Escuela
Politécnica
Superior

Desarrollo de un modelo de Procesamiento del Lenguaje Natural para la extracción de información en documentos del dominio de la salud

Máster Universitario en Ciencia de Datos

Trabajo Fin de Máster

Autor:
Eduardo Grande Ruiz
Tutor:
Yoan Gutiérrez Vázquez

Universitat d'Alacant
Universidad de Alicante

Junio 2023

Desarrollo de un modelo de Procesamiento del Lenguaje Natural para la extracción de información en documentos del dominio de la salud

Autor
Eduardo Grande Ruiz

Tutor
Yoan Gutiérrez Vázquez
Departamento de Lenguajes y Sistemas Informáticos

Máster Universitario en Ciencia de Datos

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Junio 2023

Resumen

En la actualidad existen múltiples modelos de inteligencia artificial centrados en la detección de entidades nombradas, que son capaces de detectar una amplia variedad de aspectos. En este trabajo, se centran esos aspectos a enfermedades raras, detectándolas en textos del ámbito clínico.

Todos esos textos clínicos son resúmenes de documentos científicos publicados en *PubMed*.

De las enfermedades, no solo se detectarán sus nombres en sí, sino que se quieren detectar una amplia variedad de aspectos relacionados con esas enfermedades, como por ejemplo, sus causas, tratamientos, diagnósticos... Todos esos aspectos se clasificarán en una serie de categorías.

Las anotaciones del modelo se generarán, en primera instancia, de forma automática, usando la herramienta *Metathesaurus*, contenida dentro de *UMLS*, un sistema de lenguaje médico. *Metathesaurus* contiene más de 3 millones de conceptos, siendo la inmensa mayoría del ámbito clínico. Además, cuenta con una serie de categorías ya definidas, y con los conceptos clasificados en estas categorías.

Para cada texto, se cuenta con un archivo *txt* que contiene el texto y un archivo *ann* que contiene sus anotaciones. Esas anotaciones se encuentran definidas en formato BRAT, un formato de anotación que permite después visualizarlas de forma fácil, modificarlas y crear nuevas. Para cada anotación, se especifica el inicio, final, la categoría a la que pertenece y las palabras o grupos de palabras sobre las que se aplica.

Una vez se cuenta con esas anotaciones, es posible revisarlas manualmente para que el corpus sea de la mayor calidad posible, pero al tener una base ya de anotaciones, esta tarea será más ágil.

La clasificación que se debe de realizar es compleja, ya que contiene bastantes categorías, además de que cada palabra (o grupos de palabras) pueden pertenecer a la vez a varias clases, por lo que las anotaciones se pueden superponer tanto de forma estricta (mismo inicio y final) como de forma parcial.

Para la obtención del modelo, se contará como base *PubMedBERT*, un modelo basado en *BERT* reentrenado por Microsoft con vocabulario del ámbito clínico, también extraído de *PubMed*. Este modelo será ajustado para poder ser usado en esta tarea en concreto.

Como es una tarea particular, se han definido una serie de métricas, diferenciando las tareas de detección y de clasificación. Esas métricas serán de utilidad para conocer el rendimiento del modelo, y poder ver así si es lo suficientemente bueno, o por contra, se deben de realizar mejoras para obtener mejor rendimiento.

En conclusión, este trabajo busca desarrollar un modelo para la detección de enfermedades raras en textos clínicos, usando un corpus extraído de documentos científicos clínicos. Las anotaciones podrán solaparse, por lo que al tratarse de una tarea particular de detección de entidades, se realizan modificaciones sobre el modelo para reentrenarlo y métricas para medir el modelo resultante.

Agradecimientos

Este Trabajo de Fin de Máster no habría sido posible sin la ayuda, guía y consejos de mi tutor, Yoan Gutiérrez, así como de los miembros del Grupo de Procesamiento del Lenguaje Natural y Sistemas de Información. Gracias a todos por vuestro tiempo.

Se agradece el apoyo recibido por parte de la fundación ValgrAI (Valencian Graduate School and Research Network for Artificial Intelligence) y de la Generalitat Valenciana para la realización del Máster en Ciencia de Datos. Además, se agradece al Vicerrectorado de Investigación de la Universidad de Alicante por ser beneficiario de la ayuda al fomento de la I+D+i, de la que parte de las investigaciones realizadas gracias a esta ayuda se nutre el presente trabajo.

Sólo cabe progresar cuando se piensa en grande, sólo es posible avanzar cuando se mira lejos.

José Ortega y Gasset

Índice general

Índice de figuras

Índice de tablas

1. Introducción

Podemos considerar 2023 como un año importante para el campo del procesamiento del lenguaje natural. La irrupción de GPT-4, implementado en la herramienta ChatGPT, ha causado un gran revuelo en la sociedad.

De los muchos campos en los que el procesamiento del lenguaje natural puede ayudar, este Trabajo de Fin de Máster se centrará en el ámbito médico, lugar en el que los nuevos modelos de lengua y la inteligencia artificial pueden ser de gran ayuda.

A lo largo de este primer capítulo se explicará la motivación y la metodología empleada a la hora de hacer este trabajo.

1.1. Motivación

Una enfermedad rara es aquella que afecta a menos de 5 personas por cada 10.000. Muchas de estas enfermedades son de origen genético o de anomalías congénitas, por ello, muchos casos aparecen a temprana edad. Pese a esta aparición temprana, la prevalencia de las enfermedades raras en adultos es superior a en los niños a causa de la gran mortalidad que conllevan estas enfermedades.

Según un artículo de Posada y cols. (2008), los medicamentos que hay en el mercado para curar las enfermedades raras son escasos, y los avances en investigación para desarrollar esos medicamentos son muy costosos y lentos.

En repositorios de publicaciones científicas del ámbito clínico, como *PubMed*, se pueden encontrar una gran cantidad de artículos que hablan de estas enfermedades raras, sus causas, avances en tratamientos, curas, medicamentos...

