusiones a las que se ha llegado tras la realización del proyecto, junto con las líneas de trabajo futuro que pueden ser de interés en este ámbito.

Adicionalmente, se ha elaborado información complementaria en forma de Anexos. En los Anexos A y B se detalla el proceso de transformación de los conjuntos de datos obtenidos para adecuarlos a las necesidades del proyecto. En el Anexo C se explica el proceso de instalación de *Freeling* [48], una herramienta de *C++* necesaria para el funcionamiento de la aplicación desarrollada. En el Anexo D se muestran fragmentos de código representativos de las pruebas de evaluación realizadas. Finalmente, en el Anexo E se detalla algún ejemplo de las pruebas realizadas de la detección de entidades.

Capítulo 2

Estado del arte

La minería de textos engloba el conjunto de técnicas utilizadas para extraer información significativa de un conjunto de textos escritos en lenguaje natural, es decir, convertir datos no estructurados en datos estructurados [1].

Algunas de las tareas más relevantes son las siguientes:

- Recuperación de la Información (*Information Retrieval*, *IR*) [16]. Se centra en facilitar cierta información; trata de recuperar documentos de un conjunto de datos no estructurados que satisfacen una cierta búsqueda.
- Categorización de textos. (*Text Categorization*) [58]. Trata de asignar un documento o texto a una categoría específica, como por ejemplo filtrado de spam, análisis del sentimiento, etc.
- Extracción de Información. (*Information Extraction*, *IE*) [56]. Técnicas que se basan en procesar documentos no estructurados o semi estructurados para recuperar cierta información específica.
- Reconocimiento de entidades nombradas. (*Named Entity Recognition*, *NER*) [42]. Consiste en extraer y clasificar las entidades en categorías predefinidas, como nombres propios, lugares, etc.

- Procesamiento del lenguaje natural.(*Natural Language Processing,NLP*) [15]. Se basa en el análisis lingüístico tratando de comprender el lenguaje natural mediante los ordenadores. La mayoría de los algoritmos de minería de textos hacen uso de técnicas NLP, como análisis morfológico, semántico y/o sintáctico.

Las técnicas de minería de texto pueden realizarse tanto con aprendizaje supervisado como no supervisado. Por un lado, las técnicas de aprendizaje supervisado [11] tratan de inferir un clasificador a partir de un conjunto de datos previamente procesado y etiquetado de manera manual para realizar predicciones, patrones o recuperar información de los datos. Hay una amplia gama de métodos supervisados como *Naive Bayes* [57], *Support Vector Machine (SVM)* [47], árboles de decisión [36], o métodos basados en reglas [35].

Por otro lado las técnicas de aprendizaje no supervisado, a diferencia de los métodos supervisados, no requieren de un conjunto de datos de entrenamiento. Tratan de detectar datos no etiquetados o de inferir estructuras. Las técnicas más comunes son las de *clustering* [62], donde los documentos se agrupan en distintos grupos o *clusters* de forma que los documentos dentro del mismo *cluster* guardan una similitud entre ellos, minimizando la distancia *intra-cluster*, y los documentos de distintos *clusters* presentan una diferencia entre ellos, maximizando la distancia *inter-cluster*.

2.1. Estado del arte de la minería de textos médicos

Actualmente, existe una gran cantidad de información médica en numerosas publicaciones o textos científicos de libre acceso (*free full text*), por lo que las aplicaciones *NLP* se han ido abriendo camino en el mundo de la medicina [3].

Por ello, se podría definir el *NLP* clínico como un "subcampo" de investigación dentro del propio procesamiento del lenguaje natural [44], cuyo objetivo

engloba aspectos tales como el análisis de textos escritos por los propios pacientes, la recuperación de información biomédica, los registros electrónicos dentro del mundo sanitario, etc.

Como ya se ha comentado, al existir tanta densidad de información médica procedente de diferentes puntos de vista, este campo se ha ido abriendo camino en la actualidad. En algunos estudios como [45], se señala que algunas herramientas y métodos de *NLP* están disponibles de forma gratuita y son fáciles de usar, lo que permite mayor comodidad a la hora de aplicarlos por los investigadores de otros campos.

Sin embargo, estas técnicas se enfrentan a una serie de complicaciones a la hora de aplicarlas al entorno biomédico [39]. Una de ellas es la existencia y la aparición de nuevos términos que hacen referencia a algo ya existente, es decir, a la cantidad de sinónimos, acrónimos, siglas, etc., que podemos encontrar para referirnos a un mismo aspecto. La forma de afrontar esta situación ha generado la creación de sistemas de organización del conocimientos y ontologías [65], aunque requieren la aparición de metodologías que generen una consistencia firme de cara al etiquetado para obtener una uniformidad en toda esta información, además de mejorar la efectividad de dichas herramientas.

Como es lógico, el desarrollo de la minería de textos se ha ido aplicando en los diferentes campos de la medicina. Sin embargo, los artículos suelen centrarse más en los diferentes temas que abarca la medicina, en lugar de enfatizar la metodología de *NLP*. Algunos ejemplos son:

- Oncología [63], donde se observó que, a medida que se avanza en el progreso de la aplicación del *NLP* hacia la investigación oncológica se va creando una infraestructura rentable para avanzar en la atención y el tratamiento del cáncer.
- Radiología [49], donde la aplicación de la minería de textos permite la automatización de una diversidad de tareas en dicho campo.
- Geriatría [13], donde se utilizó un modelo de CRF (*Conditional Random Fields*) con el fin de identificar los diferentes síndromes geriátricos a

partir del texto obtenido de los diferentes pacientes. En él se observó que la falta de contexto muchas veces afectaba a dicha identificación.

- Riesgo de suicidio [17], donde se llegaron a unos resultados prometedores de cara a identificar personas con mayor riesgo de suicidio a través del procesamiento de textos publicados en redes sociales.

Los anteriores puntos muestran el potencial de las aplicaciones de técnicas de minería de textos en el ámbito sanitario siendo una disciplina que se va abriendo camino en dicho sector.

2.2. Estado del arte de la minería de textos médicos en español

En términos generales, se podría decir que el lenguaje científico no se desarrolla ajustándose a la gramática ya existente del lenguaje en el que se quiere exponer, sino que se ajusta a la información que quiere transmitir sobre esa ciencia [25]. Esto quiere decir que, de cara a analizar textos en un idioma diferente al inglés, en este caso el español, antes se tienen que normalizar los términos clínicos que se van a encontrar en dichos documentos.

El principal problema que nos encontramos es el hecho de la existencia de una marcada diferencia en la disponibilidad de recursos lingüísticos en inglés respecto a otros idiomas. Así, se han desarrollado numerosas aplicaciones que tratan el procesamiento de lenguaje natural clínico en inglés, como las ya citadas en la sección anterior. Sin embargo, de cara a procesar textos en lenguaje natural obtenidos de países donde el inglés no es la lengua oficial, estas técnicas no están tan desarrolladas, por lo que abre la puerta a un importante campo de actuación. En la Figura 2.1 se puede apreciar el crecimiento de las publicaciones en este ámbito para idiomas alternativos al inglés.

Para ello se han desarrollado aplicaciones dirigidas al aprendizaje automático para el reconocimiento semántico y la normalización de términos clínicos,

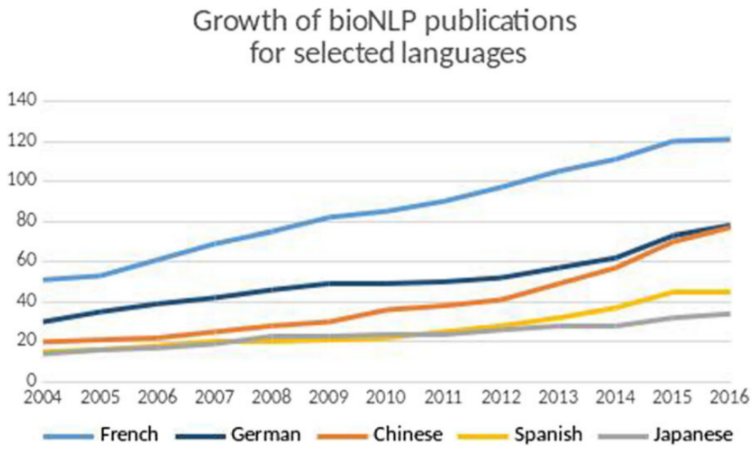

Figura 2.1: Crecimiento de las publicaciones encontradas en *MEDLINE*, durante la pasada década, de estudios en otros idiomas diferentes al inglés. Extraída de [43].

que han demostrado tener buenos resultado. Un ejemplo es el trabajo presentado en [12], donde se crean una serie de vectores de características (*feature vectors*) junto con un enfoque de aprendizaje de cara a identificar términos equivalentes en los diferentes textos a procesar.

Otro ejemplo de trabajo desarrollado en español es [18], donde se presenta una herramienta que trata de determinar las proposiciones negativas en textos clínicos en español, usando para ello un *corpus gold standard* etiquetado manualmente por expertos.

Por otra parte, también se ha avanzado en el desarrollo de aplicaciones cuyo objetivo es la de-identicación o anonimización de textos médicos en español [38], mayormente tratándose de historias clínicas con abundante contenido sensible que debe ser eliminado para su uso en aplicaciones de minería de textos médicos, respetándose así el secreto profesional.

No obstante, la existencia de estos recursos debe ser tomada como una oportunidad para aprovecharlos. Para ello, la traducción automática, ha sido utilizada y estudiada [61], dando buenos resultados de cara a la recuperación de información clínica en varios idiomas, mejorando su acceso, aunque no se ha llegado a ver una mejor recuperación cuanto mejor sea la traducción.

Capítulo 3

Planificación y gestión

En esta sección se explica cómo se ha controlado la organización y evolución del trabajo, así como las tecnologías utilizadas durante el desarrollo. En la Sección 3.1 se detalla la metodología utilizada durante el desarrollo del proyecto. En la Sección 3.2 se explican las herramientas utilizadas en el presente trabajo.

3.1. Metodología

Para el desempeño de este proyecto, se ha seguido la metodología CRISP-DM [55]. Este modelo es el más utilizado en minería de textos y de datos en general, proporciona una descripción estándar del ciclo de vida de un proyecto de minería de datos estructurado en seis fases (Figura 3.1). Algunas de estas fases son bidireccionales, lo que permite revisar las fases anteriores.

Dado que se trata de un área totalmente nueva para el alumno, es necesario una etapa inicial de formación y estudio del estado del arte sobre los temas a abordar. Las fases del proyecto son las siguientes:

- Formación y estudio del estado del arte.

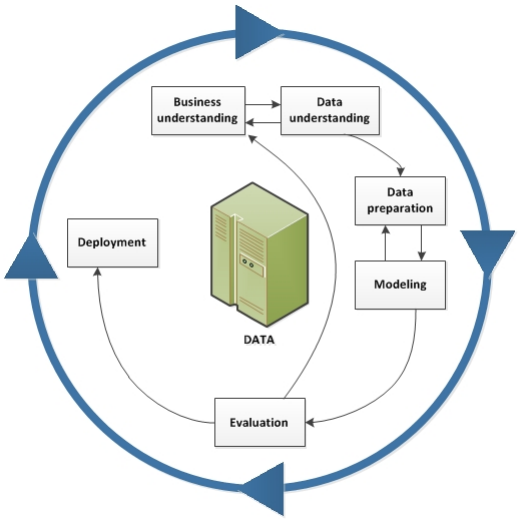

Figura 3.1: Diagrama de proceso de las diferentes fases de CRISP-DM. Extraída de [28].

- Comprensión del negocio. Consiste en definir los objetivos del proyecto, evaluar la situación (recursos necesarios, requerimientos, etc.) y convertir estos objetivos en un problema de minería de datos con el que generar el proyecto inicial (plan, técnicas, herramientas, equipo, etc.).
- Estudio y comprensión de los datos. Recolección, comprensión, exploración y verificación de la calidad de los datos.
- Preparación de los datos. Aborda las tareas de procesamiento de los datos. (selección, limpieza, construcción, integración, etc.).
- Modelado. Selección, diseño y desarrollo de las técnicas pertinentes para abordar los objetivos planificados.
- Evaluación. Estudiar los resultados obtenidos y seleccionar el modelo más óptimo. Determinar si hay alguna cuestión importante del negocio que no haya sido considerada lo suficiente.
- Despliegue. Puesta en producción del modelo, generación de memoria final y revisión del proyecto.

En la Figura 3.3 se muestra un cronograma en el que planifica inicialmente el

coste temporal de cada fase del proyecto desde la fecha de inicio (01/04/2019) hasta la fecha de entrega prevista (22/11/2019). Nótese que los meses de junio y julio están suprimidos debido a que, por motivos personales previstos de antemano, no se pudo avanzar en el desarrollo del proyecto.

Por otra parte, para la gestión de las diferentes tareas de cada etapa del trabajo, se ha hecho uso de la herramienta *Bitrix24*, un software de gestión de tareas online totalmente gratuito que dispone de una sección *Kanban* que se ha usado para dividir las tareas en tres bloques: *Pendiente, En progreso y Terminado*, tal y como se muestra en la Figura 3.2.

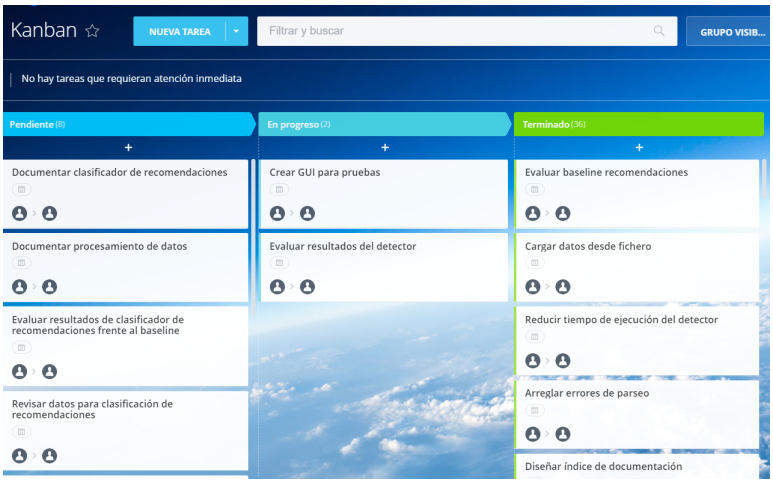

Figura 3.2: Tareas organizadas usando la herramienta *Bitrix24*.

3.2. Herramientas utilizadas

El proyecto se ha desarrollado en el ordenador personal del alumno, cuyas características son: HP Pavilion, Intel Core i7-8700, 8 GB RAM, 1 TB. Se han instalando en él todas las herramientas y datos necesarios para su desarrollo.

Para el desarrollo del proyecto se ha decidido utilizar *Python 3* [51] usando el IDE *Anaconda Navigator* [4] debido a su gran potencial para las tareas de minería de textos, ya que cuenta con un gran número de librerías que facilitan su implementación. Además, se ha usado *MongoDB* [41] como sistema

de bases de datos, ya que su almacenamiento en formato *JSON* [31] permite una carga directa y transformación rápida de los datos al tipo de dato diccionario de *Python*. A continuación se listan las herramientas instaladas para el desarrollo del proyecto:

- Freeling [48]: una herramienta de procesamiento del lenguaje natural implementada en C++ y con posibilidad de utilizar en otros lenguajes como *Java*, *Perl* o *Python* mediante instalación y compilación previa.
- nltk [8]: otra herramienta de procesamiento del lenguaje natural para *Python*.
- Spacy [27]: otra herramienta de procesamiento de lenguaje natural para *Python*.
- Pandas [40]: extensión de la librería de *Python numpy* [46] para la manipulación y análisis de datos en *Python*.
- SymSpell [37]: utilizada para la corrección ortográfica.
- Scikit-learn [10]: librería para *Python* con multitud de funcionalidades de aprendizaje automático.
- FlashText [22]: una herramienta diseñada para las tareas de búsqueda y reemplazo de elementos en textos.
- re [52]: librería de *Python* para la aplicación de expresiones regulares en textos.

Inicialmente, se decidió desarrollar el proyecto en un entorno Windows 10, ya que es el sistema operativo que había instalado por defecto en el ordenador del alumno, pero debido a problemas en la instalación de *Freeling*, que no se pudieron solucionar, fue necesario optar por otro entorno. Esta herramienta era necesaria ya que cuenta con funcionalidades no encontradas en otras herramientas desarrolladas para trabajar textos en español (lematización), por lo que se decidió instalar y desarrollar el trabajo en un entorno Linux donde esta instalación pudo realizarse correctamente. El proceso de instalación en Linux y los problemas encontrados en Windows se detallan en el Anexo C.

Figura 3.3: Cronograma inicial del proyecto.

Capítulo 4

Detector de entidades

En esta sección se detalla el proceso de análisis, implementación y resultados de un sistema de detección de entidades médicas en español. En la Sección 4.1 se detalla el análisis de los datos, en la Sección 4.2 se explica como se ha desarrollado el modelado, y en la Sección 4.3 se muestran los resultados obtenidos.

4.1. Análisis

Engloba las tareas de estudio y comprensión de los datos, y preparación de los datos de la metodología CRISP-DM.

Una de las formas más efectivas a la hora de desarrollar este tipo de herramientas consiste en construir un clasificador usando un *corpus gold standard* como datos de entrenamiento [34], es decir, un conjunto de textos con entidades previamente etiquetadas con la suficiente calidad y cantidad por expertos en la materia, en este caso, médicos especializados.

Por ejemplo, en [50] se utiliza un conjunto de datos en inglés previamente etiquetados sobre el que se aplican técnicas de procesamiento natural para

usarlos como datos de entrenamiento para un clasificador de entidades.

Por otro lado, en [21] se explica el desarrollo de un sistema de detección y clasificación de entidades en español, haciendo uso de un *corpus* con textos etiquetados manualmente, y realizando tareas de aprendizaje automático sobre éste.

Debido a la ausencia de este *corpus gold standard* y el desproporcionado esfuerzo, que excede el alcance del presente proyecto, necesario para crearlo, debido a la necesaria colaboración con expertos médicos, esta opción ha sido descartada.

Una posible alternativa para construir un detector de entidades si no se cuenta con este *corpus* consiste en usar un diccionario de términos clasificados como referencia. Un ejemplo de este tipo de trabajos es [53], donde se usa un diccionario de términos médicos en inglés para crear un detector de entidades en textos médicos. Esta opción ha sido la elegida, dada su viabilidad, por lo que el primer paso ha consistido en investigar posibles conjuntos de datos disponibles para lograr construir este diccionario.

Este proceso de búsqueda no ha sido sencillo, debido tanto a que la cantidad de datos médicos en castellano disponibles es mucho más reducida que en inglés como a que la mayoría de los datos de calidad encontrados engloban campos muy acotados de la medicina, como por ejemplo radiología, hematología, o proteínas, sin reflejar una visión global de la medicina general.

Finalmente, se ha conseguido obtener dos conjuntos de datos, a priori con bastantes términos médicos como para poder desarrollar un modelo. Estos datos se han obtenido de la versión traducida al castellano de SNOMED-CT (*Systematized Nomenclature of Medicine – Clinical Terms*) [60] y de DeCS (*Descriptores de Ciencias de la Salud*) [20], ambos con previa solicitud de licencia para poder obtener los datos.

4.1.1. SNOMED-CT

SNOMED-CT es una terminología clínica integral originalmente desarrollada en inglés. Ha sido traducida, posteriormente, a otros idiomas, entre ellos el español. La versión en castellano contiene actualmente más de un millón de términos, denominados conceptos, en distintas clases jerarquizadas (trastorno, hallazgo, procedimiento, sustancia, etc.). A su vez, estos términos presentan ciertas relaciones predefinidas entre ellos; las más comunes son relaciones de sinonimia y subclases. Toda esta información se divide en dos ficheros, uno con los términos y su respectiva clase (más de un millón de términos) y otro con las relaciones (más de cinco millones de relaciones), ambos en formato CSV.

En la Tabla 4.1 se muestra un ejemplo de la información extraída del fichero de términos, donde *conceptId* es un código identificador del concepto, que es el mismo para todos los términos que hacen referencia a este mismo concepto, y *typeId* es otro código que indica si el término es un sinónimo o es el término principal.

conceptId	**typeId**	**term**	**termType**
248425001	900000000000013009	pirexia	hallazgo

Tabla 4.1: Ejemplo de información extraída de un término de SNOMED-CT

Por otra parte, en la Tabla 4.2 se muestra un ejemplo de los datos extraídos del fichero de relaciones, donde en este caso el campo *typeId* indica el tipo de relación entre los dos conceptos definidos por su *conceptId*, en este caso llamados *sourceId* y *destinationId*.

sourceId	**typeId**	**destinationId**
248425001	116680003	64882008
pirexia	*sinonimia*	*fiebre*

Tabla 4.2: Ejemplo de información extraída de una relación de SNOMED-CT

Tras el procesamiento y adaptación de este fichero de términos con objeto de poder construir un diccionario adaptado a las necesidades de este proyecto,

se obtienen 951.213 términos, contando sinónimos, divididos en 20 clases. En el Anexo A se explica con más detalle el procesamiento los ficheros para la construcción de un diccionario.

4.1.2. DeCS

El vocabulario estructurado y multilingüe DeCS [20] fue creado por BIREME (Centro Latinoamericano y del Caribe de Información en Ciencias de la Salud) para servir como un lenguaje único en la indización de artículos de revistas científicas, libros, anales de congresos, informes técnicos, y otros tipos de materiales, así como para ser usado en la búsqueda y recuperación de asuntos de la literatura científica en las fuentes de información disponibles en la Biblioteca Virtual en Salud, como LILACS, MEDLINE y otras.

Fue desarrollado a partir del MeSH (*Medical Subject Headings*) de la *U.S. National Library of Medicine* con el objetivo de permitir el uso de terminología común para búsqueda en múltiples idiomas, proporcionando un medio consistente y único para la recuperación de la información de interés.

Los conceptos que componen DeCS son organizados en una estructura jerárquica, permitiendo la ejecución de búsqueda en términos más amplios o más específicos o la recuperación de todos los términos que pertenezcan a una misma estructura jerárquica. Es un vocabulario dinámico totalizando 33.966 descriptores y calificadores, siendo de estos 29.431 del MeSH y 4.535 exclusivamente del DeCS.

La fuente de datos obtenida de DeCS también contiene múltiples ficheros, de los cuales solamente uno es de utilidad para este trabajo. En él se recogen todos los términos que contiene DeCS traducidos al castellano en formato XML. El problema de este fichero es que, a diferencia de SNOMED-CT, no viene informada la clase de cada término; sin embargo, a partir de otro campo (*treeId*) los desarrolladores de DeCS ofrecen una aplicación basada en servicios web [19], mediante la cual se puede obtener diversa información, entre ella la clase de más alto nivel de la jerarquía de cada término, que se ha usado para determinar la clase de cada elemento. Esta información se obtiene a través del siguiente *endpoint REST*:

`http://decs.bvsalud.org/cgi-bin/mx/cgi=@vmx/decs/?tree_id=`

En la Tabla 4.3 se muestra un ejemplo de la información extraída de este fichero, donde el campo *ancestor* hacer referencia a la clase de nivel superior obtenida con el método explicado.

termName	**termUI**	**treeId**	**ancestor**
Pulmón	spa0000586	D03.633.100.221.173	ANATOMÍA

Tabla 4.3: Ejemplo de información extraída de una relación de DeCS

En total se obtienen 91.823 términos médicos divididos en 20 clases, 10 veces menos términos que para SNOMED-CT. Todo el procesado y adaptación del fichero para la creación de un diccionario adaptado a las necesidades de este proyecto se explica con más detalle en el Anexo B.

4.2. Modelado

En este apartado se detalla la fase de modelado de la metodología CRISP-DM.

Tras las primeras pruebas en la fase de implementación, se llegó a la conclusión de que es necesario contextualizar el texto que se está etiquetando para obtener resultados de mejor calidad. Esto se debe a que al disponer únicamente de un listado de términos, no se sabe si el término que se está detectando tiene sentido dentro del contexto actual. Un ejemplo es el siguiente, donde se detecta 'costillas' como 'estructura corporal', que a priori tiene sentido ya que es una estructura anatómica del cuerpo humano, pero no en el contexto de la situación que refleja el texto, donde se refiere a un tipo de comida:

*"Varón acude al Servicio de Urgencias por reacción anafiláctica tras la ingesta de '**costillas** de cerdo', según refiere. El paciente alega la aparición de numerosas lesiones habonosas, eritematosas y pruriginosas tanto en extremidades superiores como en tórax y abdomen, tras la comida. Afebril. No*

refiere haber cambiado de productos de higiene, ni haber utilizado perfumes o ropa nueva. No otra clínica asociada al cuadro".

La forma idónea de crear este contexto sería a partir de una ontología bien formada, donde en función de las relaciones existentes entre los términos médicos detectados dentro de la ontología, se determine si cada término tiene sentido dentro del contexto actual. Sin embargo, a partir de los datos disponibles no es posible crear tal estructura, debido a que , como se verá más adelante, las relaciones de las que se dispone (únicamente en SNOMED-CT) no son las ideales, por lo que se han estudiado algunas alternativas para solventar este problema:

- Distancia entre códigos. Esta primera prueba consiste en generar el contexto en función de la distancia entre los identificadores de los términos detectados. Esta distancia se ha calculado comparando los dígitos en la misma posición de ambos elementos, de tal forma que si el número de dígitos coincidentes es mayor o igual que el número de dígitos no coincidentes, se estima que son cercanos, y por tanto relacionados.

  Para esta prueba se ha supuesto que los códigos de los términos de ambos conjuntos de datos están ordenados por relación jerárquica. Sin embargo, como se verá en la sección de resultados, esta suposición resultó falsa.

- Relación entre términos. En esta fase se ha intentado utilizar las relaciones existentes en las fuentes de datos, de manera que sólo se dan por válidos los términos detectados que tengan relaciones entre sí.

  Con DeCS no se ha podido realizar esta prueba, ya que no facilita ningún fichero con esta información, y aunque se podría obtener a partir de los identificadores facilitados y la aplicación basada en servicios web, no se ha podido recuperar debido al enorme tiempo de ejecución necesario y, a que, tras un número de peticiones, el sistema bloquea la IP y no permite realizar más llamadas hasta pasado cierto tiempo. El límite de peticiones permitidas o el tiempo de restricción no se mencionan en sus términos de uso.

  Con SNOMED-CT, gracias al fichero de relaciones facilitado, sí que es posible realizar esta prueba. Para este experimento, se detectan solo

términos que estén relacionados entre sí con una profundidad de 3. Se ha decidido que la profundidad sea tres porque, debido a los tamaños de ambos ficheros, obtener mayor profundidad supone un coste de ejecución del orden de días.

Se han implementado tres tipos de detectores, cada uno con diferentes funcionalidades, para valorar cuál de ellos da mejor resultado, aplicando cada una de las pruebas descritas.

4.2.1. *Baseline*

Como primer paso, se ha desarrollado un modelo simple que sirve de referencia a la hora de valorar la mejora de otros detectores más complejos. Se realiza únicamente un preprocesado en el que elimina caracteres no válidos, tanto de los textos como de los términos, y convierte todo en minúsculas. En este modelo se compara cada término con cada *ngrama* del texto y devuelve aquellos términos que sean idénticos en el texto a procesar.

Inicialmente se utilizó la librería *re* de *Python* [52], que permite el uso de expresiones regulares mediante las cuales comprueba la coincidencia de cada término en el texto. En el Algoritmo 1 se muestra el funcionamiento descrito.

El tiempo de ejecución de este algoritmo es ligeramente elevado, desde 15 segundos hasta 200 dependiendo del tamaño del texto. Este tiempo es más alto cuando se usa el diccionario de términos de SNOMED-CT que contiene casi un millón de elementos, por lo que se han investigado posibles mejoras para reducir este tiempo de ejecución.

Existe una librería llamada *FlashText* creada específicamente para las tareas de búsqueda y remplazo de palabras en documentos [22], que realiza la misma función que el algoritmo anterior, pero de manera más eficiente, reduciendo considerablemente el tiempo de ejecución. En este caso es necesario introducir como parámetro todos los elementos del diccionario en un objeto de esta librería, llamado *KeywordProcessor*, para posteriormente buscar coincidencias en el texto.

Algoritmo 1 *Baseline* inicial

Entrada: Texto a procesar, diccionario de términos
Salida: Términos coincidentes en el texto.

```
preprocesar texto
preprocesar términos
para todo término hacer
  comprobar coincidencia de término en texto mediante expresión regular.
  si coincide entonces
    añadir a lista de coincidencias
  fin si
fin para
devolver lista de coincidencias
```

Los términos que se introducen en esta librería son transformados en una estructura de datos *trie* [32]. Un *trie* está formado por un conjunto de datos organizados en una estructura "tipo árbol" que permite una recuperación de información más eficiente. Toda la información que está contenida dentro de un *trie* consiste en una agrupación de claves. En este caso, una clave hace referencia a una secuencia de una serie de símbolos que forman parte de un alfabeto determinado. Las claves son almacenadas en las hojas del árbol y los nodos internos son pasarelas para guiar la búsqueda. Estas claves van a ser almacenadas en las "hojas" de la estructura de árbol, existiendo una serie de nodos internos que van a permitir la intercomunicación entre ellas, facilitándose así la búsqueda. Este árbol está estructurado de forma que cada letra de la clave se ubica en un nodo de tal manera que los hijos de un nodo representan las distintas posibilidades de símbolos diferentes que pueden continuar al símbolo representado por el nodo padre. En la Figura 4.1 se muestra un ejemplo gráfico de la estructura descrita .

En el Algoritmo 2 se muestra el funcionamiento del *baseline* usando esta librería. El único inconveniente es que esta librería no permite añadir metadatos a los términos introducidos, por lo que al incluirlos se pierde el resto de información, entre ellas la clase del término. Debido a esto, para cada coincidencia es necesario recuperar el resto de metadatos. Aun así, el rendimiento de este segundo algoritmo es mucho mejor que el primero, como se puede ver en la Figura 4.2, donde se ha procesado 50 textos con cada uno de los

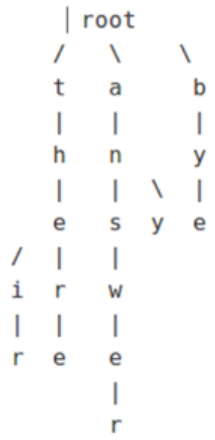

Figura 4.1: Estructura de datos tipo *trie*. Extraída de [32].

algoritmos propuestos, siendo el tamaño de estos textos, 45 de entre 130 y 500 caracteres, y los 5 restantes de más de 500. El resto de valores de tiempo se han interpolado.

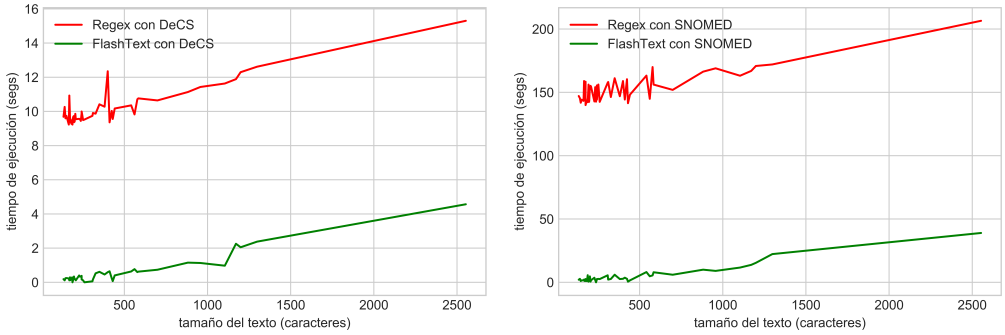

Figura 4.2: Comparativa del tiempo de ejecución usando *re* y *FlashText*.

4.2.2. Lematización

El principal problema del *baseline* es que al mínimo cambio de una palabra en el texto el detector es incapaz de encontrar dicha palabra; por ejemplo, si en el diccionario aparece el término *dolor de pierna* y en el texto *dolor en las*

Algoritmo 2 *Baseline* optimizado usando *FlashText*

Entrada: Texto a procesar, diccionario de términos
Salida: Términos coincidentes en el texto.

```
preprocesar texto
preprocesar términos
incializar KeywordProcessor
para todo término hacer
  introducir término en KeywordProcessor.
fin para
buscar coincidencias en el texto(KeywordProcessor.extract_keywords(texto))

para todo término encontrado hacer
  recuperar metadatos del término
  añadir a la lista de coincidencias
fin para
devolver lista de coincidencias
```

piernas, el detector anterior no encontraría coincidencia. Para mejorar estos casos, se han preprocesado todos los términos y texto de la siguiente forma:

1. Convertir todo a minúsculas. Al igual que con el modelo anterior, se han convertido tanto los términos como el texto a minúsculas. Para las siglas se ha hecho una excepción: si el término en el diccionario aparece todo en mayúsculas, se entiende que se trata de siglas, y por tanto este paso no se aplica ni en el término ni en los textos.

2. Eliminación de palabras vacías. Suprimir aquellas palabras que carecen de significado y no son relevantes para este tipo de aplicaciones, como artículos, preposiciones, determinantes, etc.

3. Aplicar lematización. Es un proceso lingüístico que consiste en, dada una forma flexionada, hallar el lema correspondiente. El lema es la forma que por convenio se acepta como representante de todas las formas flexionadas de una misma palabra, llamado también base léxica. Por ejemplo, el lema de las palabras *rojo*, *roja*, *rojos* o *rojas* es *rojo*

Para este procesado y lematización se ha hecho uso de la herramienta *Free-*

ling [48], una librería programada en *C++* que cuenta con funcionalidades de análisis del lenguaje para múltiples idiomas, entre ellos el español. Esta herramienta cuenta con una *API* para poder ser utilizada desde *Python*, que requiere una previa instalación y compilación para poder ser utilizada, ya que no puede ser instalada mediante *pip install* como es habitual en *Python*.