Todos estos artículos se encuentran en texto plano sin procesar. Por tanto, el procesamiento automático de grandes volúmenes de estos textos científicos es complejo.

En este Trabajo de Fin de Máster se pretende crear un modelo de inteligencia artificial que sea capaz de detectar aspectos relevantes en artículos científicos que traten de enfermedades raras.

Gracias a este modelo, se deberían de poder detectar de manera automática enfermedades, medicamentos, terapias, causas, consecuencias... Así, se podrá de forma fácil buscar relaciones entre las enfermedades raras y demás aspectos.

Este TFM se enfoca en ayudar en las tareas del proyecto de investigación *T2KNOW: Plataforma de análisis avanzado de textos científico-técnicos para la extracción de tendencias y conocimiento mediante técnicas de PLN*, desarrollado por el Grupo de Procesamiento del lenguaje y sistemas de información de la Universidad de Alicante.

1.2. Metodología empleada

Para la realización de este Trabajo de Fin de Máster se ha contado con un plazo aproximado de 5 meses, a lo largo de los cuales se ha ido desarrollando todo el trabajo, así como la presente memoria y presentación de cara a la defensa oral.

Al comenzar el proyecto, se definieron una serie de objetivos que se deberían de cumplir durante el transcurso. Todos ellos se detallan en el Capítulo 3.

Durante el desarrollo del proyecto, se ha estado en permanente contacto con el tutor académico del trabajo, así como con diferentes miembros de su grupo de investigación, tanto de forma presencial como telemática. También, se ha estado en contacto con la persona responsable de la tutoría académica de la Ayuda para estudios de máster oficial e investigación, otorgada por la Universidad de Alicante, dentro del programa propio del Vicerrectorado de Investigación para el fomento de la I+D+I.

A lo largo de estos meses, se han ido resolviendo problemas, clarificando dudas técnicas y teóricas que han surgido, así como redefinir y comprobar el estado de los objetivos conforme avanzaba el desarrollo.

2. Marco Teórico

Como marco teórico de este trabajo, se destacan aspectos relacionados con las tecnologías empleadas a la hora de desarrollar el modelo para la detección de enfermedades raras. Es por ello por lo que a continuación se explicarán conceptos como *modelo BERT*, qué es el procesamiento del lenguaje natural, o de forma general, qué es la inteligencia artificial.

2.1. Inteligencia artificial

¿Qué es la inteligencia artificial? Múltiples definiciones podemos encontrar a lo largo de la historia.

Según la Real Academia Española (Asale (s.f.)), la inteligencia artificial es una "Disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico." De hecho, en el 2022, este término fue el elegido como *palabra del año 2022* (Morales y cols. (2022)).

La Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura (UNESCO) publicó en 2021 un documento con recomendaciones sobre la ética de la inteligencia artificial (UNESCO (2021)). En él, se dice que una definición de la IA "tendría que cambiar con el tiempo en función de los avances tecnológicos".

2.2. Procesamiento del lenguaje natural

El procesamiento del lenguaje natural (PLN, o NLP por sus siglas en inglés de *Natural Language Processing*), es una disciplina encargada de estudiar las interacciones entre el lenguaje humano y los ordenadores.

Dentro de este campo, se encuentran muchas técnicas, algunas de ellas se muestran a continuación:

- Tokenización: Proceso mediante el cual un texto se divide en unidades más pequeñas llamadas *tokens*.
- Etiquetado de partes de habla (*POS*, de las siglas en inglés de *Part of Speech*): Proceso en el cual se asignan etiquetas a cada uno de los tokens de un texto. Estas etiquetas indican categorías gramaticales.
- Análisis sintáctico: Se trata de analizar la estructura gramatical de una frase para comprender las relaciones entre las palabras.
- Reconocimiento de entidades nombradas (*NER*, por las siglas en inglés de *Name Entity Recognition*): Consiste en identificar y clasificar elementos de un texto, como puedan ser personas, lugares, objetos... Es una tarea muy útil para extraer información específica de un texto.

- Desambiguación: Tarea para conocer el significado concreto de una palabra en un contexto determinado. Se pueden usar técnicas como la desambiguación léxica basada en el contexto o técnicas más modernas como el uso de lenguajes preentrenados.

- Análisis de sentimientos: Tarea enfocada a determinar la actitud, emoción o sentimientos expresados en un texto.

- Traducción automática: Proceso de traducir, de forma automática, un texto de un lenguaje a otro.

Vistas las técnicas, se detallan ahora algunas de las dificultades a las que se enfrenta este campo de estudio, algunas de ellas muy relacionadas con ciertas técnicas mostradas.

- Ambigüedad: Una misma palabra puede tener varios significados, por lo que seleccionar el significado correcto en un cierto contexto es una de las tareas del PLN. Concretamente, a esta tarea se le denomina en inglés *word sense desambiguation* (*WSD*).

- Referencias: Muchas veces al hablar hacemos referencias a personas u objetos sin referirnos directamente a ellos. Por ejemplo, la frase "Él es el autor de este trabajo", no conocemos a quién se refiere con "Él". Por tanto, la resolución de referencias o anáforas es un campo complejo dentro del PLN.

- Sentido común y conocimiento del mundo: En ciertas ocasiones, al procesar texto se requiere tener conocimiento general del mundo para poder entenderlo. Además, el sentido común (el menos común de los sentidos según el filósofo Voltaire), también es necesario para la comprensión de los textos.

- Reconocimiento de entidades nombradas: La complejidad de esta tarea puede llegar a ser importante, al necesitar contar con un amplio conocimiento para realizar una buena detección.

- Análisis sintáctico y gramatical: Estos dos tipos de análisis son un gran desafío en este campo, al tener que analizar dependencias, etiquetar la gramática, realizar la desambiguación sintáctica de los textos y otras costosas tareas.

- Ironía, emociones, metáforas, sarcasmo...: La detección de todos estos aspectos es muy compleja. La identificación de uno de estos elementos puede suponer un gran cambio a la hora de comprender una frase de un texto. Por tanto, es muy importante detectarlos.