No se ha encontrado ninguna librería en *Python* que recupere el lema de las palabras en español con tan buenos resultados como *Freeling*. Por ejemplo, la librería *nltk* y su paquete *SnowballStemmer* cuenta con múltiples funcionalidades para el castellano, pero no proporciona la funcionalidad de lematizar palabras: únicamente se puede obtener la raíz o lexema, lo cual es menos preciso y puede dar a errores a la hora de detectar comparando raíces, como con las palabras *hombre* y *hombro*, que comparten la misma raíz *hombr*, pero que se tratan de unidades semánticas diferentes, entendiendo como unidad semántica al conjunto de palabras o expresiones que tienen un significado similar. Otro ejemplo es *spacy*, otra librería de procesamiento del lenguaje natural que soporta el español. Esta herramienta sí que cuenta con la opción de lematizar palabras, el problema es que su índice de acierto es bastante menos elevado, errando sobre todo a la hora de lematizar sustantivos, donde palabras como *huevo*, *correo* o *corte*, las convierte en *huevar*, *correar* y *cortar*, aun detectándolas correctamente como sustantivos en el análisis morfológico.

Mediante una previa segmentación del texto en palabras, llamado *tokenización*, se procesa cada una de éstas, obteniendo su lema, transformándola en minúsculas, eliminándola si se trata de una palabra vacía. Para comprobar la coincidencia de los elementos procesados, también se ha hecho uso de la librería *FlashText*. En el Algoritmo 3 se muestra el funcionamiento de este modelo.

Para mejorar el tiempo de ejecución de este detector se ha introducido, en la base de datos de donde se recupera la información de cada término, un campo con el término ya procesado y transformado a su lema, por lo que la primera parte de este modelo donde se calculan los lemas de los términos, cuyo tiempo de ejecución era de hasta 40 segundos en el caso de *SNOMED-CT*, ya no es necesario.

Adicionalmente, tras las primeras pruebas, se hizo uso de un corrector ortográfico para paliar los problemas a la hora de no detectar palabras escritas

Algoritmo 3 Detector con lematización

Entrada: Texto a procesar, diccionario de términos
Salida: Términos coincidentes en el texto.

```
incializar KeywordProcessor
para todo término hacer
  tokenizar término
  para todo palabra del término hacer
    si es palabra vacía entonces
      eliminar palabra
    fin si
    transformar palabra a su lema
  fin para
fin para

tokenizar texto
para todo palabra del texto hacer
  si es palabra vacía entonces
    eliminar palabra
  fin si
  transformar palabra a su lema
fin para

para todo lema hacer
  introducir lema en KeywordProcessor.
fin para
buscar coincidencias en el texto(KeywordProcessor.extract_keywords(texto))

para todo elemenos encontrados hacer
  recuperar metadatos del término
  añadir a la lista de coincidencias
fin para
devolver  lista de coincidencias
```

incorrectamente, como *hormiguoe* al querer escribir *hormigueo*. Para esta tarea se ha hecho uso de la librería *SymSpell* [37], una herramienta muy eficiente con una media de 0.002 segundos de cálculo por palabra. Esta librería permite cargar un diccionario específico como fuente para la corrección, lo que ha permitido el uso de un diccionario en español, que consta de 1.211.000 elementos, obtenido de un proyecto de código abierto donde se recogen las palabras más frecuentes obtenidas de subtítulos de películas [26]. Esta herramienta usa la distancia de Levehnstein como métrica para determinar la similitud entre palabras [6], que se explicará en la siguiente sección. La eficacia de esta herramienta está limitada por el contenido del diccionario, no siendo capaz de corregir palabras muy técnicas que no aparezcan en éste. La confianza de los elementos detectados tras utilizar el corrector ortográfico debe ser menor, ya que es posible que la corrección aplicada no sea correcta, pudiendo corregir palabras demasiado técnicas que sí que están correctamente escritas, por lo que se incluirá *Probabilidad Media* en estos elementos.

4.2.3. Detector aproximado

Otra alternativa consiste en realizar un cálculo para comprobar la similitud entre el contenido del texto y los términos del diccionario médico. Esto solucionaría el problema de no detectar términos con faltas de ortografía, que son frecuentes en los documentos de tipo historia clínica, o palabras en singular y plural o masculino y femenino que también se solventa en el detector descrito anteriormente. Por otro lado, es posible que se detecten palabras que sean parecidas, pero que se traten de unidades semánticas diferentes. Este detector sería el equivalente a usar un corrector ortográfico, pero con las fuentes de datos de SNOMED-CT o DeCS como diccionario de corrección.

Este cálculo para comprobar la similitud entre palabras se realiza mediante la distancia de Levenshtein [6]. Es una métrica que mide la diferencia entre dos secuencias de caracteres. Representa el número mínimo de ediciones en dichos caracteres (añadir, cambiar o borrar) necesarios para transformar una secuencia en la otra. La definición matemática de la distancia de Levenshtein entre dos secuencias $a, b, lev_{a,b}(|a|, |b|)$ es la siguiente:

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & if\ min(i,j) = 0, \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & otherwise. \end{cases}$$

Donde $1_{(a_i \neq b_j)}$ es 0 cuando $a = b$ y 1 si no. Nótese que las tres filas de la parte del mínimo en la anterior fórmula corresponden con un borrado, una inserción y una sustitución, respectivamente.

Por ejemplo en la Figura 4.3, la distancia entre TRASTO y RESTO es 2, ya que hace falta una sustitución (A por E) y una eliminación (la primera T).

T	R	A	S	T	O
	R	E	S	T	O

Figura 4.3: Ejemplo de cálculo de la distancia de Levenshtein.

Existe un módulo para *Python*, llamado *python-Levenshtein* [5], que realiza este cálculo. Esta librería también cuenta con una relación de similitud o ratio de Levenshtein, que se calcula mediante la siguiente fórmula:

$$\frac{(|a| + |b|) - lev_{a,b}(i,j)}{|a| + |b|}$$

Esta medida es interesante porque tiene en cuenta el tamaño de las secuencias. No es lo mismo una distancia de Levenshtein = 2 en secuencias de 3 ó 4 caracteres, donde muy posiblemente se trate de palabras totalmente distintas, como *dado* y *duro*, que en secuencias de 10 donde seguramente se trate de algún error ortográfico o singular/plural. Por lo tanto, se ha decidido usar esta medida para calcular la similitud entre palabras, siendo el umbral mínimo 0.9, ya que un umbral inferior podría suponer el aumento de falsos positivos sobre todo en palabras cortas.

Para que este cálculo sea correcto, han de compararse elementos con el mismo número de palabras, por lo que es necesario dividir el texto en *ngramas* en función del número de palabras que tenga cada término a detectar.

Ya que calcular los ngramas del texto para cada término es muy ineficiente, y elevaría el tiempo de ejecución del detector, se han calculado previamente los ngramas de los textos para n = 1,2,3,4,5,6; siendo 6 el número máximo de palabras que tienen los términos del diccionario, y en función del tamaño de cada término, se compara con una lista u otra.

El funcionamiento de este modelo se detalla en el Algortimo 4. Su gran inconveniente es su elevado tiempo de ejecución con respecto a los dos anteriores, siendo de entre 200 y 500 segundos dependiendo del tamaño del texto, mientras que los anteriores no superaban el segundo. El coste máximo de este algoritmo es del orden de O(m*n) donde m es el número de términos del diccionario y n el número de palabras del texto. Se ha decidido mantenerlo tal y como se ha descrito, y en el caso de que los resultados de este modelo sean mucho mejores con respecto a los demás en la fase de evaluación, modificar el algoritmo para abordarlo de forma más eficiente mediante paralelización, ya que al procesar cada término de manera independiente podría ser aplicable, y optimización de código.

4.3. Resultados

En esta sección se detalla la fase de evaluación de la metodología CRISP-DM. Tras valorar los resultados de las distintas pruebas, se concluirá qué diccionario de términos es más eficaz y qué tipo de detector da mejores resultados.

Para la valoración de los resultados se han usado 10 textos obtenidos de historias clínicas totalmente anonimizadas proporcionadas por los directores del TFM. Al ser textos extensos y con bastante contenido médico, se puede comprobar el funcionamiento de los detectores implementados etiquetando previamente los términos médicos encontrados en el texto de forma manual y comparando el etiquetado con los resultados de cada prueba. Estos resultados

Algoritmo 4 Detector aproximado

Entrada: Texto a procesar, diccionario de términos
Salida: Términos coincidentes en el texto.

```
para todo n entre n=1 y 6 hacer
  calcular ngramas del texto
fin para
para todo término hacer
  calcular tamaño de ngrama del término = t
  para todo ngrama del texto para n = t hacer
    calcular ratio de Leveneshtein
    si ratio mayor que umbral entonces
      añadir término a la lista de coincidencias
    fin si
  fin para
fin para
devolver lista de coincidencias
```

se dividen en *Verdadero Positivo (True Positive), Falso Positivo (False Positive), Verdadero Negativo (True Negative) y Falso Negativo (False Negative)*, pudiendo crear así una matriz de confusión.

- Verdadero Positivo: Elemento que aparece etiquetado en el texto y se ha detectado correctamente.
- Falso Positivo: Elemento que no aparece en el texto y se ha detectado incorrectamente (puede ocurrir sobre todo en el detector aproximado). También se incluyen en este grupo los términos que aparecen en el texto pero han sido detectados en una clase incorrecta.
- Verdadero Negativo: Se trata de elementos que sí que se han detectado, pero se ha estimado correctamente que no procede etiquetar en el caso actual. En el caso del texto de ejemplo de la Sección 5.2, detectar *costillas (estructura corporal)* y estimar que no procede en ese texto sería un Verdadero Negativo. Este tipo solo aparecería en la pruebas de contextualización, ya que sin este proceso, el detector únicamente detecta los términos encontrados, dando todos como correctos, sin estimar ninguno como negativo.

- Falso Negativo: Elemento que sí que está etiquetado previamente, pero que el detector no ha encontrado. En el caso de las pruebas de contextualización, detectar un término y estimarlo como negativo cuando debería ser positivo también es un Falso Negativo.

Como medidas de evaluación se han utilizado la precisión y el *recall*:

- Precisión: Mide el índice de acierto del modelo, es decir, cuántas instancias se han detectado correctamente. Se define mediante la siguiente fórmula:
$$Precision = \frac{TP}{TP + FP}$$
- *Recall*: Mide la exhaustividad del modelo, es decir, cuántos elementos que haya que detectar se han detectado realmente. Se define mediante la siguiente fórmula:
$$Recall = \frac{TP}{TP + FN}$$

Cabe mencionar que, en ambos conjuntos de datos, hay elementos que no se consideraron en el etiquetado manual, como *antecedente de*, que es un concepto clínico que implica modificación del riesgo de padecer algo o *hermano fallecido*, que es un antecedente familiar y puede ser claramente un indicador de riesgo, pero que sí aparecen etiquetados correctamente y son términos a tener en cuenta. Estos términos no se tuvieron en cuenta en el etiquetado manual debido a la falta de conocimiento médico por parte del alumno, que no se etiquetaron por desconocimiento o por acotar demasiado el etiquetado, centrándose en los elementos que el alumno consideraba más relevantes, como medicamentos, síntomas, enfermedades o procedimientos médicos. Todo esto ha dificultado enormemente esta tarea de evaluación, por lo que ha sido necesario adaptar el etiquetado en función de estos términos no considerados.

Por otra parte, debido a la gran variación entre los elementos de SNOMED-CT y DeCS, no se han podido cuadrar los elementos etiquetados en ambos conjuntos de datos ya que bastantes términos que aparecen en SNOMED-CT no aparecen en DeCS y viceversa, como por ejemplo *hospital*, *historia clínica*

o *acude a*, que aparecen en SNOMED-CT pero no en DeCS. Dado que la relevancia de estos elementos no es muy elevada, se ha decidido considerarlos correctos, pero no como falsos negativos en el otro conjunto de datos.

4.3.1. Búsqueda de contexto

Como primera prueba se ha utilizado el modelo *baseline* con ambos métodos para tratar de contextualizar el texto descritos en la Sección 5.2 (similitud de códigos y relaciones).

En primer lugar, en la Tabla 4.4 se muestran las matrices de confusión sin aplicar ningún tipo de contextualización, para comparar si el aplicarlas mejora o empeora los resultados.

SNOMED-CT

		Detección	
		Positivo	Negativo
Real	Positivo	240	149
	Negativo	82	0

DeCS

		Detección	
		Positivo	Negativo
Real	Positivo	118	190
	Negativo	70	0

Tabla 4.4: Matrices de confusión del *baseline* sin contextualización con ambos conjuntos de datos.

En la Tabla 4.5 puede apreciarse cómo los resultados son superiores, aunque mejorables, en SNOMED-CT con respecto a DeCS, sobre todo en cuanto al *recall*. Se puede apreciar cómo el número de elementos no detectados (Falsos Negativos) es superior en DeCS frente a SNOMED-CT, y también como se detectan más elementos en SNOMED-CT que en DeCS. Esto se debe a que el contenido de SNOMED-CT es mucho más completo, en cuanto al número de términos presentes, tal y como se refleja en los resultados, habiendo 10 veces más términos que en DeCS.

En segundo lugar, en la Tabla 4.6 se muestran las matrices de confusión de los resultados aplicando el método de similitud de códigos para buscar el contexto con ambos conjuntos de datos. En la Tabla 4.7, se puede apreciar como

	Baseline	
	SNOMED-CT	**DeCS**
Precisión	0.74	0.64
Recall	0.62	0.34

Tabla 4.5: Métricas de evaluación del detector *baseline sin contextualización*.

los resultados no tan eficaces como los anteriores. La precisión ha aumentado, pero el *recall* ha disminuido considerablemente, debido a que estima como negativos muchos de los términos cuando no deberían serlos. Con los resultados obtenidos podría decirse que la similitud entre códigos no aporta información acerca de la relación entre los términos y que, por tanto, no es útil para la búsqueda de contexto.

		SNOMED-CT	
		Detección	
		Positivo	Negativo
Real	Positivo	157	232
	Negativo	33	49

		DeCS	
		Detección	
		Positivo	Negativo
Real	Positivo	102	206
	Negativo	23	47

Tabla 4.6: Matrices de confusión del *baseline* usando la similitud entre códigos como contextualización con ambos conjuntos de datos.

	Similitud de códigos	
	SNOMED-CT	**DeCS**
Precisión	0.83	0.82
Recall	0.40	0.33

Tabla 4.7: Métricas de evaluación del detector *baseline* con similitud entre códigos como contextualización.

Por último, en la Tabla 4.8 se muestra la matriz de confusión de los resultados aplicando las relaciones entre términos de la fuente de datos para obtener el contexto, en este caso solo de SNOMED-CT, ya que, como se ha explicado, no ha sido posible obtener esta información para el conjunto de datos de DeCS. Los resultados obtenidos son incluso peores que con la similitud entre

códigos, clasificando la mayoría como negativos, lo que reduce considerablemente el *recall*, como se refleja en la Tabla 4.9. Este mal funcionamiento se debe a que la mayoría de relaciones existentes son de hiponimia, es decir, se relaciona palabras de carácter más específico con otras más generales, como *carcinoma epidermoide* que es hipónimo de *tumor*, o de sinonimia, como *fiebre* y *pirexia*. Esto fomenta el aumento de elementos que se estiman negativos erróneamente, ya que en textos donde no se haga mención de dos elementos que sean sinónimos o que sean de la misma jerarquía serían estimados como negativos.

En el siguiente texto de ejemplo, los términos *fiebre*, *amigdalitis bacteriana* y *amoxicilina*, serían clasificados como negativos, aun existiendo en el conjunto de datos, ya que no pertenecen a la misma clase jerárquica ni son sinónimos. Sin embargo, están claramente relacionados, siendo *fiebre* un síntoma de *amigdalitis bacteriana* y *amoxicilina* un posible tratamiento para *amigdalitis bacteriana*:

*"Paciente acude por persistencia de **fiebre** tras ser diagnosticado de **amigdalitis bacteriana**. Refiere haber tomado **Amoxicilina** 1/8h durante sólo 4 días. Se aconseja terminar el tratamiento (7-10 días)"*

		SNOMED-CT Detección	
		Positivo	Negativo
Real	Positivo	84	298
	Negativo	33	59

Tabla 4.8: Matriz de confusión del *baseline* usando las relaciones entre términos como contextualización con SNOMED-CT.

	Distancia entre códigos	
	SNOMED-CT	**DeCS**
Precisión	0.78	-
Recall	0.22	-

Tabla 4.9: Métricas de evaluación del detector *baseline* con relaciones entre términos como contextualización.

Tras tratar de investigar posibles alternativas para la contextualización del texto, no se ha encontrado ninguna manera de hallar un contexto claro con los datos facilitados. Una alternativa planteada consiste en establecer una relación jerárquica manual entre las clases en las que están divididas los términos (ya que se ha visto que la jerarquía proporcionada por SNOMED no genera buenos resultados), de manera que sólo se estimen como positivos los elementos de una clase si se ha detectado algún otro elemento de una clase superior. El problema radica en que las clases son demasiado genéricas y la relación jerárquica a aplicar es demasiado pobre como para obtener resultados eficaces. Se trató de profundizar un nivel en la relación de jerarquía proporcionada por SNOMED-CT para obtener un desglose más específico, pero el número de clases en este segundo nivel se eleva a más de 22 mil, haciendo imposible una relación jerárquica manual.

Por tanto, el resto de pruebas realizadas con los dos detectores restantes se realizarán sin ningún tipo de contextualización, ya que como se ha visto, con las pruebas realizadas la eficacia de los resultados disminuye.