El PLN es un campo que en los últimos años ha tenido grandes avances. Los modelos de lenguaje basados en transformers han supuesto una gran revolución. Algunos de ellos, como los modelos GPT (siglas del inglés *Generative Pre-trained Transformers*) o los modelos BERT (siglas del inglés *Bidirectional Encoder Representations from Transformers*) han obtenido un espectacular rendimiento, dando lugar a aplicaciones comerciales muy conocidas como *ChatGPT*, desarrollado por la empresa OpenAI.

Además, algunos de estos modelos se han realizado para ser capaces de procesar textos en múltiples idiomas (modelos multilingües).

Hoy en día, el reentrenamiento y ajuste de los modelos (*fine tunning* en inglés), está a la orden del día. Estas técnicas han demostrado ser muy efectivas, además de ahorrar tiempo al partir de un modelo base.

Por último, en este apartado cabe destacar a la Sociedad Española para el Procesamiento del Lenguaje Natural (SEPLN), asociación sin ánimo de lucro cuyas actividades están enfocadas en el campo del PLN. A ella pertenecen investigadores de multitud de universidades españolas.

2.3. Modelos *Transformer*

Un modelo transformer es una arquitectura de red neuronal que aprende contexto mediante el seguimiento de relaciones en datos secuenciales, a diferencia de arquitecturas como las redes neuronales recurrentes (RNN) o las redes neuronales convolucionales (CNN), que dependían de la estructura secuencial de las entradas. Usa el mecanismo de autoatención, mediante el cual se le da un peso diferente a cada parte de la entrada.

Los transformers fueron por primera vez descritos por investigadores de Google en el paper *All you need is atention* de Vaswani y cols. (2017).

El componente principal de un transformer es el mecanismo de auto-atención, que permite que la red se centre en diferentes partes de la entrada durante el proceso de codificación y decodificación. Esa arquitectura de codificación y decodificación consiste en diferentes capas que procesan la entrada de manera iterativa, al igual que se procesa de forma iterativa la salida.

Esas diferentes capas son varias capas de autoatención (*self-attention* en inglés) y capas de alimentación hacia adelante (*feed-forward* en inglés).

2.3.1. Codificador (*encoder*)

El codificador de un transformer (parte izquierda de la Figura 2.1) es el responsable de transformar secuencias de palabras o tokens en vectores de representación semántica. El codificador se compone de 6 bloques, teniendo cada bloque dos capas, las cuales se definen a continuación:

- Capa de atención multicabeza (multi-head attention layer): Esta capa permite que el codificador aprenda las relaciones a diferentes niveles de abstracción, realizando una atención multicabeza. Esta, se calcula dividiendo los vectores de entrada en múltiples subespacios y realizando la atención en paralelo de cada uno de esos subespacios. Una explicación más formal sería que esta capa implementa h cabezales y recibe una proyección lineal diferente de las consultas, claves y valores (*queries, keys* y *values* en inglés), lo cual produce h salidas en paralelo que se usan para generar el resultado final.

- Red neuronal hacia adelante completamente conectada (*fully connected feed-forward network*): Esta red consta de dos transformaciones lineales separadas por una función de activación no lineal, como puede ser ReLU (de las siglas en inglés de *Rectified Linear Unit*). Se puede definir como

$$FFN(x) = ReLU(W_1 x + b_1)W_2 + b_2$$

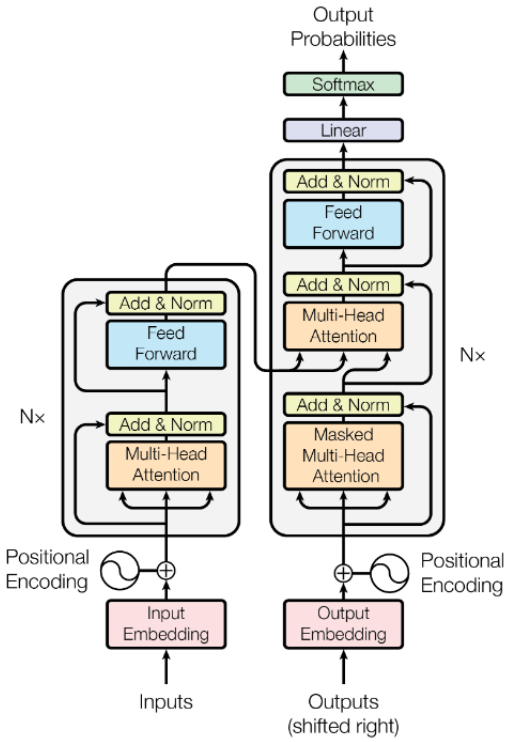

Figura 2.1: Bloque transformer. Figura obtenida del paper Vaswani y cols. (2017)

donde W_1 y W_2 son los pesos y b_1 y b_2 los sesgos.

A cada una de estas dos capas le sucede una capa de normalización. Esta capa se encarga de normalizar la suma entre la entrada de la subcapa y la salida generada por esa misma subcapa. Se puede definir matemáticamente como:

$$layernorm(x + sublayer(x))$$

2.3.2. Decodificador (*decoder*)

El decodificador de un transformer (parte derecha de la Figura 2.1) es el encargado de generar una salida secuencial basada en la información codificada de la entrada. Su objetivo principal es generar una salida relevante basada en la información de entrada.

Al igual que el codificador, el decodificador también se compone de 6 bloques, pero en este caso cada bloque tiene tres capas, las cuales se definen a continuación:

- Capa de atención multicabeza enmascarada (*Masked multi-head attention layer*): La primera capa recibe la salida emitida por el codificador. Se utiliza esta capa para evitar que el modelo preste atención a los pasos de tiempo futuros durante la generación de la salida (de ahí el término máscara). La máscara aplicada garantiza que el decodificador solo conozca la información disponible hasta el momento en el que se genera cada token futuro. A más bajo nivel, esto se consigue introduciendo una máscara sobre los

valores producidos por la multiplicación escalada de las matrices Q y K. La máscara se implementa suprimiendo los valores de la matriz a los cuales el decodificador no puede acceder.

- Capa de atención codificador-decodificador: Esta capa, similar a la primera del codificador, recibe las consultas (*queries*) de la subcapa del decodificador anterior y las claves y valores (*keys* y *values*) de la salida del codificador. Todo esto permite al decodificador tratar todas las palabras de la secuencia de entrada.
- Red neuronal hacia adelante completamente conectada (*fully connected feed-forward network*): Similar a la implementada en el codificador, captura las relaciones no lineales de los datos de salida generados.

Estas tres capas del decodificador tienen entre sí algunas pequeñas conexiones, así como algunas capas de normalización.

Para finalizar esta sección, se puede decir que los transformers son una arquitectura muy utilizada hoy en día en el campo del procesamiento del lenguaje natural, habiendo aportado grandes resultados en tareas como la generación de lenguaje o la traducción automática.

2.4. Modelos BERT

Un modelo BERT (de las siglas en inglés de *Bidirectional Encoder Representations from Transformers*) es una clase de modelo de lenguaje basado en la arquitectura de transformers. Actualmente, son muy usados en el campo del procesamiento del lenguaje natural.

Todo el código de BERT se encuentra publicado en este repositorio de GitHub.

Estos modelos fueron introducidos en 2018 por investigadores de Google, resultados de cuya investigación fueron publicados en el paper *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* de Devlin y cols. (2019).

En sus comienzos, los modelos BERT estaban implementados para ser usados en textos en inglés, teniendo dos versiones de diferente tamaño. $BERT_{BASE}$ contaba con 12 codificadores y 12 cabezales de autoatención multidireccionales, teniendo en total 110 millones de parámetros. Por otro lado, $BERT_{LARGE}$ contaba con 24 codificadores y 16 cabezales de autoatención multidireccionales, teniendo en total 340 millones de parámetros. Ambos fueron entrenados usando un corpus extraído de la Wikipedia en inglés y de *Toronto Book Corpus* (un corpus que tiene unos 11.000 libros extraídos de internet).

A diferencia de los modelos de lengua que existían previamente, los cuales consideraban solo el contexto anterior o posterior, los modelos BERT son bidireccionales. Es decir, utilizan tanto el contexto anterior como el posterior para comprender el significado de una palabra en un determinado contexto.

Los modelos BERT fueron preentrenados simultáneamente en dos tareas:

- Modelado del lenguaje: Se entrenó el modelo con el objetivo de que fuera capaz de predecir el siguiente token dado un contexto. Para ello, se eligió de forma aleatoria el 15% de los tokens del corpus, y de estos, el 80% fueron remplazados por una máscara, el 10% por un token aleatorio y el restante 10% no fue reemplazado. Se utilizó una función de error con entropía para predecir el token original.

- Predicción de la próxima frase: Para entrenar el modelo que pudiera entender las relaciones entre frases, se eligieron frases de todo el corpus, teniendo pares de frases A y B. El 50% de las ocasiones la frase B era la siguiente a la A, y en el otro 50% de las ocasiones, la frase B era una frase aleatoria del corpus.

Tras este primer gran entrenamiento, el modelo BERT puede ser ajustado (*fine-tuned*) para ser optimizado en una tarea concreta, siendo necesarios pocos recursos computacionales en comparación con los necesarios en la fase de entrenamiento.

BERT ha demostrado ofrecer un gran rendimiento en muchas de las tareas que abarca el campo del procesamiento del lenguaje natural, como el etiquetado de entidades nombradas (*NER* en inglés) o la clasificación de texto.

En repositorios como *Hugging Face* se pueden encontrar multitud de modelos que usan BERT como base, los cuales se encuentran reentrenados con corpus específicos y para tareas en concreto.

3. Objetivos

En este apartado se detallarán los objetivos a conseguir una vez finalizado este trabajo. Se desgranan en tres apartados, si bien se adelanta que el objetivo principal será contar como resultado del trabajo con un modelo capaz de anotar textos del dominio de la salud con unas anotaciones determinadas. Se añade una sección extra, para especificar objetivos generales del trabajo.

La forma en la que se van desarrollando acciones para cumplir los objetivos se irá explicando en las siguientes secciones, hasta que en la parte de conclusiones se pueda exponer una visión de en que punto ha quedado el trabajo.

3.1. Corpus

Para poder obtener un modelo que presente buenos resultados, se debe de contar un con corpus de calidad. Por tanto, uno de los objetivos es que la calidad del corpus de este trabajo sea de la mejor posible.

Además, se deben de obtener anotaciones de los textos, especificando en que categorías se quieren clasificar las diferentes anotaciones. Por tanto, se espera poder obtenerlas de la forma más rápida y automatizada posible, para realizar luego una revisión de estas de forma manual.

Por último, otro requisito es poder visualizar de forma fácil y representativa esas anotaciones, en consecuencia, se deberá de buscar una solución a ese problema.

3.2. Modelo

Como se ha mencionado anteriormente, el objetivo es contar con un modelo que detecte todas las enfermedades requeridas. Para conseguir este objetivo, se elegirá un modelo existente que cuente ya con un vocabulario del dominio clínico. El modelo se deberá de ajustar a la tarea que persigue este trabajo, que es la de poder detectar entidades que pertenezcan a varias categorías a la vez, por lo que las anotaciones estarán solapadas tanto de forma exacta (mismo inicio y final), como de forma parcial.

Tras esta elección y ajuste, se deberá de realizar el reentrenamiento (**fine-tunning**), para darle al modelo el nuevo vocabulario, así como el conjunto de etiquetas (tipos de enfermedades y demás entidades a reconocer), para que tras una serie de iteraciones, se obtenga un nuevo modelo.

3.3. Calidad

Al finalizar el reentrenamiento del modelo, este se debe de evaluar. Como se ha comentado anteriormente, la tarea que se persigue es particular, por lo que se debe de pensar como

evaluar los resultados, y una vez pensado, implementar funciones que permitan dado un corpus, obtener una evaluación del modelo.