4.3.2. Lematización

En la Tabla 4.10 se muestran las matrices de confusión de los resultados de detección en los 10 textos tras procesar y lematizar cada texto y término. Se puede apreciar cómo el número de elementos detectados ha aumentado con respecto al *baseline*, reduciendo el número de Falsos Negativos, es decir, términos no encontrados. En el conjunto de datos de DeCS se puede apreciar en la Tabla 4.11 cómo los resultados son bastante inferiores, habiendo un número de elementos detectados erróneamente (Falsos Positivos) mucho mayor que el *baseline* en relación con SNOMED-CT, que se ha mantenido casi constante. Esto se debe a que los datos de DeCS contienen más términos metodológicos como *en la consulta* o *al diagnóstico*, que no son estrictamente médicos y que, al transformarlos, se obtienen términos muy generales como *consultar* o *diagnosticar*, que la mayoría de veces se detectan erróneamente. Además, cabe mencionar que, en ambos conjuntos de datos, hay algunos términos que en la mayoría de las ocasiones se trata de palabras no médicas, como *varios(sustancia)* o *triple(fármaco de uso clínico)* que se detectan y en casi todas las ocasiones son falsos positivos, lo que reduce la precisión del

detector.

		SNOMED-CT	
		Detección	
		Positivo	Negativo
Real	Positivo	265	90
	Negativo	85	0

		DeCS	
		Detección	
		Positivo	Negativo
Real	Positivo	187	121
	Negativo	147	0

Tabla 4.10: Matrices de confusión del detector tras lematizar el contenido.

	Lematización	
	SNOMED-CT	**DeCS**
Precisión	0.76	0.56
Recall	0.75	0.6

Tabla 4.11: Métricas de evaluación del detector tras lematizar el contenido.

Por otro lado, una gran parte de los términos no detectados son siglas de procedimientos diagnósticos, terapéuticos, estructuras corporales o enfermedades. Las siglas son muy difíciles de gestionar, ya que los médicos suelen usarlas muy frecuentemente y no existe una estandarización clara; una misma sigla puede referirse a varios conceptos diferentes, o usarse en situaciones diferentes, dependiendo del médico, como por ejemplo *TEP* puede hacer referencia tanto a *Tromboembolismo Pulmonar* como a *Triángulo de Evaluación Pediátrica*. La aparición de siglas en ambos diccionarios es muy reducida, mientras que en los textos clínicos usados como prueba aparecen con frecuencia, lo que reduce la calidad de los resultados. Sin tener en cuenta las siglas, los falsos negativos de SNOMED-CT se reducen a 62, obteniendo un recall de 0.81.

Por último, no se puede pasar por alto el hecho de la ausencia de nombres comerciales de medicamentos en ambos conjuntos de datos, por lo que hay tratamientos que no se detectan como tal, al no estar escritos por el principio activo del fármaco, que sí se recoge.

En general, los resultados de SNOMED-CT son ligeramente superiores con respecto al *baseline* y presentan un mejor funcionamiento que DeCS, aunque la aparición reducida de siglas en el diccionario de términos y la detección

errónea o la no detección de algunos elementos, según lo explicado anteriormente, reduce la calidad final obtenida.

4.3.3. Detector aproximado

En la Tabla 4.12 se muestran las matrices de confusión de la prueba del detector aproximado con ambos conjuntos de datos. Se aprecia un aumento en el número de elementos correctamente detectados, y como consecuencia el número de no detectados se ha reducido. En contraste, el número de elementos detectados incorrectamente ha aumentado considerablemente debido a que este método da pie a muchos errores en palabras como, por ejemplo, *hombre* y *hombro*, que son dos términos con un significado semántico diferente, pero muy parecidos dada su raíz, por lo que este detector detectaría *hombre* si en el texto aparece la palabra *hombro*.

		SNOMED-CT	
		Detección	
		Positivo	Negativo
Real	Positivo	293	62
	Negativo	291	0

		DeCS	
		Detección	
		Positivo	Negativo
Real	Positivo	215	93
	Negativo	120	0

Tabla 4.12: Matrices de confusión del detector aproximado.

Como consecuencia, y tal y como se refleja en la Tabla 4.13, el *recall* ha aumentado considerablemente en relación con los dos detectores anteriores. Sin embargo, debido a estos errores en la aproximación de palabras, la precisión ha disminuido vertiginosamente, sobre todo en el caso de SNOMED-CT, ya que al contener muchos más términos que DeCS la probabilidad de presentar palabras parecidas pero incorrectamente detectadas es mucho mayor.

	Detector aproximado	
	SNOMED-CT	**DeCS**
Precisión	0.50	0.64
Recall	0.83	0.70

Tabla 4.13: Métricas de evaluación del detector aproximado.

4.3.4. Evaluación de los resultados

Tras observar los resultados obtenidos de las distintas pruebas, se puede apreciar cómo el conjunto de datos de SNOMED-CT en general proporciona mejores resultados que DeCS, cosa que era de esperar debido a que el contenido de SNOMED-CT es más amplio y completo.

En cuanto a las distintas pruebas de los detectores, los resultados no son lo esperado inicialmente en ninguna de ellas, siendo el detector tras aplicar lematización el que proporciona unos resultados más equilibrados en cuanto a precisión y *recall*. Esto es debido, por un lado a una falta de contextualización del texto que da pie a que algunos términos no encajen dentro de un determinado contexto, y por otro lado a la ausencia de términos entre ellos, siglas y nombres comerciales de medicamentos.

En resumen, los resultados de la detección se ven mermados debido a la ausencia de términos en el conjunto de datos; la eficacia del detector se ve limitada por los términos del diccionario, hay palabras como *osteofitosis* que no se detectan debido a que no aparecen en el diccionario de términos, incluyéndose aquí también el problema ya mencionado de los nombres comerciales de los medicamentos, a la aparición de términos demasiado generales que la mayoría de veces son incorrectos debido a una falta de contextualización, la mayoría de ellos en formatos como *preposición + verbo* o *conjunción + determinante + verbo*, que al lematizar y eliminar palabras vacías se generalizan demasiado, y a la falta de siglas en los diccionarios de términos, reduciendo la eficacia de los resultados ya que en los textos de prueba la aparición de siglas es frecuente (35 ocurrencias en los 10 textos de pruebas).

Capítulo 5

Detector de recomendaciones

De forma paralela al desarrollo del detector de entidades, se han investigado posibles técnicas para la elaboración de un clasificador de textos que sea capaz de identificar si en un fragmento de texto se está hablando de una recomendación, sugerencia o evidencia dentro de un contexto médico, o es simplemente informativo o divulgativo, siguiendo la misma metodología CRISP-DM descrita anteriormente. En la Sección 5.1 se muestra el análisis realizado, en la Sección 5.2 se detalla el modelado diseñado y finalmente en la Sección 5.3 se explican los resultados obtenidos.

5.1. Análisis

Esta fase engloba las tareas de estudio y comprensión de los datos y preparación de los datos de la metodología CRISP-DM. Para esta tarea se cuenta con una herramienta proporcionada por los directores del TFM que extrae textos de documentos en formato PDF, dividiendo el contenido del documento por frases. También se han facilitado 65 guías clínicas en PDF que han sido procesadas por la herramienta. Por lo que el conjunto de datos disponible consiste en un listado de frases extraída de estos protocolos.

Aunque el funcionamiento de la herramienta es correcto, también extrae elementos no deseados como bibliografía, algunos títulos o textos de marcas de agua, por lo que el conjunto de datos puede contener una cantidad no muy elevada de ruido. Tras procesar las guías clínicas, se obtiene un conjunto de datos formado por 58.864 frases.

Para poder evaluar la calidad de las técnicas de clasificación a aplicar es necesario contar con un conjunto de datos clasificados, por ello el primer paso dado ha sido etiquetar manualmente un subconjunto de estos textos extraídos, diferenciándolos entre 'recomendación' o no. Para esta tarea se ha hecho una especie de *crowdsourcing*, en el que compañeros han ayudado a etiquetar un subconjunto de datos para lograr conseguir más textos clasificados. En total se han conseguido 5 anotadores contando el propio alumno.

Nótese que la clasificación entre 'recomendación' y 'no recomendación' en algunos casos puede no ser trivial, ya que la concepción de lo que es recomendación entre varías personas puede diferir en ciertos matices, por lo que los textos clasificados obtenidos pueden estar sesgados por esta forma de interpretación de cada anotador en particular.

Para valorar el acuerdo entre los anotadores, se han seleccionado al azar 100 textos y han sido clasificados por todos ellos. El índice de acuerdo se ha calculado mediante un promedio de las coincidencias de la clasificación entre todos los anotadores, usando la siguiente fórmula:

$$\acute{I}ndice\ de\ acuerdo = \frac{N\ acuerdos}{N\ elementos}$$

Finalmente, se obtiene un índice de acuerdo de 0.63, lo que refleja cómo en ciertas frases varía el decidir si algo es una recomendación o no. Algunos casos conflictivos son los siguientes, donde unos anotadores consideraron que no se estaba recomendando o sugiriendo nada y otros determinaron que sí:

1. *"La presentación de TAG más complejos y graves en el inicio, el fracaso en completar el tratamiento y la cantidad de tratamientos intermedios durante el período de seguimiento se asocian con peores resultados de la TCC a largo plazo."*

En este texto, extraído de una de las guías clínicas utilizadas, el conflicto entre los observadores radicó en el hecho de la presencia, en el texto, de una exposición de resultados. Algunos pensaron que por el mero hecho de exponer un resultado, ya estaban recomendando (de forma positiva o negativa, como sería el caso) por lo que lo calificaron como recomendación. Por otro lado, otros llegaron a la conclusión de que el mero hecho de reflejar resultados no era una clara recomendación, por lo que no lo consideraron como recomendación.

2. *"Un modelo integrado en el que los médicos de familia son apoyados por especialistas, que durante 8 semanas (4-8 sesiones) ayudan a los pacientes a desarrollar habilidades cognitivo-conductuales a través de relajación, reconocimiento de pensamientos ansiogénicos y de falta de autoconfianza, búsqueda de alternativas útiles y entrenamiento en acciones para resolución de problemas, técnicas para mejorar el sueño y trabajo en casa."*

   En este caso, las diferentes opiniones sobre si el hecho de desarrollar un concepto, en este caso el modelo integrado, es una recomendación o no provocaron esta discordancia. Por un lado, unos observadores consideraron que el hecho de explicar un término, por mucho que fuese un tratamiento, no era suficiente para calificarlo como recomendación, mientras que otros sí lo entendieron de ese modo.

Gracias a esta anotación, se ha conseguido obtener 4.519 textos clasificados manualmente entre recomendación y no recomendación.

5.2. Modelado

En este apartado se detalla la fase de modelado de la metodología CRISP-DM.

Primero se ha implementado un modelo sencillo que será un *baseline* inicial, y posteriormente se han implementado los algoritmos de aprendizaje automático más populares en minería de textos [11]: *Support Vector Machine (SVM)*, *Naive Bayes (NB)*, *Random Forest (RF)* y *K-Nearest Neightbour (KNN)* .

Como *baseline*, se ha realizado una categorización de verbos que consiste en identificar qué verbos se utilizan con mayor frecuencia en estas recomendaciones médicas usando lenguaje natural y determinar, a partir de la aparición o ausencia de cualquiera de estos verbos en el texto, si es, o no, una recomendación. Se ha decidido centrarse en los verbos porque es la parte de las oraciones que más información proporciona acerca del sentido de la frase en este caso, además de que la lista de sustantivos o adjetivos existente es mucho más amplia y menos fácil de analizar y categorizar ya que son mas susceptibles a la subjetividad, dependiendo del resto de elementos del texto. A continuación, se muestran los verbos elegidos:

actuar	concluir	favorecer	provocar
administrar	consultar	garantizar	recibir
aliviar	contemplar	hallar	recomendar
ayudar	contrarrestar	indicar	solucionar
afrontar	deber	intervenir	sugerir
asesorar	descartar	justificar	suscribir
advertir	demostrar	manifestar	tomar
argumentar	desinfectar	mediar	tratar
barajar	estimar	mejorar	valorar
calmar	evaluar	optar	
combatir	evidenciar	prevenir	
curar	examinar	prever	

Los verbos presentes en los textos pueden estar conjugados en diferentes tiempos verbales, por lo que se ha lematizado el contenido de los textos de igual manera que con los textos del detector.

Por otra parte, para el desarrollo de los modelos de aprendizaje automático se ha hecho uso de la librería *scikit-learn* [10], que cuenta con múltiples modelos y funciones para el preprocesado, clasificación y análisis de resultados de las técnicas utilizadas.

Los algoritmos de aprendizaje automático no pueden trabajar con los textos en crudo directamente, sino que estos textos deben ser convertidos a vectores de números, creando un modelo de espacio vectorial. Las técnicas más comunes para realizar este proceso [33] son *Bag of Words (BoW) y Term*

Frequency - Inverse Document Frequency (TF-IDF).

El primer método, *BoW*, consiste en contar el número de apariciones de cada palabra en el documento. El problema de este método es que no tiene en cuenta el ruido, por lo que palabras que se repiten con frecuencia y que no añaden ningún significado semántico pueden hacer que el modelo empeore sus resultados.

En el segundo método, *TF-IDF*, mide la importancia de cada palabra en el documento, normalizando la frecuencia de cada término *(TF)* en el documento en función de su frecuencia relativa en todo el conjunto de datos *(IDF)*, permitiendo valorar la relevancia de cada palabra en el documento, por lo que se ha elegido como métrica para la vectorización de los textos. Este valor se calcula mediante las siguientes fórmulas [54]:

Frecuencia del término: número de veces que aparece una palabra en un documento dividido por el total de palabras en el documento.

$$TF_{i,d} = \frac{|\{i \in d\}|}{|d|}$$

Donde $|\{i \in d\}|$ es el número de apariciones de la palabra i en el documento d y $|d|$ es el número de palabras del documento.

Frecuencia inversa de documento: medida que indica si una palabra es frecuente en la colección o no:

$$IDF_i = log\frac{|D|}{1 + |\{d \in D : i \in d\}|}$$

Donde $|D|$ es el número de documentos en la colección y $|\{d \in D : t \in d\}|$ es el número de documentos en los que aparece la palabra i. Para evitar división por cero se suma uno en el denominador.

Finalmente, la fórmula para calcular el valor TD-IDF es la siguiente:

$$TF\text{-}IDF_{i,j} = TF_{i,f} * IDF_i$$
$$TF\text{-}IDF_{i,j} = \frac{|\{i \in d\}|}{|d|} * log\frac{|D|}{1 + |\{d \in D : i \in d\}|}$$

El valor *TF-IDF* aumenta en función de la frecuencia de un término en el texto, pero se compensa por la frecuencia en la colección de documentos, lo que permite manejar el hecho de que algunas palabras sean más comunes que otras. Por otra parte, un inconveniente de este método es que no tiene en cuenta la posición de las palabras, sino que únicamente asigna valores a cada una de ellas de forma individual, y en algunos casos la posición de las palabras sí que puede tener cierta relevancia. En resumen:

- Si una palabra aparece con mucha frecuencia en muchos documentos, la importancia se reduce.
- Si una palabra aparece con mucha frecuencia en un conjunto reducido de documentos, la importancia aumenta.
- Si una palabra aparece con poca frecuencia, la importancia se reduce.

Una vez vectorizados los textos, ya se pueden aplicar como datos de entrenamiento a los modelos, los cuales tienen diferentes parámetros que se puede ajustar para obtener mejores resultados dependiendo del objetivo de clasificación.

Support Vector Machine [47]: trata de determinar el mejor hiperplano en un espacio euclídeo que separe los datos de entrenamiento en sus respectivas clases. Una buena forma de seleccionarlo es elegir el hiperplano que deje el máximo margen entre las clases, donde este margen es definido como la suma de las distancias al hiperplano de los elementos más cercanos a éste, como se puede apreciar en la Figura 5.1 donde se espera una mayor generalización en el hiperplano con margen más amplio (b). A continuación, se detallan los parámetros utilizados en las pruebas a realizar [7]:

- C: Determina el nivel de optimización del modelo, de cuánto se desea evitar clasificar cada dato de entrenamiento incorrectamente. Al ser

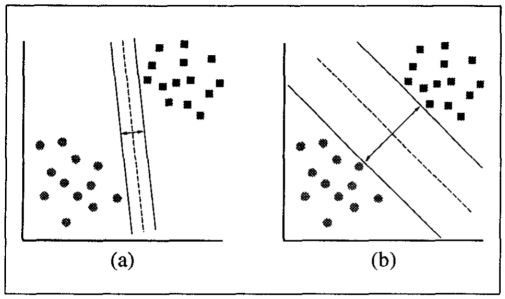

Figura 5.1: Ejemplo de posibles hiperplanos del modelo SVM. Extraída de [47].

más elevado, menor será el margen del hiperplano seleccionado. Valores muy altos de este parámetro pueden derivar en un sobreajuste de los datos de entrenamiento.

- *Kernel*: indica el tipo de hiperplano a utilizar para separar los datos. Con la opción *'linear'* se utilizará un hiperplano lineal y con *'poly'* se calcularán hiperplanos polinomiales.
- *Degree*: es un parámetro utilizado cuando el kernel es *'poly'*. Indica el grado del polinomio utilizado para calcular el hiperplano.

Multinomial Naive Bayes [57]: es un modelo probabilístico basado en el teorema de Bayes, muy usado para la clasificación de documentos. Según este teorema, la probabilidad de que un documento $d \in D$ pertenezca a una clase c se calcula de la siguiente forma:

$$P(c_i|d_j) = \frac{P(c_i) * P(d_j|c_i)}{P(d_j)}$$

La probabilidad de un documento $P(d_j)$ no aporta información, al ser una constante, por lo que suele omitirse. Este modelo asume que todos los predictores, en este caso palabras, son independientes entre sí, por lo que la probabilidad de un documento dada su clase $P(d_j|c_i)$ puede calcularse como la probabilidad conjunta de todos los términos $w \in d_j$ del documento dada su clase:

$$P(d_j|c_i) = \prod_{t=1}^{|D|} P(w_t|c_i)$$

Adicionalmente, este modelo tiene en cuenta la frecuencia de cada término en el documento x_t en vez de considerar únicamente si el término aparece o no. Finalmente, la clasificación se realiza buscando el argumento que maximiza la siguiente función:

$$c^*(d) = argmax_{ci} p(c_i) \prod_{t=1}^{|D|} P(w_t|c_i)^{x_t}$$

Random Forest [9]: consiste en construir un conjunto independiente de árboles de decisión, basados en reglas. Este modelo está basado en dos conceptos clave:

1. Muestreo aleatorio de datos de entrenamiento al construir árboles: cada árbol aprende de un subconjunto de datos aleatorio extraído con reemplazo, conocido como *bootstrapping*, lo que significa que cada muestra se utilizará únicamente en un árbol.
2. Subconjuntos aleatorios de características al dividir los nodos: en cada árbol, la reglas se aplican seleccionando un subconjunto aleatorio de características.

La predicción final se realiza mediante un promedio de las predicciones individuales de cada árbol. El número de árboles de decisión puede modificarse mediante el parámetro *n_estimators* de *scikit-learn*.

Las ventajas de este modelo se basan en que, al realizar el promedio de la predicción de múltiples árboles no correlacionados entrenados con distintos subconjuntos de datos, se reduce la varianza del modelo, aumentando la precisión, además de reducir la probabilidad de sobreajuste. Como desventajas, este modelo es difícil de interpretar, siendo una especie de caja negra, además de que requiere un elevado tiempo de ejecución y memoria comparado con otros clasificadores.

K-Nearest Neightbours [2]: consiste en basar la clasificación de una instancia a partir de las clases de los K elementos más próximos a esta instancia seleccionando la clase mayoritaria de los vecinos, en ocasiones, ponderando

los votos de acuerdo a la distancia de cada vecino. La Figura 5.2 muestra una descripción gráfica de este modelo, donde la instancia del centro es clasificada como *rojo* con K = 3, al tener, de sus tres vecinos más próximos, dos *rojo* y uno *verde*. Sin embargo, para un valor de K = 5, 3 vecinos son *verde* y dos *rojo*, por lo que sería clasificada como *verde*.

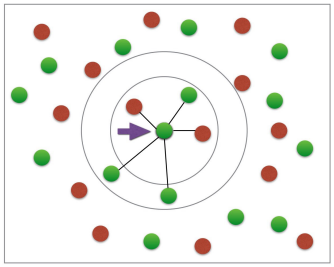

Figura 5.2: Ejemplo de modelo KNN. Extraída de [2].

Al transformar los documentos a un modelo *TF-IDF*, este algoritmo puede calcular las distancias entre los documentos usando estos vectores. Existen múltiples formas de calcular la distancia entre dos elementos [14]. Entre ellas, se han seleccionado las siguientes, disponibles en la librería *scikit-learn*:

- Distacia Euclídea: es la distancia más popular, que se basa en el teorema Pitágoras, midiendo la distancia como la longitud del segmento que une los dos puntos y que se calcula con la siguiente fórmula:

$$Euclidean(a,b) = \sqrt{\sum_{i=1}^{n}(x_{ai} - x_{bi})^2}$$

- Distancia de Manhattan: se calcula sumando las distancias absolutas de todas las respectivas coordenadas cartesianas de las dos instancias, mediante la siguiente fórmula:

$$Manhattan(a,b) = \sum_{i=1}^{n} |x_{ai} - x_{bi}|$$

Otras métricas muy conocidas, como la distancia del coseno o *Jaccard*, no han podido ser aplicadas al no estar disponibles en esta librería. A parte de

la métrica utilizada para calcular la distancia, también se ha empleado el parámetro K, que determina el número de vecinos más próximos a tener en cuenta para las pruebas a realizar.

5.3. Resultados

Para evaluar los resultados de los modelos se ha hecho uso de la técnica de validación cruzada [59], que consiste en dividir los datos de entrenamiento en un número fijo de subconjuntos, determinados por un parámetro k, usando iterativamente uno para testear y el resto como entrenamiento. De esta manera se asegura que los resultados son independientes de la partición entre datos de entrenamiento y prueba. En la Figura 5.3 se muestra un ejemplo gráfico.

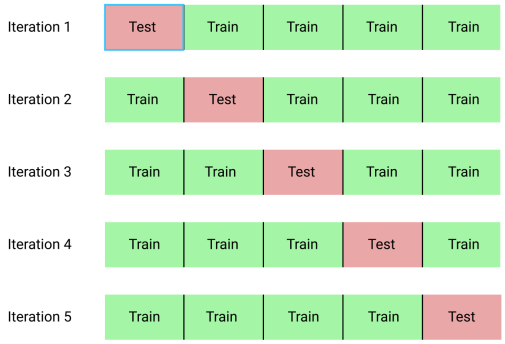

Figura 5.3: Validación cruzada con k = 5. Extraída de [59].

En la Tabla 5.1 se muestran los resultados obtenidos al aplicar los modelos descritos, utilizando validación cruzada con $k = 5$. Se aprecia cómo los resultados obtenidos son más elevados de lo que se esperaba con un índice de acuerdo entre anotadores no muy alto.

Los resultados reflejan cómo la categorización de verbos no es muy efectiva al compararse con el resto de clasificadores. Se observa un aumento de la precisión y una disminución del *recall* a medida que se reduce el parámetro C en el modelo SVM lineal. Usando un modelo no lineal, los resultados

	Precisión	**Recall**
Categorización de verbos	0.64	0.49
SVM-linear C=1	0.84	0.79
SVM-linear C=0.5	0.85	0.75
SVM-linear C=0.1	0.93	0.46
SVM-poly C=1 Degree=2	0.46	0.65
Naïve Bayes	0.81	0.73
RF n_estimators=1000	0.86	0.78
RF n_estimators=2000	0.86	0.78
KNN-3 Distancia Euclídea	0.77	0.84
KNN-5 Distancia Euclídea	0.78	0.83
KNN-3 Distancia Manhattan	0.94	0.39
KNN-5 Distancia Manhattan	0.91	0.43

Tabla 5.1: Resultados de los diferentes modelos para clasificar recomendaciones.

empeoran significativamente, por lo que ha sido descartado. El modelo KNN funciona mejor usando la distancia Euclídea frente a la de Manhattan, no habiendo mucha diferencia entre usar 3 vecinos como referencia a usar 5. También puede verse cómo los resultados del modelo *Random Forest* no se ven alterados al aumentar el número de árboles.

Finalmente, para el despliegue final, se ha escogido el modelo Random Forest con 1000 árboles de decisión aleatorios al presentar unos resultados ligeramente superiores al resto de modelos.

Capítulo 6

Despliegue

Como despliegue en la metodología CRISP-DM, se ha desarrollado una interfaz de usuario a modo de prueba de concepto implementada en *Python*. En ella, el cliente tiene la posibilidad de escribir un texto en lenguaje natural que será analizado por la aplicación, mostrándose los términos encontrados en el mismo junto con la determinación de si es una recomendación o no.

Como detector para esta aplicación, se ha utilizado el que mejores resultados ha mostrado, el detector con lematización y eliminación de palabras vacías. Como clasificador para recomendaciones, se ha empleado *Random Forest*, siendo de nuevo el que mejores resultados ha obtenido.

En la Figura 6.1, se observa un texto de ejemplo introducido en la pantalla de inicio de esta interfaz. A su vez, en la Figura 6.2 se reflejan los resultados obtenidos, es decir, tanto los términos detectados como la indicación de si es o no una recomendación.

Ha sido necesario implementar lógica adicional para devolver el texto original etiquetado, ya que el detector trabaja con el texto procesado en el que se han eliminado palabras vacías y signos, se ha lematizado el contenido y convertido a minúsculas.

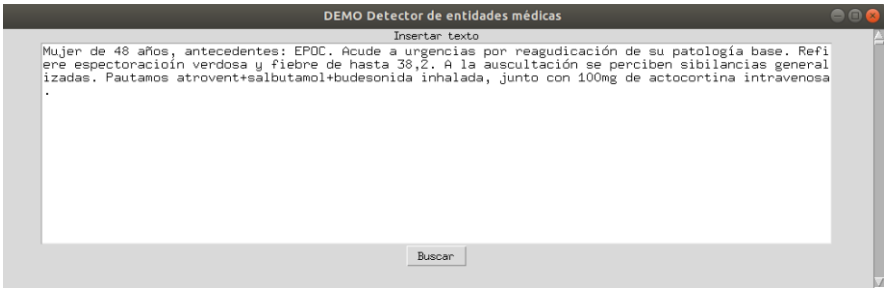

Figura 6.1: Pantalla de inicio de la aplicación.

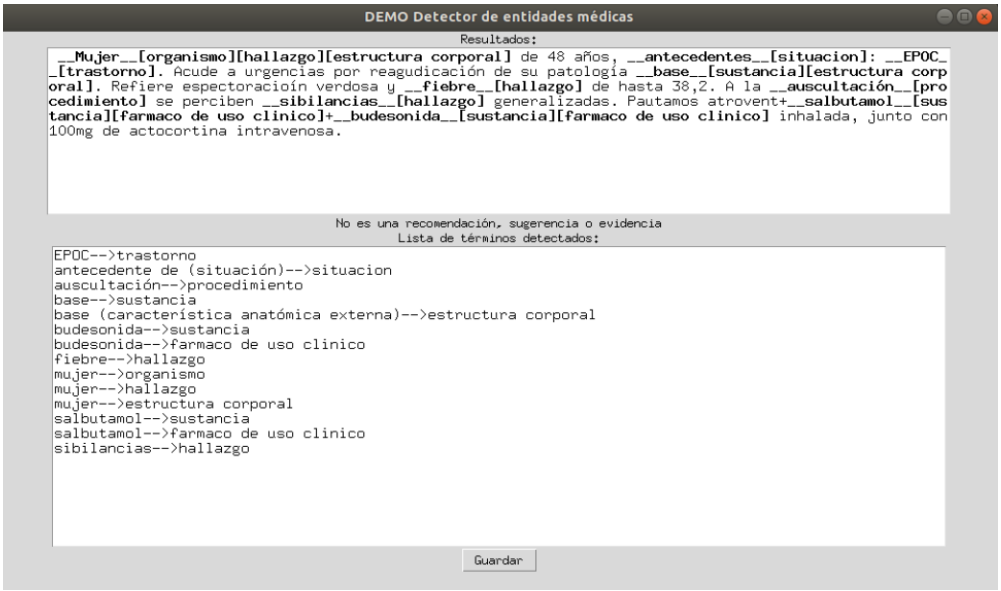

Figura 6.2: Pantalla de resultados de la aplicación.

Capítulo 7

Conclusiones y trabajo futuro

En este Capítulo se detallan las conclusiones del proyecto desarrollado y se valoran posibles líneas futuras de trabajo. En la Sección 7.1 se detallan las conclusiones del trabajo tanto profesionales como personales. En la Sección 7.2 se explican posibles líneas de trabajo futuro.

7.1. Conclusiones

El objetivo principal de este Trabajo de Fin de Máster ha consistido en elaborar una aplicación con dos funciones principales. Por un lado, la capacidad de detectar términos médicos escritos en un texto redactado por un individuo en lenguaje natural. Por otro lado, la identificación adecuada de si dicho texto hace referencia a un acto recomendado, basado en la evidencia científica, o si, por el contrario, es un texto meramente informativo sin contenido de recomendación diagnóstica o terapéutica.

Para conseguir estos objetivos, se ha realizado un arduo trabajo analizando diferentes técnicas de minería de texto y basándose en dos conjuntos de datos en forma de diccionario previamente formados, obtenidos de las fuentes SNOMED-CT [60] y DeCS [20]. Además, se ha requerido el uso de numero-

sas guías clínicas para entrenar el modelo de detección de recomendaciones y lograr diferenciar entre lo que es una recomendación y lo que no lo es. Conjuntamente se ha requerido de varias historias clínicas anonimizadas para poder comprobar la funcionalidad de los diferentes detectores y qué aspectos podían ir mejorándose durante la realización del trabajo.

Finalmente, se ha conseguido alcanzar los objetivos descritos dentro de los plazos indicados en el cronograma inicial. Se ha logrado obtener una herramienta capaz de encontrar términos en un texto escrito en lenguaje natural en castellano con una calidad aceptable teniendo en cuenta las limitaciones de los conjuntos de datos disponibles (precisión y *recall* de 0,76 y 0,74 respectivamente). Cabe mencionar que el desarrollo este módulo ha requerido más tiempo del que se preveía inicialmente, en mayor medida debido a la inexperiencia en este ámbito, teniendo que dedicar horas extra, cuando era posible, tanto a la formación como a la aplicación de lo aprendido para poder alcanzar los objetivos del proyecto.

Así mismo, se ha logrado crear un modelo que es capaz de detectar intención de recomendación en una guía clínica con unos valores de precisión y *recall* de 0,86 y 0,78, respectivamente. Este módulo ha requerido un menor número de horas invertidas para su realización debido a la aparición de un número inferior de dificultades en su elaboración.

Los resultados finales del proyecto han sido positivos, a pesar de las múltiples dificultades que ha supuesto realizar este trabajo con datos en castellano, y de los problemas encontrados en los conjuntos de datos que se han podido obtener, lo que ha reducido parcialmente la efectividad final del detector.

Como conclusiones personales, el resultado final también ha sido satisfactorio. Este trabajo ha proporcionado al alumno nuevas competencias al desarrollar el proyecto en un ámbito del que no se tenía conocimiento previo, y ha ayudado a ampliar sus aptitudes tanto personales como profesionales.

7.2. Trabajo futuro

De cara a un trabajo futuro, estos módulos desarrollados podrían ser utilizados en la herramienta de búsqueda de consultas médicas descrita en el Capítulo 1. A partir del detector de recomendaciones, podrían priorizarse los resultados de este buscador que se estimen como recomendación. Por otra parte, el detector de entidades médicas podría utilizarse como base para determinar que secciones de las guías clínicas encajan con la búsqueda escrita por el usuario mediante los términos detectados.

Otro posible avance podría consistir en investigar posibles alternativas para lograr contextualizar el texto para mejorar la calidad de los resultados, ya sea investigando diferentes opciones no contempladas en este proyecto, o generando conjuntos de datos alternativos con esta información de contexto mejor estructurada. Las modificaciones de la aplicación para adaptarla a un nuevo conjunto de datos serían mínimas y podrían mejorar la precisión y el *recall* del detector. También se podría enriquecer el diccionario añadiendo los nombres de todos los medicamentos comerciales asociados a sus principios activos así como las siglas asociadas a sus respectivos elementos.

Por último, otro experimento que podría ser interesante contemplar, consiste en usar técnicas de *Deep Learning* [64] para el desarrollo del modelo de detección de recomendaciones. Las redes neuronales son un sistema de clasificación que está actualmente en auge y utilizado en múltiples aplicaciones de minería de textos y otros ámbitos con muy buenos resultados, pero debido a que estos modelos necesitan un tiempo de entrenamiento muy elevado y una mayor cantidad de datos en comparación con los que se han desarrollado no ha sido posible utilizarlo en este proyecto.

Se espera que el trabajo desarrollado sirva como base para futuros trabajos e investigaciones llevadas a cabo por grupos de investigación de la Universidad de Zaragoza, en particular el grupo COSMOS (Computer Science for Complex System modelling), así como al desarrollo de la Biblioteca Inteligente GUÍA SALUD [29] (proyecto BIGS [30]). Además, podrían servir también de aplicación en contextos docentes, como base de estudio en asignaturas como Manipulación y Análisis de Grandes Volúmenes de Datos del Máster Universitario en Ingeniería Informática.

Anexos

Anexo A

Descripción del procesamiento de datos de SNOMED-CT

SNOMED CT (Systematized Nomenclature of Medicine – Clinical Terms) [60] es una terminología clínica integral originalmente desarrollada en inglés. Ha sido traducida, posteriormente, a otros idiomas, entre ellos el español. Actualmente contiene más de un millón de términos, denominados conceptos, traducidos al español. A su vez, estos términos presentan ciertas relaciones predefinidas entre ellos, es decir, que sólo permiten cierto tipo de relaciones dependiendo de la clase de ambos términos. Una de las relaciones más frecuentes es la de sinonimia, en la que un concepto es denominado como principal o preferido (haciendo alusión al término más utilizado), y todos los conceptos que hagan referencia a este principal estarán relacionados con él. Por ejemplo *deglute rápidamente (hallazgo)* y *traga rápidamente* son dos hallazgos relacionados como sinónimos, siendo el primero el principal.