Una vez se sabe cómo realizar la evaluación, se deben de obtener los resultados, para así poder discutirlos.

3.4. Objetivos generales

Como objetivos generales del trabajo, se busca conseguir una arquitectura que establezca una base de cara a futuros proyectos de similares características.

Como se ha comentado, al ser una tarea particular la de que las anotaciones sean superpuestas, no se cuentan con muchos recursos ya desarrollados para realizar el reentrenamiento y evaluación de estos modelos.

Crear código y funciones genéricas que puedan ser aplicadas independientemente del corpus, categorías y modelos a utilizar, será de utilidad de cara a un futuro en el que se puedan crear proyectos similares.

4. Desarrollo

A lo largo de esta sección se explicará todo el trabajo realizado para conseguir, al final de este proyecto, los objetivos marcados.

Como el principal objetivo es la obtención de un modelo, se dividirá este capítulo en 3 grandes bloques:

- Corpus: Se detallará todo el proceso y trabajo realizado para la obtención de un corpus sólido, textos con los que se realizará el reentrenamiento del modelo. Además, se analizará la distribución que tienen los datos y se verá que es necesario obtener varios corpus.

- Selección y adaptación de un modelo: Se indicará el modelo elegido como base, el cual se reentrenará. Se explicará como se realizan ajustes en el modelo para la tarea de este trabajo.

- Reentrenamiento: Una vez se cuenta con el corpus y el modelo base adaptado, se explicará el proceso seguido para reentrenarlo.

- Inferencia: Al contar ya con un modelo reentrenado, esta breve sección detallará cómo usarlo para inferir nuevos datos.

- Métricas implementadas: Se explicarán las métricas implementadas para medir la destreza del nuevo modelo.

4.1. Corpus

El corpus es una de las partes más importantes a la hora de confeccionar el nuevo modelo. El texto y sus anotaciones serán lo que aprenderá el modelo. Por ello, es crucial que la calidad de las anotaciones sea la mejor posible.

Se describirá a continuación como se han obtenido los textos y las anotaciones. Antes de eso, se muestra la Figura 4.1, en la cual se detallan todos los pasos seguidos desde que se obtienen los textos a analizar, hasta que se genera el conjunto de entrenamiento y de prueba.

Para la creación de este modelo, se han elegido en torno a 1400 textos, siendo todos ellos resúmenes (*abstracts*) de textos científicos (*papers*) del ámbito médico.

Esa selección se ha realizado en base a la temática principal de aspectos que se persiguen detectar, que son las enfermedades raras y neurodegenerativas, como pueda ser la enfermedad de Huntington, la esclerosis múltiple, o el Alzheimer. Además de detectar las enfermedades en sí, se pretenden anotar aspectos relacionadas con las mismas, como medicamentos, causas, pruebas clínicas...

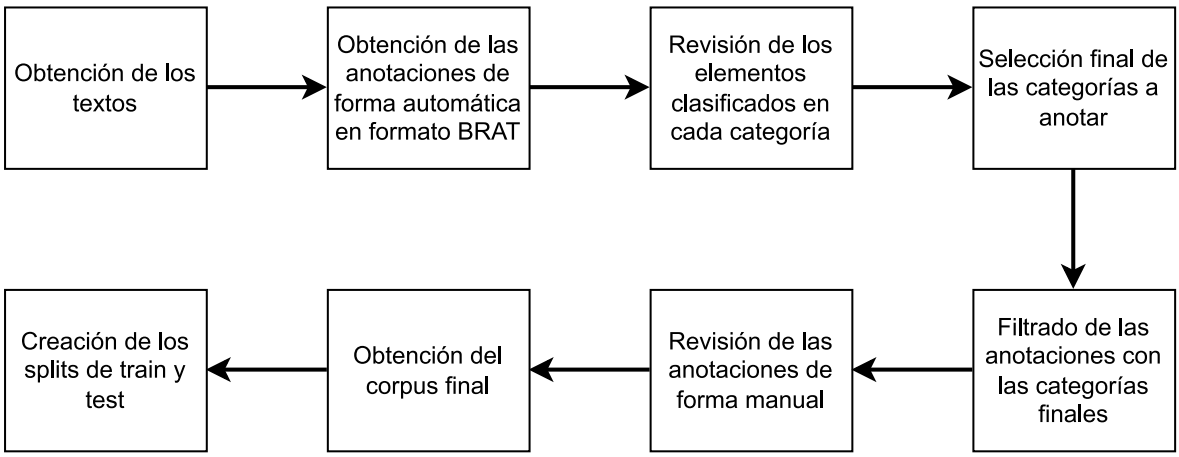

Figura 4.1: Proceso seguido a la hora de elaborar el corpus

Se han recopilado esos documentos y extraído su resumen. Cada texto se ha almacenado en un archivo de texto (archivo *.txt*) diferente, nombrándolos de forma secuencial (text0.txt, text1.txt...).

Una vez se tenían los documentos cada uno almacenado por separado, se ha procedido a generar las anotaciones. Para obtenerlas, se ha buscado la forma de hacer el proceso lo más automatizada posible, para no tener que anotar de forma manual todo el corpus.

Es por esto por lo que se ha utilizado UMLS.

UMLS (de las siglas de *Unified Medical Language System*) es un sistema de lenguaje médico desarrollado por la Biblioteca Nacional de Medicina de Estados Unidos (mismos desarrolladores que de *PubMed*). Este sistema contiene vocabulario, clasificaciones y ontologías del ámbito médico.

Entre las diferentes herramientas que ofrece se encuentra *Metathesaurus*. Esa herramienta es una base de datos que contiene más de 3 millones de conceptos (también del ámbito médico). Cada concepto tiene un identificador único llamado *CUI*, además de contener sus sinónimos, definiciones, relaciones con otros conceptos, categorías...

De esta herramienta, lo que interesa en este caso son las diferentes categorías semánticas. Cada concepto incluido en *Metathesaurus* está relacionado con uno o varias de ellas.