Este conjunto de datos se puede obtener en su totalidad desde la página del Ministerio de Sanidad tras realizar una petición de licencia. Se divide en múltiples ficheros con varias extensiones. Para este caso, únicamente es necesario el fichero donde se almacenan todos los conceptos con su respectiva clase asociada y el fichero de relaciones obtenido de la versión en inglés, ya que en la versión en castellano no se incluye dicho fichero.

En este fichero se informa de los términos de SNOMED, uno por línea, estando los campos separados por tabulaciones bajo la estructura detallada en la Tabla A.1.

Campo	**Valor de ejemplo**	**Descripción**
id	845114013	Id único asociado a cada término
effectiveTime	20031031	Fecha de inserción
active	1	Activo (1) o inactivo (0)
moduleId	450829007	Igual para todos. No relevante
conceptId	100005	Id único asociado a cada concepto. Un concepto puede tener varios términos
languageCode	es	Término en español
typeId	900000000003001	Indica si es el término principal o preferido del concepto (900000000003001) o es un término semejante (900000000013009)
term	concepto de SNOMED CT	Texto con el nombre del término

Tabla A.1: Descripción de campos informados en el fichero de términos de SNOMED-CT.

Tras analizar el fichero, el primero paso a realizar es la separación de conceptos por clase en diferentes carpetas para poder analizar cada una de estas clases más detenidamente y así observar mejor la posible aportación de cada clase. En el fichero no viene informado un campo con la clase de cada término; sin embargo, en el nombre de los términos principales aparece entre paréntesis el nombre de la clase de nivel superior en la jerarquía a la que

pertenecen, por lo que a partir de esta información ha sido posible obtener la clase de los términos.

Para obtener la clase de los términos que no son principales se ha realizado una búsqueda de su término principal por *conceptId*, donde el identificador *typeId* sea el asociado a los términos principales (*900000000000003000*), y extrayendo el texto que tiene este término principal entre paréntesis. En la Tabla A.2, se muestran dos ejemplos de términos que hacen referencia al mismo concepto, siendo el primero el término preferido o principal, y el resto sinónimos.

conceptId	**termTypeId**	**term**
64882008	900000000000003000	fiebre (hallazgo)
64882008	900000000000013009	temperatura corporal elevada
64882008	900000000000013009	pirexia

Tabla A.2: Relación entre términos principales y sinónimos.

Finalmente, se han obtenido 98 clases diferentes; el principal problema de esta división es la similitud que hay entre algunas clases, así como la poca relevancia que tienen algunas de ellas para el dominio de este proyecto. Por ejemplo, hay dos clases, *trastorno* y *hallazgo*, que parecen hacer referencia a enfermedades, pero con matices ligeramente distintos. Hay clases como sustancia, donde la mayoría de elementos están registrados con su fórmula química (*2,3-dihidroxibenzoato 3,4-dioxigenasa (sustancia)*) en la que rara vez se va a encontrar algo así escrito, por lo que su utilidad para esta tarea es mínima. Por otro lado, se encuentran clases que contienen términos del mismo tipo separadas por errores ortográficos, como es el caso de estructura corporal al aparecer escrito de cinco formas diferentes (*estrcutura coporal*, *estrcutura corporal*, *estructura coporal*, *estructura corporal* y *erstructura corporal*).

Tras analizar estas clases, se han eliminado algunas de contenido irrelevante, como *atributos* (*debido a*, *con*, *que causa*, etc.), y se han juntado otras clases de contenido similar como *medicamento clínico* y *fármaco de uso clínico*. Cabe mencionar que en una de las clases, la de fármacos de uso clínico, había multitud de términos repetidos, pero descritos de forma demasiado específica, por lo que sería muy poco probable que un detector pudiera llegar

a identificar estos términos. A continuación, se muestran algunos ejemplos:

compuesto con lactato de sodio, solución para infusión, bolsa de 1 l
compuesto con lactato de sodio, solución para infusión, bolsa de 1 l (fármaco de uso clínico)
compuesto con lactato de sodio, solución para infusión, bolsa de 500 ml
compuesto con lactato de sodio, solución para infusión, bolsa de 500 ml (fármaco de uso clínico)
compuesto con lactato de sodio, solución para infusión, frasco de polietileno de 1 l
compuesto con lactato de sodio, solución para infusión, frasco de polietileno de 1 l (fármaco de uso clínico)
compuesto con lactato de sodio, solución para infusión, frasco de polietileno de 500 ml
compuesto con lactato de sodio, solución para infusión, frasco de polietileno de 600 ml

Todos estos casos repetidos se han agrupado en un único término, mediante expresiones regulares, con el nombre del fármaco, en este caso *lactato de sodio*.

A continuación, se explica qué campos se han utilizado para crear el conjunto de datos con el que se realiza el trabajo de detección de entidades:

- Id: no es relevante, ya que simplemente es un identificador único para cada término que no aporta ningún tipo de información.
- effectiveTime: la fecha de inserción no es relevante.
- active: se han seleccionado sólo los términos activos. Los términos inactivos, son elementos ya repetidos (que existe ya un término con ese nombre con el parámetro active a 1).
- moduleID: no relevante, ya que es el mismo identificador para todos los términos.
- conceptId: identificador del concepto. Se ha seleccionado para saber con qué concepto está relacionado cada término.

- languageCode: no es relevante, ya que son todo términos en español.
- typeId: se ha seleccionado para saber cuál es el término principal y cuál es un sinónimo. En todos los términos principales se informa entre paréntesis a qué clase corresponde, por lo que a partir de este identificador se sabrá de qué términos extraer la clase.
- term: es necesario para saber el nombre de los términos.

Finalmente, el resultado es de 18 clases con 951.213 términos.

El contenido de estas clases se ha almacenado en una base de datos MongoDB [41] mediante un script en *Python* para un acceso mas fácil y rápido a éstos bajo la estructura de la Tabla A.3.

Campo	**Valor de ejemplo**	**Descripción**
conceptId	6232195	Id único asociado a cada concepto. Un concepto puede tener varios términos.
typeId	9000013009	Indica si es el término principal o preferido del concepto o es un término semejante.
term	pirexia	Texto con el nombre del término.
termType	hallazgo	Indica la clase del término.

Tabla A.3: Información almacenada en base de datos.

Anexo B

Descripción del procesamiento de datos de DeCS

DeCS (Descriptors de Ciencias de la Salud) [20] es un vocabulario médico dinámico, totalizando 33.966 descriptores y calificadores, siendo de estos 29.431 de MeSH y 4.535 exclusivamente de DeCS, utilizado para servir como un lenguaje único en la indización de artículos de revistas científicas, libros, anales de congresos, informes técnicos, y otros tipos de materiales, así como para ser usado en la búsqueda y recuperación de asuntos de la literatura científica en las fuentes de información disponibles en la Biblioteca Virtual en Salud, como LILACS, MEDLINE y otras. Al igual que SNOMED, los datos de DeCS se puede obtener solicitando una licencia de uso.

La fuente de datos obtenida de DeCS también contiene múltiples ficheros, de los cuales solamente uno va a ser de utilidad para este trabajo. En él se recogen todos los términos que están incluidos en DeCS traducidos al castellano en formato XML. En la Figura B.1, se muestra un ejemplo de la estructura de un término de DeCS en formato XML.

Cada descriptor tiene asociada una lista de conceptos y cada concepto una lista de términos. Las relaciones son bastante similares a las de SNOMED, habiendo un nivel más, el de *Descriptor*, que agrupa diferentes conceptos, siendo el nombre del descriptor igual al nombre del concepto principal o

preferido. En la Tabla B.1 se describe los campos más relevantes de este fichero.

Campo	**Valor de ejemplo**	**Descripción**
descriptorName	Calcimicina[Calcimycin]	Nombre del descriptor.
descriptorUI	D000001	Id asociado a cada descriptor.
qualifiersAbs	[.ᴬA", .ᴬD", .ᴬE", .ᴬG", .ᴬI"]	Abreviatura de calificadores asociada a cada descriptor.
treeNumberList	["D03.633.100.221.173"]	Id asociado a este concepto en el árbol de jerarquía de DeCS. Estos ids se usan para obtener el Descriptor raíz, que es el que indicará la clase de cada concepto.
conceptName	Calcimicina[Calcimycin]	Nombre del concepto asociado al descriptor.
conceptUI	M0000001	Id asociado a cada concepto.
conceptIsPrefered	Y	Indica si el concepto es el principal o no.
termName	Calcimicina	Nombre del término.
termUI	spa0000586	Id asociado a cada término.
termPrefered	Y	Indica si el término es el preferido o no.

Tabla B.1: Descripción de campos informados en el fichero de términos de DeCS.

Para obtener la clase de cada término, es necesario obtener su "descriptor raíz", es decir, el descriptor de nivel superior en la jerarquía, lo cual es posible gracias al parámetro *treeNumberList* que viene informado, y a una aplicación basada en servicios Web que ofrecen los desarrolladores de DeCS [19], mediante la cual se puede obtener diversa información de un término a través de este parámetro, entre ella los descriptores predecesores de un término.

Mediante el siguiente *enpoint* se puede obtener toda esta información, a partir del campo *treeId* donde se introduce el valor del parámetro *treeNumberList*:

`http://decs.bvsalud.org/cgi-bin/mx/cgi=@vmx/decs/?tree_id=123456`

Este *endpoint* devuelve contenido en formato XML en el que se informan los predecesores (*ancestors*) de un descriptor mediante la estructura de la Figura B.1, en la que se puede apreciar que devuelve múltiples resultados, de los que es necesario filtrar solo los considerados descriptores raíz. En la Tabla B.2 se listan todos los posibles descriptores raíz que ofrece DeCS.

Se ha almacenado la información de este fichero a nivel de término, ya que es la mínima granularidad y la que más información aporta. En la Tabla B.3 se detalla los campos que se han almacenado en MongoDB procesando este fichero mediante un script en *Pyhton*. Finalmente se obtienen 91.823 términos divididos en 20 clases diferentes.

treeId	**Descriptor**
A	ANATOMÍA
B	ORGANISMOS
C	ENFERMEDADES
D	COMPUESTOS QUÍMICOS Y DROGAS
E	TÉCNICAS Y EQUIPOS ANALÍTICOS, DIAGNÓSTICOS Y TERAPÉUTICOS
F	PSIQUIATRÍA Y PSICOLOGÍA
G	FENÓMENOS Y PROCESOS
H	DISCIPLINAS Y OCUPACIONES
HP	HOMEOPATÍA
I	ANTROPOLOGÍA, EDUCACIÓN, SOCIOLOGÍA Y FENÓMENOS SOCIALES
J	TECNOLOGÍA, INDUSTRIA Y AGRICULTURA
K	HUMANIDADES
L	CIENCIA DE LA INFORMACIÓN
M	DENOMINACIONES DE GRUPOS
N	ATENCIÓN DE SALUD
SH	CIENCIA Y SALUD
SP	SALUD PÚBLICA
V	CARACTERÍSTICAS DE PUBLICACIONES
VS	VIGILANCIA SANITARIA
Z	DENOMINACIONES GEOGRÁFICAS

Tabla B.2: Listado de descriptores raíz disponibles en DeCS.

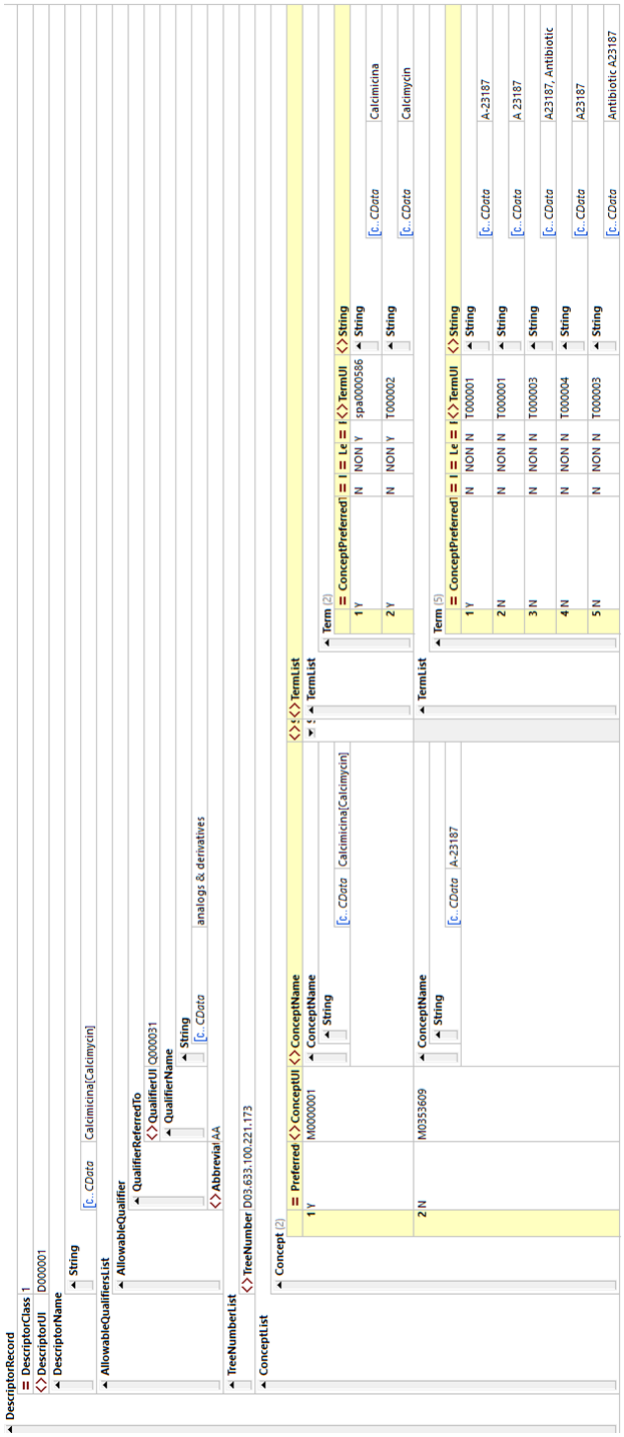

Figura B.1: Estructura de un término de DeCS en formato XML.

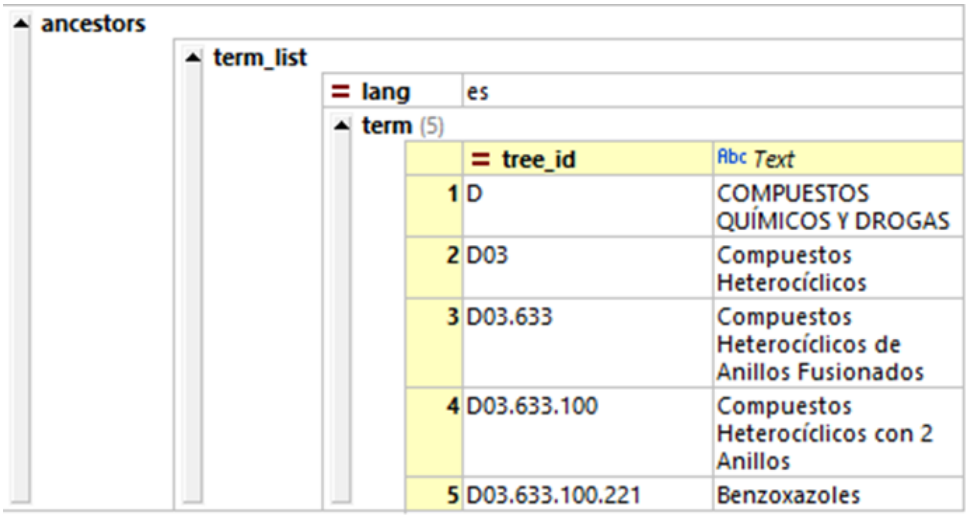

Figura B.2: Estructura de la respuesta de la aplicación DeCS.

Campo	Valor de ejemplo	Descripción
termName	Calcimicina	Nombre del término.
termUI	spa0000586	Id asociado a cada término.
termPrefered	Y	Indica si el término es el preferido o no.
ancestor	COMPUESTOS QUÍMICOS Y DROGAS	Descriptor raíz del término actual. Indica la clase del término.

Tabla B.3: Estructura de la información almacenada en base de datos del fichero de términos de DeCS.

Anexo C

Instalación de Freeling

En el Anexo C.1 se muestran los problemas encontrados al intentar instalar sin éxito una API para usar *Freeling* en *Python*. En el Anexo C.2 se detalla el proceso de instalación bajo un entorno Linux donde si que pudo completarse satisfactoriamente.

C.1. Instalación en Windows

Inicialmente se intentó instalar esta aplicación en un entorno Windows, ya que es el que venía instalado por defecto en el ordenador del alumno. Para ello se han seguido los pasos del manual de instalación disponibles en la documentación de Freeling [23], en donde se explican los requisitos necesarios para poder construir una API que pueda ser utilizada en *Python*.

El primer paso consiste en descargar el código fuente desarrollado en *C++*, disponible en el manual de instalación, para posteriormente compilarlo y crear una API utilizable en *Python*. Adicionalmente, es necesario instalar *MSVC* (*Microsoft Virtual C++*, con compilador para *C++*), *CMake* (usada para construir y empaquetar software) en su versión 3.8 o superior, *zlib* (librería utilizada para la compresión de datos) y *SWIG* (herramienta utilizada

para conectar programas escritos en *C* o *C++* con lenguajes de *scripting*, como en este caso *Python*).

Por último, también se requiere la descarga de unos binarios pre-compilados de unas librerías llamadas *ICU* y *Boost* en una versión específica en función del compilador utilizado, en este caso *MSVC 2019*, necesarios para la compilación de la herramienta. Aquí es donde surge el problema: en los links que se facilitan en el manual de instalación para la descarga de estos binarios, solo aparecen disponibles hasta la versión del compilador de *MSVC 2017*. Se intentó usar la última versión de los binarios disponibles con el compilador de *MSVC 2019*, pero la compilación falló, como era esperado. También se intentó instalar *MSVC 2017*, pero en la página de Microsoft Visual Studio sólo se permite la descarga completa de la última versión, la de 2019. Como última prueba se intentó buscar en páginas externas los compilados de estas versiones de las librerías para *MSVC 2019* sin éxito.

Debido al elevado tiempo necesario en investigar soluciones a estos problemas, se decidió intentarlo en un entorno Linux, ya que todas las herramientas restantes que se han utilizado para este proyecto, pueden ser instaladas y aplicadas sin problemas en ambos entornos.

C.2. Instalación en Linux

Al igual que para el entorno Windows, el manual de instalación de *Freeling* también detalla los pasos a seguir para la construcción de una API de *Freeling* para *Python* en un entorno Linux, donde se consiguió completar la instalación satisfactoriamente, más fácilmente y sin problemas de versiones como en Windows.

El primer paso, al igual que en la Sección anterior, consiste en descargar el código fuente de la aplicación. Posteriormente, es necesario instalar *g++* y sus dependencias (en una versión con compilador compatible con C++ 11), *CMake* en su versión 3.8 o posterior, y *SWIG*. Estos tres paquetes pueden ser instalados fácilmente en la mayoría de distribuciones usando los siguientes comandos:

```
apt-get install build-essentials
apg-get install cmake
apt-get install swig
```

Adicionalmente, es necesario instalar las librerías *libicu*, *libboost* y *libz*, siendo en este entorno más sencillo que el Windows, únicamente instalando los siguientes componentes: *libicu*, *libboost-regex*, *libboost-system*, *libboost-thread*, *libboost-program-options* y *zlib*. Bajo los siguientes comandos, estas librerías estarían instaladas correctamente (comandos específicos para la distribución Ubuntu/Debian, para distribuciones diferentes pueden variar alguno de los comandos utilizados):

```
apt-get install libboost-dev
apt-get install libboost-regex-dev
apt-get install libboost-system-dev
apt-get install libboost-filesystem-dev
apt-get install libboost-program-options-dev
apt-get install libboost-thread-dev
apt-get install libicu-dev
apt-get install zlib1g
```

Una vez instalados todos los componentes, para la compilación, es necesario abrir un terminal, cambiar al directorio donde se encuentra el código fuente descargado y ejecutar los siguientes comandos:

`mkdir build`
`cd build`
`cmake ..`
`make install` (esto instalará *Freeling* en /usr/local)

Si se desea instalar en otro directorio, añadir `-DCMAKE_INSTALL_PREFIX= $FLINSTALL` al comando `cmake`. En este caso, será necesario configurar una variable de entorno llamada *FREELINGDIR* con la ubicación de la instalación para que la aplicación funcione correctamente.

En esta carpeta *build*, se creará, en el subdirectorio *APIs/python3*, un archivo llamado *pyfreeling.py* con el que ya se podrá hacer uso de *Freeling* en *Python*.

Será necesario conocer el valor del directorio absoluto de este archivo, ya que se utiliza como parámetro de entrada en la aplicación desarrollada para que la herramienta *Freeling* funcione correctamente.

Anexo D

Código representativo de pruebas realizadas

En el Anexo D.1 se muestra el código utilizado para realizar las pruebas de evaluación del detector de entidades médicas. Por otra parte, en el Anexo D.2 se muestra el código utilizado para evaluar el detector de recomendaciones.

D.1. Código de pruebas del detector de términos médicos

A continuación, se muestra el código realizado para evaluar la calidad del detector de entidades médicas. En el código se usa el detector con lematización, la estructura del código para realizar las pruebas de los dos restantes, es similar al mostrado en esta sección.

Las librerías *Database*, y *TextProcessing* importadas son librerías propias creadas para gestionar las tareas de almacenamiento y procesamiento de los datos.

```
import time
from Database import Database
from TextProcessing import TextProcessing
from flashtext import KeywordProcessor
from NER import NER
import json

class LemaNER(NER):

    def __init__(self,source,freeling_path):
        NER.__init__(self,source)
        self.textProcessing = TextProcessing(freeling_path)
        self.database = Database()
        self.processor = KeywordProcessor()
        #Cargar diccionario
        with open("data/"+source+'LemaTermDict.json', 'r') as fp:
            self.dicTerms= json.load(fp)
        #Añadir términos a FlashText
        for d in self.dicTerms:
            self.processor.add_keyword(d)

    def findMatches(self,file_path, res_path, mode =1):
        classKey = 'termType'
        matches = []
        texts = self.database.getTextsFromFile(file_path)
        cont = 0
        start_time_global = time.time()
        for textIn in texts:
            start_time = time.time()
            #Obtener texto eliminando carácteres no deseados y lematizado
            text = self.textProcessing.cleanText(textIn)
            text = self.textProcessing.lematize(text)
            text = self.textProcessing.removeAccentuation(text)
            #Cálculo de matches
            found = self.processor.extract_keywords(text)
            for f in found:
                foundTerms = self.dicTerms[f]
                for ft in foundTerms:
                    matches.append(ft)
            #Ordenar resultados
            matches = sorted(matches, key=lambda k: k['term'])
            uniqueList = []
            resultList = []
            #Eliminar repetidos (Algunos términos aparecen repetidos)
            for match in matches:
```

```
                if self.source  == 'snomed':
                    result = match['termLematized'] + '-->'
                    + match[classKey]+"\n"
                else:
                    result= match['termLematized'] + '-->'
                    + match['conceptName'] + '-->'
                    + match['descriptorName'] + '-->'+match[classKey]+"\n"
                if result not in resultList:
                    resultList.append(result)
                    uniqueList.append(match)
            #Eliminar términos que unicamente tienen número tras el lematizado
            uniqueList = self.applyRules(uniqueList,'termLematized',classKey)
            #Corrector ortográfico
            correctedData = self.textProcessing.getMisspelledWords(text)
            correctText = text
            #No corregir las palabras detectadas(hay palabras tecnicas
            #no recogidas en el diccionario del corrector)
            for element in correctedData:
                for item in element:
                    replace = True
                    for f in found:

                        if element[item] in f:
                            replace = False
                    if replace:
                        correctText=correctText.replace(element[item], item)
            correctedFound = self.processor.extract_keywords(correctText)
            correctedMatches = []
            for f in correctedFound:
                foundTerms = self.dicTerms[f]
                for ft in foundTerms:
                    correctedMatches.append(ft)
            correctUniquelist = []
            correctResultList = []
            for match in correctedMatches:
                if self.source  == 'snomed':
                    result = match['termLematized'] + '-->'
                    + match[classKey] + "\n"
                else:
                    result= match['termLematized'] + '-->'
                    + match['conceptName'] + '-->'+ match['descriptorName']
                    + '-->'+ match[classKey] + "\n"
                if result not in correctResultList:
                    correctResultList.append(result)
                    correctUniquelist.append(match)
```

```
            newMatches = []
            for mc in correctUniquelist:
                add = True
                for m in uniqueList:
                    if m['termLematized'] == mc['termLematized']:
                        add = False
                if add:
                    newMatches.append(mc)
            res_file= open(res_path,'a+','utf8')
            res_file.write("\n")
            res_file.write(text)
            res_file.write("\n")


            #Añadir cada match al fichero
            for u in uniqueList:
                res_file.write(u)

            #Añadir diferencia entre el corrector y no
            res_file.write("Detecciones tras pasar un corrector ortográfico\n")
            for match in newMatches:
                res_file.write("Probabilidad "+match['probability']+"-->"
                               +match['termLematized']
                               + '-->' +match['term']
                               + '-->' +match[classKey]+"\n")
            res_file.write("**********************************************")
            res_file.write("\n")
            print("--- %s seconds processing actual text [%s]---"
                  % (time.time() - start_time,str(cont)))
            cont = cont +1

        res_file.close()
        print("--- %s seconds processing all texts ---"
              % (time.time() - start_time_global))
```

D.2. Código de pruebas del detector de recomendaciones

A continuación, se muestra el código realizado para evaluar la calidad del detector de recomendaciones.

```
import os
import codecs
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import cross_validate
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import *
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm, naive_bayes
import spacy
from flashtext import KeywordProcessor
from Freeling import Freeling
nlp = spacy.load("es")
np.random.seed(500)
freeling = Freeling(freeling_path)
'''
    Preprocesar textos
'''
# Cargar datos de entrenamiendo
Corpus = pd.read_csv(r"test.csv",encoding='utf8')
#Carar y añadir a flashtext verbos
file = open("categorizacion de verbos.txt",'r',encoding='utf8')
processor = KeywordProcessor()
for line in file:
    processor.add_keyword(line.replace('\n',''))
# Eliminar filas vacias
Corpus['text'].dropna(inplace=True)
# Poner todo el texto en minúsculas
Corpus['text'] = [entry.lower() for entry in Corpus['text']]
#Tokenization y lemmatización
for i,textrow in enumerate(Corpus['text']):
    proc_words = []
    #Tokenizar texto
    for word in nlp(textrow):
        if str(word).isalpha():
            #Limpiar y lematizar
```

```
            proc_word = str(freeling.lemmatize_word(str(word)))
            proc_word = re.sub('[^a-zA-ZñáéíóúÁÉÍÓÚ]+', ' ',proc_word)
            proc_word = re.sub(' +',' ',proc_word)
            proc_words.append(proc_word)
    Corpus.loc[i,'text_final'] = str(proc_words)
tp=0
fp=0
fn=0
tn=0
'''
    Categorización de textos
'''
for i, row in Corpus.iterrows():
    label = row['label']
    found = processor.extract_keywords(row['text_final'])
    #True Positives
    if str(label) == '1' and len(found)>0:
        tp +=1
    #False Positives
    if str(label) == '0' and len(found)>0:
        fp +=1
    #False Negatives
    if str(label) == '0' and len(found)==0:
        fn+=1
    #True Negatives
    if str(label) == '1' and len(found)==0:
        tn+=1
print("Categorización de verbos")
precision= tp/(tp+fp)
recall = tp/(tp+fn)
accuracy = (tp+tn)/(tp+fp+tn+fn)
f1 = 2 * (precision * recall) / (precision + recall)
print("Categorización de verbos Precision Score: "+ str(precision))
print("Categorización de verbos Recall Score: "+ str(recall))
print("Categorización de verbos F1 Score: "+ str(f1))
print("Categorización de verbos Accuracy Score: "+ str(accuracy))

'''
    Modelos de aprendizaje automático
'''
#TF-IDF
Tfidf_vect = TfidfVectorizer(max_features=5000)
data = Tfidf_vect.fit_transform(Corpus['text_final'])
target = Corpus['label']
#Añadir modelos a testear
```

```
models = []
models.append(('SVM-linear C=1.0', svm.SVC(C=1.0, kernel='linear'),1))
models.append(('SVM-linear C=0.5', svm.SVC(C=0.5, kernel='linear'),2))
models.append(('SVM-linear C=0.1', svm.SVC(C=0.1, kernel='linear'),3))
models.append(('SVM-poly degree 2', svm.SVC(C=0.5, kernel='poly', degree=2)))
models.append(('NB',naive_bayes.MultinomialNB()))
models.append(('RF 1000',RandomForestClassifier(n_estimators = 1000)))
models.append(('RF 2000',RandomForestClassifier(n_estimators = 2000)))
models.append(('KNN-3 euclidean',KNeighborsClassifier(n_neighbors=3,
                                                       metric='euclidean')))
models.append(('KNN-5 euclidean',KNeighborsClassifier(n_neighbors=5,
                                                       metric='euclidean')))
models.append(('KNN-3 manhattan',KNeighborsClassifier(n_neighbors=3,
                                                       metric='manhattan')))
models.append(('KNN-5 manhattan',KNeighborsClassifier(n_neighbors=5,
                                                       metric='manhattan')))

scoring = {'accuracy' : make_scorer(accuracy_score),
           'precision' : make_scorer(precision_score),
           'recall' : make_scorer(recall_score),
           'f1_score' : make_scorer(f1_score)}
for n, m in models:
    #Calcular métricas de evaluacion con kfold-cross validation con k =5
    scores = cross_validate(m,data,target,cv=5,scoring=scoring)
    print(n+" Precision Score -> ",np.mean(scores['test_precision']))
    print(n+" Recall Score -> ",np.mean(scores['test_recall']))
    print(n+" F1 Score -> ",np.mean(scores['test_f1_score']))
    print(n+" Accuracy Score -> ",np.mean(scores['test_accuracy']))
    print("\n")
```

Anexo E

Ejemplo de resultados del detector

En el siguiente Anexo se muestran dos textos de ejemplo etiquetados utilizando la herramienta desarrollada en el presente trabajo, en el Anexo E.1 con el diccionario creado con datos de SNOMED-CT y en el Anexo E.2 con el diccionario de términos de DeCS.

E.1. Resultados con datos de SNOMED-CT - Ejemplo de texto

Leyenda

Detectado correctamente: VERDE
Detectado incorrectamente: ROJO
No detectado: AZUL

“ LUMBOCIATICA Descripción de la(s) exploración(es): - EXPLORACIÓN: RM de columna lumbosacra, secuencias en ponderación T1 sagital, secuencia DIXON sagital y T1 y T2 plano axial. Hallazgos: Pérdida de la lordosis

lumbar con rectificación. Abombamientos de platillos generalizados, pero con correcta altura de cuerpos vertebrales. Alineación anteroposterior conservada. Moderados signos espondilósicos con incipiente osteofitosis de predominio anterior. Salidas difusas circunferenciales discales. Disminución generalizada de intensidad de señal a nivel discal en T2 indicativo de deshidratación, mucho más evidente en los últimos niveles lumbares. Esclerosis interapofisaria asociada. - NIVEL L2-L3: bandas parcheadas de hiperseñal en T1 y T2 en platillos indicativos de cambios degenerativos tipo II. Salida difusa circunferencial discal. Leve deshidratación discal. Ligera esclerosis interapofisaria. - NIVEL L3-L4: salida difusa circunferencial discal. Leve deshidratación discal.Ligera esclerosis interapofisaria. - NIVEL L4-L5: salida difusa circunferencial discal. Pequeña hernia posteromedial del núcleo pulposo. Marcada deshidratación discal. Disminución del espacio intersomático. Marcada esclerosis interapofisaria, con hipertrofia. Se asocia con hipertrofia ligamentaria que disminuyen el calibre transverso del canal. En su conjunto se reconoce ligero compromiso de recesos laterales y de ambos forámenes secundario. - NIVEL L5-S1: Grandes bandas de hiperseñal en T2 y T2 en platillos, fundamentalmente en el inferior de L5, que indican cambios degenerativos tipo II. Importante disminución del espacio intersomático. Hiperintensidad de señal discal en T1 y T2, que se suprime con la secuencia saturación grasa, que indica recambio degenerativo graso discal. Marcada hipertrofia interapofisaria. Osteofitosis y protrusión disco osteofitaria posteromedial. Ligera disminución del calibre del canal. Diagnóstico: Nombre Responsable 1: [1] - Fecha de Firma: ZARAGOZA, N.Colegiado: Categoría Profesional 1: - Informe de Resultados de Pruebas de Imagen. Servicio de Radiodiagnóstico. Fecha de Impresión: - Pérdida de la lordosis lumbar con rectificación. - Signos espondilósicos con salidas difusas circunferenciales discales. Deshidratación es discales asociadas y esclerosis interapofisaria. - L2-L3: cambios degenerativos y II en platillos. - L4-5: disminución del espacio intersomático, esclerosis e hipertrofia interapofisaria ligamentaria con disminución de calibre transverso del canal. Compromiso de recesos laterales. Pequeña hernia posteromedial y de núcleo pulposo. - L5-S1: cambios degenerativos tipo II en platillos. Recambio graso discal. Marcada hipertrofia interapofisaria."

Resultados
Estructura: "Término detectado"→"Clase/s asociada/s"

cambio degenerativo →anomalía morfológica

canal →estructura corporal
cuerpo vertebral →estructura corporal
deshidratación →trastorno
disco →fármaco de uso clínico
disminución →anomalía morfológica
esclerosis →anomalía morfológica
exploración →procedimiento
grasa →sustancia, estructura corporal
hallazgo →hallazgo
hernia →anomalía morfológica
hipertrofia →anomalía morfológica
lordosis →trastorno
núcleo pulposo, L5 - S1 →estructura corporal
protrusión →anomalía morfológica
prueba Con A →procedimiento
se reconoce →hallazgo
signo →hallazgo
Probabilidad media → espondilosis (trastorno) →trastorno

Nótese que *prueba Con A* es incorrectamente detectado porque al procesar el término, y por tanto eliminar palabras vacías, éste se transforma en únicamente *prueba*.