Entre estas categorías semánticas podemos encontrar aspectos muy relacionados con el ámbito clínico, como *Disease or Syndrome* ("Enfermedad o Síndrome" en castellano), pero también categorías más genéricas, como *Organization* ("Organización" en castellano).

Para realizar las anotaciones del texto se ha utilizado *spaCy* y *scispaCy*. *spaCy* es una biblioteca de procesamiento del lenguaje natural para Python. Junto a *spaCy*, se ha utilizado *scispaCy*. Concretamente. se ha utilizado la tubería *en_core_sci_sm*, la cual ha sido diseñada específicamente para procesar datos biomédicos. Junto a ella, se ha utilizado un detector de abreviaturas (componente *abbreviation_detector*), así como el componente *EntityLinker*, el cual permite detectar relaciones gracias a cierta base de conocimiento. En este caso, y tal y como se ha explicado anteriormente, la base de conocimiento escogida ha sido UMLS.

En el Código 4.1 se muestra cómo se ha construido todo lo descrito anteriormente para poder posteriormente realizar las anotaciones del texto.

Código 4.1: Utilización de spaCy y scispaCy

```
# Import the scispacy model
nlp = spacy.load("en_core_sci_sm")

# Add abbreviation pipe to the spacy pipeline
nlp.add_pipe("abbreviation_detector")

# Add the scispacy entity linker to the spacy pipeline
nlp.add_pipe("scispacy_linker", config={"linker_name": "umls", "resolve_abbreviations": True})

# Get the scispacy entity linker from the spacy pipeline
linker = nlp.get_pipe("scispacy_linker")
```

Se realizó una primera iteración para generar las anotaciones de los textos. Ahora, para cada texto se contaba con un fichero de texto (*.txt*) que contiene el texto en sí, y un fichero de anotaciones (*.ann*) que contiene las anotaciones generadas.

Ese fichero de anotaciones ha sido generado en formato *BRAT*. Este formato, llamado así de las siglas en inglés de *Brat Annotation Tool*, es un programa libre para anotar textos.

Los ficheros con anotaciones en *BRAT* no tienen cabecera, y en cada línea se especifica un número secuencial de anotación, la categoría semántica, la posición de comienzo y fin de la palabra, y la palabra anotada (puede ser palabra o multipalabra).

En el Anexo I se muestra un ejemplo del texto de entrada y un ejemplo del fichero de anotaciones en formato *BRAT*. Además, se muestra como se representan las anotaciones en el servidor montado de BRAT.

Al finalizar, se utilizaron esas primeras anotaciones en la que se generaron etiquetas para todos los textos. Se procesaron los archivos para generar uno por cada tipo de categoría semántica que contuviera todas las palabras detectadas en los textos de esa categoría.

Metathesaurus en total tiene 127 categorías semánticas, algunas de ellas irrelevantes para este trabajo. Por tanto, estos nuevos archivos servirían para decidir con cuáles categorías se entrenaría el modelo y cuáles serían descartadas.

Algunas de las categorías descartadas son *Bird*, *Animal* o *Amphibian* (de nada sirve detectar animales), *Event*, *Geographic Area* o *Governmental or Regulatory Activity* (categorías muy generales, no cercanas al dominio de interés).

Tras filtrar las categorías de interés, se tienen en total 90 categorías semánticas que serán las que se utilizarán para entrenar el modelo y las que este detectará.

En el Anexo II se muestra una tabla en la que se describen las categorías seleccionadas y las descartadas.

Habiendo realizado el filtrado, se volvieron a generar todos los ficheros de anotación de los textos para que solo incluyeran entidades clasificadas de las categorías seleccionadas.

Una vez se tenían las anotaciones filtradas, fueron revisadas por personas expertas en el dominio del problema para, ahora ya manualmente, comprobar si las anotaciones generadas eran correctas, detectar problemas con las etiquetas, y poder al finar contar con un corpus procesado mecánicamente y revisado manualmente.

Concretamente, de los 1400 textos que tenía el corpus, se cuenta con 300 textos (un 10%) revisados manualmente. No se pudieron tener más por limitaciones de tiempo por parte de las personas expertas en el dominio. Por lo tanto, se debía de tomar una decisión acerca de como distribuir el corpus que se tenía en dos conjuntos, el de entrenamiento y el de test.

Tal y como se muestra en la Figura 4.2, el conjunto de entrenamiento se ha construido usando 600 textos sin revisar y 200 revisados, 800 en total. Por otro lado, el conjunto de

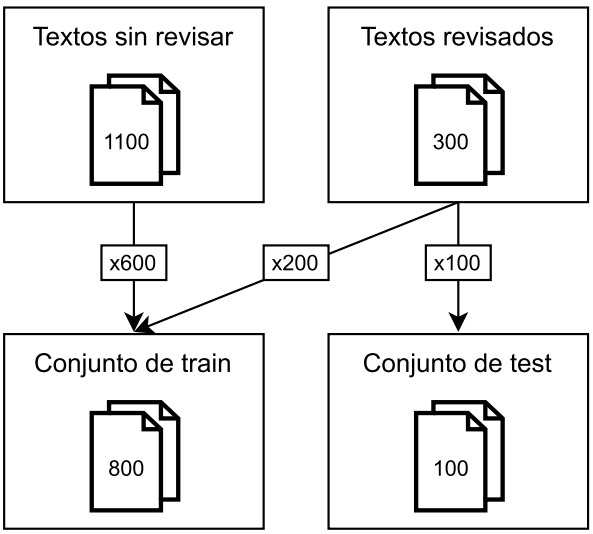

Figura 4.2: Distribución del corpus en conjunto de entrenamiento y test

prueba se ha construido usando 100 textos revisados.

En total, el conjunto de entrenamiento tiene 88667 anotaciones y 6989 frases, y el conjunto de prueba 9044 anotaciones y 902 frases.