El término *espondilosis* tiene una probabilidad media debido a que es un término detectado tras pasar un detector ortográfico, sin el cual no se detectaría este elemento en concreto.

E.2. Resultados con datos de DeCS - Ejemplo de texto

Leyenda

Detectado correctamente: VERDE
Detectado incorrectamente: ROJO
No detectado: AZUL

"SINDROME CORONARIO AGUDO Paciente intervenido triple Bypass coronario AMI a DA y vena safena a Dx yDp .- se copiainforme, (se encuentra en OMI) le indican que han solicitado consulta en Cardiologia , pero no consta en su historico. Motivo del Alta: Curación o mejoría. Motivo inmediato del ingreso: Paciente de años de edad que ingresa procedente de Hospital Miguel Servet para cirugía coronaria urgente. Anamnesis: Antecedentes personales: Dudosa alergia a Amoxcilina-Clavulanico. Exfumador. No HTA. DM tipo 2 (ADO). Dislipemia. Poliquistosis renal y ectasia pielocalicial derecha. Esteatosis hepática. Bocio. Diagnosticado de SAOS. Reflujo Gastroesofágico. Hipoacusia. Intervenido de septoplastia. Colelitiasis. Colecistectomía. Historia Cardiológica: Estudiado por dolor torácico atípico en Medicina Interna, y Cardiología, con ergometría no sugerente de isquemia con 10 METS de carga en . El acude a Urgencias por clínica de ángor de reposo de algunas horas de duración, sin componente postural, y desencadenados por esfuerzo hace unas semanas. A su llegada a Urgencias, nuevo dolor, realizando ECG que evidencia pseudopositivización de onda T, que desaparece tras comenzar pc de SLN + mínima elevación de TnUS (troponina pico 180), decidiendo ingreso en UCI. El se realiza coronariografía que evidencia enfermedad multivaso, es presentado en sesión médico-quirúrgica decidiéndose cirugía en el ingreso. Exploraciones Complementarias: Ecocardiograma: Cavidades cardiacas y Aorta ascendente de dimensiones normales. HVI ligera. Contractilidad global conservada, sin apreciar alteraciones segmentarias. Patrón de relajación disminuida, sin elevación de las PTDVI. Válvulas estructural y funcionalmente normales (VAo trivalva) Contractilidad normal del VD. Cava y suprahepáticas no dilatadas, sin inversión de flujos y normocolapso inspiratorio. No signos indirectos de HTP. No afectación pericárdica Cateterismo: Tronco: Sin lesiones. DA en segmento proximal presenta estenosis

crítica y luego estenosis significativa respectivamente. 1ra diagonal: 1 mm con lesión significativa ostial. 2da diagonal. lesión significativa ostial. Lesión significativa en tercio distal de CX que involucra ostium de rama marginal. Arteria Intermedia: Estenosis en límite de la significancia en segmento proximal. CD: Estenosis ligera tercio proximal. DP con estenosis ostial en límite de la significancia y lesión en tercio medio significativa. Procedimientos Terapéuticos: Fecha de la intervención: . Cirujano. Se realiza bajo CEC triple bypass coronario: AMI a DA y vena safena a Dx y DP. En quirófano inestabilidad hemodinámica con bradicardia extrema que precisa de entrada urgente en C.

Resultados

Estructura: "Término detectado" → "Clase/s asociada/s"

alergia →ENFERMEDADES - SALUD PÚBLICA

anamnesis →TÉCNICAS Y EQUIPOS ANALÍTICOS, DIAGNÓSTICOS Y TERAPÉUTICOS

bocio →ENFERMEDADES - SALUD PÚBLICA

bradicardia →ENFERMEDADES

cec →TÉCNICAS Y EQUIPOS ANALÍTICOS, DIAGNÓSTICOS Y TERAPÉUTICOS

cardiologia →→DISCIPLINAS Y OCUPACIONES

cateterismo →TÉCNICAS Y EQUIPOS ANALÍTICOS, DIAGNÓSTICOS Y TERAPÉUTICOS

cirugia →DISCIPLINAS Y OCUPACIONES

cirujano →DENOMINACIONES DE GRUPOS - ATENCIÓN DE SALUD

colecistectomia →TÉCNICAS Y EQUIPOS ANALÍTICOS, DIAGNÓSTICOS Y TERAPÉUTICOS

colelitiasis →ENFERMEDADES

consulta →ATENCIÓN DE SALUD

dislipemia →ENFERMEDADES

dolor →ENFERMEDADES - PSIQUIATRÍA Y PSICOLOGÍA - FENÓMENOS Y PROCESOS

dolor toracico →ENFERMEDADES

ecg →TÉCNICAS Y EQUIPOS ANALÍTICOS, DIAGNÓSTICOS Y TERAPÉUTICOS

ectasia →ENFERMEDADES

elevacion →FENÓMENOS Y PROCESOS
enfermedad →ENFERMEDADES - SALUD PUBLICA
ergometria →TÉCNICAS Y EQUIPOS ANALÍTICOS, DIAGNÓSTICOS Y TERAPÉUTICOS
estenosis →ENFERMEDADES
hemodinamica →FENÓMENOS Y PROCESOS
hepatico →ORGANISMOS
hipoacusia →ENFERMEDADES
historia →HUMANIDADES
hospital →ATENCIÓN DE SALUD - VIGILANCIA SANITARIA - SALUD PÚBLICA
ingreso →ATENCIÓN DE SALUD - SALUD PÚBLICA
inversion →ATENCIÓN DE SALUD - SALUD PÚBLICA
isquemia →ENFERMEDADES
lesion →ENFERMEDADES - SALUD PÚBLICA
medicina interno →DISCIPLINAS Y OCUPACIONES
medico →DENOMINACIONES DE GRUPOS - SALUD PÚBLICA - ATENCIÓN DE SALUD
paciente →DENOMINACIONES DE GRUPOS
pico →ANATOMÍA
procedimiento terapeutico →TÉCNICAS Y EQUIPOS ANALÍTICOS, DIAGNÓSTICOS Y TERAPÉUTICOS - VIGILANCIA SANITARIA
quirofano →ATENCIÓN DE SALUD - VIGILANCIA SANITARIA
reflujo gastroesofagico →ENFERMEDADES
relajacion →ANTROPOLOGÍA, EDUCACIÓN, SOCIOLOGÍA Y FENÓMENOS SOCIALES
reposo →ANTROPOLOGÍA, EDUCACIÓN, SOCIOLOGÍA Y FENÓMENOS SOCIALES
signo →ENFERMEDADES
sindrome coronario agudo →ENFERMEDADES
troponina →COMPUESTOS QUÍMICOS Y DROGAS
uci →ATENCIÓN DE SALUD - VIGILANCIA SANITARIA
urgencia →ENFERMEDADES - ATENCIÓN DE SALUD

Bibliografía

[1] Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E. D., Gutierrez, J. B., and Kochut, K. (2017). A brief survey of text mining: Classification, clustering and extraction techniques. *arXiv preprint arXiv:1707.02919*.

[2] Allibhai, E. (2018). Building a k-Nearest-Neighbors (k-NN) Model with Scikit-learn. https://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a. Accedido 15-09-2019.

[3] Altman, R. (2017). Artificial intelligence (AI) systems for interpreting complex medical datasets. *Clinical Pharmacology & Therapeutics*, 101(5):585–586.

[4] Anaconda Software Foundation (2016). Computer software. Vers. 2-2.4.0. Anaconda, Nov. 2016. Web. https://anaconda.com.

[5] Arias, F. J. C. (2019). Fuzzy String Matching in Python. https://www.datacamp.com/community/tutorials/fuzzy-string-python. Accedido 15-08-2019.

[6] Babar, N. (2018). The Levenshtein Distance Algorithm. https://dzone.com/articles/the-levenshtein-algorithm-1. Accedido 15-08-2019.

[7] Ben-Fraj, M. (2018). In Depth: Parameter tuning for SVC. http://medium.com/. Accedido 20-08-2019.

[8] Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc.

[9] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

[10] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122.

[11] Caruana, R. and Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *23rd international conference on Machine learning*, pages 161–168. ACM.

[12] Castano, J., Gambarte, M. L., Park, H. J., Williams, M. d. P. A., Perez, D., Campos, F., Luna, D., Benitez, S., Berinsky, H., and Zanetti, S. (2016). A machine learning approach to clinical terms normalization. In *15th Workshop on Biomedical Natural Language Processing*, pages 1–11.

[13] Chen, T., Dredze, M., Weiner, J. P., Hernandez, L., Kimura, J., and Kharrazi, H. (2019). Extraction of Geriatric Syndromes From Electronic Health Record Clinical Notes: Assessment of Statistical Natural Language Processing Methods. *JMIR Medical Informatics*, 7(1):e13039.

[14] Chomboon, K., Chujai, P., Teerarassamee, P., Kerdprasop, K., and Kerdprasop, N. (2015). An empirical study of distance metrics for k-nearest neighbor algorithm. In *3rd International Conference on Industrial Application Engineering*, pages 1–6.

[15] Chowdhury, G. G. (2003). Natural language processing. *Annual review of information science and technology*, 37(1):51–89.

[16] Christopher, D. M., Prabhakar, R., and Hinrich, S. (2008). Introduction to information retrieval. *An Introduction To Information Retrieval*, 151(177):5.

[17] Coppersmith, G., Leary, R., Crutchley, P., and Fine, A. (2018). Natural language processing of social media as screening for suicide risk. *Biomedical Informatics Insights*, 10.

[18] Costumero, R., Lopez, F., Gonzalo-Martín, C., Millan, M., and Menasalvas, E. (2014). An approach to detect negation on medical documents

in Spanish. In *International Conference on Brain Informatics and Health*, pages 366–375. Springer.

[19] DeCS (2018). Servicios DeCS. http://wiki.reddes.bvsalud.org. Accedido 01-08-2019.

[20] Descriptores de Ciencias de la Salud (2017). DeCS [Internet]. Sao Paulo (SP): BIREME / OPS / OMS. http://decs.bvsalud.org/E/homepagee.htm. Accedido 25-04-2019.

[21] Ferrández Escámez, Ó., Kozareva, Z. P., Montoyo, A., and Muñoz, R. (2005). NERUA: sistema de detección y clasificación de entidades utilizando aprendizaje automático. *Procesamiento del lenguaje natural, nº 35 (sept. 2005); pp. 37-44.*

[22] FlashText 1.0 (2017). Flashtext documentation. https://flashtext.readthedocs.io/. Accedido 20-05-2019.

[23] Freeling (2016). Freeling User Manual. https://freeling-user-manual.readthedocs.io/en/v4.1/. Accedido 25-09-2019.

[24] Gutiérrez Fernández, R. (2017). La humanización de (en) la atención primaria. *Revista Clínica de Medicina de Familia*, 10(1):29–38.

[25] Harris, Z. (1988). *Language and information.* Columbia University Press.

[26] Hermitdave (2018). Frequency Words. https://github.com/hermitdave/FrequencyWords. Accedido 01-09-2019.

[27] Honnibal, M. and Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

[28] IBM (2012). Conceptos básicos de ayuda de CRISP-DM. https://www.ibm.com/support/knowledgecenter/. Accedido 15-04-2019.

[29] Instituto Aragonés de Ciencias de la Salud (2019). Biblioteca de Guías de Práctica Clínica del Sistema Nacional de Salud. https://portal.guiasalud.es/. Accedido 01-04-2019.

[30] Instituto Aragonés de Ciencias de la Salud. (2019). BiGS GuiaSalud Intelligent Library. http://www.iacs.es/wp-content/uploads/2017/07/BiGS_Background.pdf.

[31] JavaScript Programming Language Standard (2002). Introducing a JSON. Web. http://www.json.org.

[32] Joshi, V. (2017). Trying to understand tries. https://medium.com/basecs/trying-to-understand-tries-3ec6bede0014. Accedido 01-09-2019.

[33] Lan, M., Tan, C. L., Su, J., and Low, H. B. (2007). Text representations for text categorization: a case study in biomedical domain. In *2007 International Joint Conference on Neural Networks*, pages 2557–2562. IEEE.

[34] Leser, U. and Hakenberg, J. (2005). What makes a gene name? Named entity recognition in the biomedical literature. *Briefings in Bioinformatics*, 6(4):357–369.

[35] Liu, B., Ma, Y., and Wong, C. K. (2000). Improving an association rule based classifier. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 504–509. Springer.

[36] Magerman, D. M. (1995). Statistical decision-tree models for parsing. In *33rd annual meeting on Association for Computational Linguistics*, pages 276–283. Association for Computational Linguistics.

[37] Mammothb (2019). Symspellpy. https://github.com/mammothb/symspellpy. Accedido 15-09-2019.

[38] Marimon, M., Gonzalez-Agirre, A., Intxaurrondo, A., Rodrguez, H., Lopez Martin, J., Villegas, M., and Krallinger, M. (2019). Automatic de-identification of medical texts in Spanish: the meddocan track, corpus, guidelines, methods and evaluation of results. In *Iberian Languages Evaluation Forum (IberLEF 2019). CEUR Workshop Proceedings (CEUR-WS.org), Bilbao, Spain (Sep 2019).*

[39] Marrero, M., Sánchez-Cuadrado, S., Urbano, J., Morato, J., and Moreiro, J.-A. (2009). Sistemas de recuperación de información adaptados al dominio biomédico. *El profesional de la información*, 19(3):246–254.

[40] McKinney, Wes and others (2010). Data structures for statistical computing in Python. In *9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX.

[41] Mongo DB, Inc (2019). Vers 4.2.0. Web. https://www.mongodb.com/.

[42] Nadeau, D. and Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26.

[43] Névéol, A., Dalianis, H., Velupillai, S., Savova, G., and Zweigenbaum, P. (2018a). Clinical natural language processing in languages other than English: opportunities and challenges. *Journal of biomedical semantics*, 9(1):12.

[44] Névéol, A., Zweigenbaum, P., et al. (2016). Clinical natural language processing in 2015: leveraging the variety of texts of clinical interest. *Yearbook of Medical Informatics*, 25(01):234–239.

[45] Névéol, A., Zweigenbaum, P., et al. (2018b). Expanding the Diversity of Texts and Applications: Findings from the Section on Clinical Natural Language Processing of the International Medical Informatics Association Yearbook. *Yearbook of Medical Informatics*, 27(01):193–198.

[46] Oliphant, T. (2006). *Guide to NumPy*.

[47] Osuna, E., Freund, R., and Girosit, F. (1997). Training Support Vector Machines: an application to face detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 130–136. IEEE.

[48] Padró, L. (2011). Analizadores Multilingües en FreeLing. *Linguamatica*, 3(2):13–20.

[49] Pons, E., Braun, L. M., Hunink, M. M., and Kors, J. A. (2016). Natural language processing in radiology: a systematic review. *Radiology*, 279(2):329–343.

[50] Porumb, M., Barbantan, I., Lemnaru, C., and Potolea, R. (2015). RE-Med: automatic relation extraction from medical documents. In *17th International Conference on Information Integration and Web-based Applications & Services*, page 19. ACM.

[51] Python Software Foundation (2019). Python Language Reference, version 3.7. Web. http://www.python.org.

[52] Python Standar Library (2019). re - Regular expression operations. https://docs.python.org/3/library/re.html. Accedido 10-05-2019.

[53] Quimbaya, A. P., Múnera, A. S., Rivera, R. A. G., Rodríguez, J. C. D., Velandia, O. M. M., Peña, A. A. G., and Labbé, C. (2016). Named entity recognition over electronic health records through a combined dictionary-based approach. *Procedia Computer Science*, 100:55–61.

[54] Ramos, J. et al. (2003). Using TF-IDF to determine word relevance in document queries. In *First Instructional Conference on Machine Learning*, volume 242, pages 133–142.

[55] Román, J. V. (2018). CRISP-DM: La metodología para poner orden en los proyectos. https://www.sngular.com/es/data-science-crisp-dm-metodologia. Accedido 15-04-2019.

[56] Sarawagi, S. et al. (2008). Information extraction. *Foundations and Trends® in Databases*, 1(3):261–377.

[57] Schneider, K.-M. (2005). Techniques for improving the performance of Naive Bayes for text classification. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 682–693. Springer.

[58] Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47.

[59] Shaikh, R. (2018). Cross validation explained: Evaluating estimator performance. https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85. Accedido 15-09-2019.

[60] SNOMED International (2019). SNOMED-CT. http://www.snomed.org/snomed-ct/why-snomed-ct. Accedido 20-04-2019.

[61] Wu, C., Xia, F., Deleger, L., and Solti, I. (2011). Statistical machine translation for biomedical text: are we there yet? In *AMIA Annual Symposium Proceedings*, volume 2011, page 1290. American Medical Informatics Association.

[62] Yang, J., Parikh, D., and Batra, D. (2016). Joint unsupervised learning of deep representations and image clusters. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5147–5156.

[63] Yim, W., Yetisgen, M., Harris, W. P., and Kwan, S. W. (2016). Natural language processing in oncology: a review. *JAMA oncology*, 2(6):797–804.

[64] Zhang, Q., Yang, L. T., Chen, Z., and Li, P. (2018). A survey on deep learning for big data. *Information Fusion*, 42:146–157.

[65] Zhou, G., Zhang, J., Su, J., Shen, D., and Tan, C. (2004). Recognizing names in biomedical texts: a machine learning approach. *Bioinformatics*, 20(7):1178–1190.