En cuanto a cómo se formatean los datos, se ha comentado anteriormente que el objetivo es tener las anotaciones en formato BRAT para poder utilizar posteriormente este sistema de visualización. El modelo no admite recibir datos en este formato (ni tampoco genera anotaciones en este formato), por tanto, hay que formatear los datos para que estén en un formato que se admita como entrada.

Para ello, se procesan los ficheros de anotaciones en formato BRAT y se extraen de ellos todas las anotaciones. Además, se leen los archivos txt que contienen el texto en sí.

Lo primero que se hace es la lectura de los textos. Con todos ellos, se crea un vector, en el que cada posición contiene un diccionario que contiene:

- tags: Vector con las anotaciones. Por ahora vacío.
- id: Identificador del texto, es decir, el número de texto que se encuentra en el nombre del fichero. Este identificador coincide con el del archivo de anotación, por lo que se usa para poder cuadrar el texto con sus anotaciones. No se puede llevar a cabo usando la posición en el índice del array ya que la numeración de los textos es secuencial, pero falta algún número, por lo que se perdería la referencia.
- text: Texto en sí leído de los archivos txt.
- sentences: Vector que contiene la posición final de cada frase del texto. Esto se realiza viendo cuando acaba una frase, dado el símbolo '.', teniendo en cuenta aspectos como no encontrar puntos suspensivos o puntos usados con otro uso que no sea el de delimitar una frase. Estos índices serán usados al obtener las métricas del modelo.

Tras haber leído los textos, así como haber calculado el vector *sentences*, ya se tiene un

vector creado. Ahora, ese mismo vector es el que se usa para ir leyendo todos los archivos de anotaciones.

Los archivos de anotaciones revisadas y sin revisar se guardan físicamente en directorios diferentes, para poder tener una referencia de cuales son revisados y cuales no. Por tanto, a la hora de leerlos, se leen por separado, obteniendo dos vectores de los anteriormente comentados.

Para cada anotación (representada en una línea del fichero), se guarda la posición de inicio, la de final y la clase en un diccionario de *Python*, teniendo como claves *start*, *end* y *tag*.

Con esos dos vectores ya totalmente completos (con los textos, sus anotaciones, índices y las frases), se procede a construir el conjunto de entrenamiento y prueba, usando la distribución de textos anteriormente comentada. Por tanto, dos nuevos vectores se crean, uno para cada partición de datos.

Esos vectores se crean por duplicado, una pareja de vectores se crea para ser la entrada del modelo, y el otro par para ser utilizados para medir el rendimiento del modelo. De la pareja usada como entrada del modelo, se elimina el campo *sentences* de cada diccionario del vector. Esto es así al no poder procesar el modelo este campo, por lo cual se lanza un error en caso de encontrarlo.

Una vez se tiene toda es información procesada y almacenada, se hace uso de la librería *Datasets*, creada por *Hugging Face*.

Se crea un *dataset* usando el formato JSON creado anteriormente. Contar con un dataset creado con esta librería facilita el proceso de generación de los diferentes conjuntos de datos, así como el proceso de reentrenamiento del modelo.

Como el reentrenamiento se realizará usando una máquina virtual en la nube, ese JSON será el archivo que se subirá para luego capturar los datos y realizar el reentrenamiento.

4.1.1. Análisis de la distribución de los datos

Como se ha comentado, se usarán 800 textos para el conjunto de entrenamiento y 100 para el conjunto de prueba. Además, se ha visto que se tienen 90 categorías posibles para clasificar elementos.

En la Figura 4.3 se muestra la distribución de los datos de ambos conjuntos.

Como se puede observar, los datos se encuentran fuertemente desbalanceados, observando que hay una clase predominante (*DiseaseOrSyndrome*), y clases que apenas cuentan con instancias. Habiendo tantas clases es complicado apreciarlas todas en el gráfico mostrado.

Viendo que hay esta gran cantidad de clases, el fuerte desbalance en los datos, y la poca cantidad de datos en alguna de las clases, es necesario tomar acciones para atajar estos problemas, ya que dejar el corpus así desembocaría previsiblemente en obtener un nuevo modelo de mala calidad.

Por ello, es necesario realizar un balanceo de datos y crear nuevos corpus que contengan un mejor balanceo de etiquetas por cada categoría.

En este caso, el balanceo no es un proceso trivial, ya que si se quieren eliminar instancias de una clase, se deben de eliminar no solo las etiquetas de la clase, sino también el texto en el que está contenido. Por tanto, al eliminar el texto, también se eliminan instancias de otras clases que aparecían en el texto. Eliminar instancias de una clase conlleva eliminar instancias de otras clases.

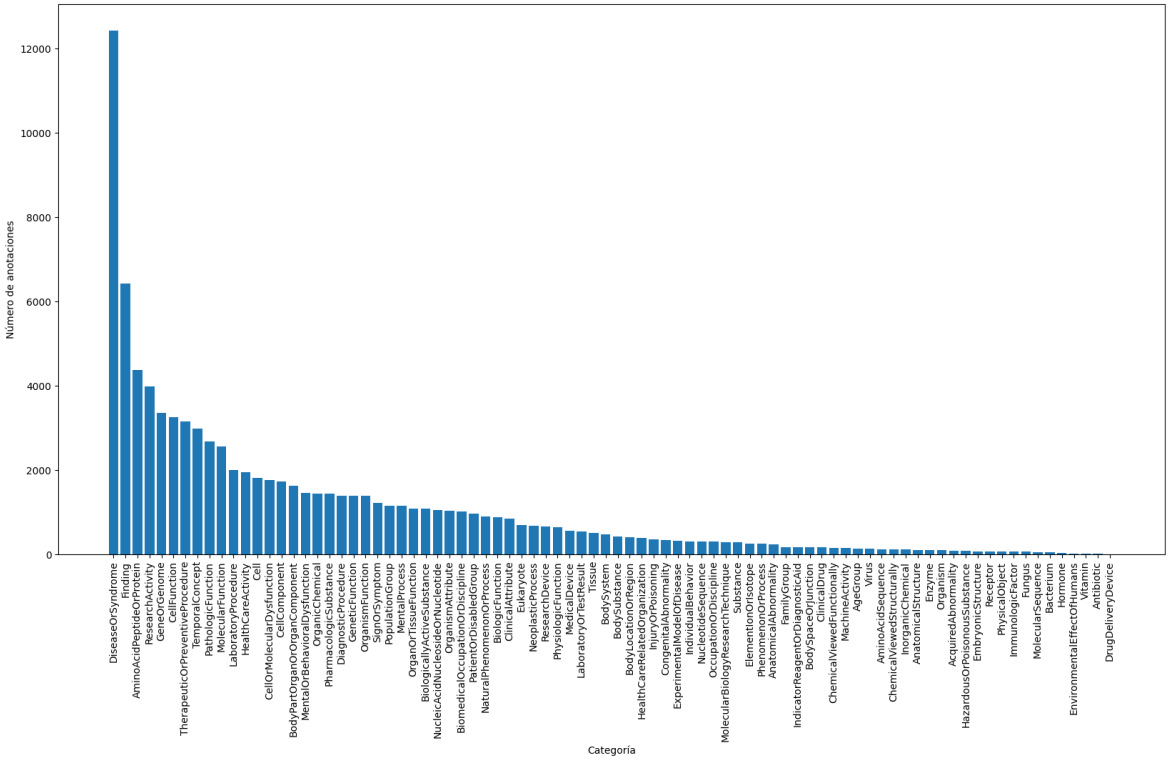

Figura 4.3: Distribución de etiquetas de las diferentes categorías en el corpus original

Para comprender mejor lo que se acaba de explicar, se explica sobre un ejemplo real. Dada la Figura A.1, en la que se muestran las anotaciones de las tres primeras frases de un texto, se propone eliminar la clase *DiseaseOrSyndrome* (la que se ha visto que más predomina). Centrándonos en la primera frase, al eliminar esa categoría, se tiene también que eliminar el texto en el que aparece, es decir, la frase entera (no se puede quitar solo la palabra anotada, la frase perdería el sentido). Al quitar toda la frase, categorías como *Finding* o *Cell* también se verían afectadas. Por tanto, se disminuiría la cantidad de etiquetas de múltiples clases.

Estos motivos explicados sirven para justificar que en este trabajo es muy complicado obtener corpus estrictamente balanceados, por tanto, se balancearán en la medida de lo posible, intentando conseguir un equilibrio aceptable dentro del desequilibrio a asumir al tener varias clases de etiquetas en una misma frase.

Para realizar esta ardua tarea, se han definido una serie de funciones que facilitan y automatizan lo máximo posible esta tarea.

Entre estas funciones, encontramos:

- Una función que calcula el número total de etiquetas por categoría dado un corpus.
- Una que elimina las etiquetas de una categoría determinada en un elemento (eliminando también, como se ha comentado antes, el resto de etiquetas involucradas en ese texto eliminado).
- Una que realiza un aumentado de datos, creando nuevos elementos en base a texto y

anotaciones de otros elementos, y aumentando así los elementos de una categoría en concreto (y también de categorías que se encuentren en el texto elegido).

- Una función que realiza una reducción de datos, eliminando textos y anotaciones de cierta categoría (y, como se ha dicho, también se eliminan categorías involucradas en ese texto).
- Una función que elimina las categorías que tienen menos de un cierto número de elementos a lo largo de un corpus.

Una vez creadas estas funciones, se pueden crear nuevos conjuntos de datos más balanceados, así como eliminar categorías con pocas anotaciones.

El proceso ahora es sencillo. En el código 4.2 se muestra como se pueden eliminar del corpus categorías con menos de 500 etiquetas, así como realizar una normalización de los datos de entrenamiento, indicando que se quieren tener 400 instancias de cada una de las categorías mantenidas.

Código 4.2: Eliminación de categorías con menos de 500 etiquetas y balanceo del conjunto de entrenamiento

```
1 trainDataCopy, testDataCopy, listaCategoriasNew = deleteCategLessThan(dfAnnPerCatTrain, listaCategorias,
      trainDataCopy, testDataCopy, n = 500)
2
3 normalizaDatasets(trainDataCopy, dfAnnPerCatTrain, 400)
```

Por tanto, ahora se pueden crear nuevos conjuntos de entrenamiento y de test, con datos más balanceados y limpios. Por ello, se definen a continuación los conjuntos creados.

4.1.1.1. Conjunto de datos 1

Este es el conjunto de datos original, en el que se tienen 800 textos para el conjunto de entrenamiento (600 sin revisar y 200 revisados) y 100 para el de prueba (todos ellos revisados).

Se detalla a continuación más información de los datos:

- Número de categorías: 91
- Total de elementos en train: 800
- Total de elementos en test: 100
- Número total de etiquetas en train: 88667
- Número total de etiquetas en test: 9044
- Número total de frases en train: 6989
- Número total de frases en test: 902

La distribución de estos datos es la mostrada en la Figura 4.2. Como se ha comentado anteriormente, esta distribución es muy dispar, habiendo categorías con muchísimas anotaciones y categorías con apenas etiquetas. Además, en este primer corpus había categorías en el conjunto de prueba que no se encontraban en el conjunto de entrenamiento y viceversa, cosa que perjudica gravemente a los resultados. Por todo ello, se crea el siguiente conjunto de datos.

4.1.1.2. Conjunto de datos 2

A partir del conjunto 1, se crea este. Como estrategias a la hora de crearlo, se han eliminado categorías con pocas etiquetas, se ha realizado un aumentado y disminución de datos para intentar dejar el corpus lo más equilibrado posible, y también se ha tenido en cuenta que en ambos conjuntos hubiera las mismas categorías, es decir, que no pasara como en el conjunto anterior donde por ejemplo categorías que estaban en el conjunto de prueba no estuvieran en el de entrenamiento.

Se muestra a continuación diversa información relativa a este conjunto de datos.

- Se eliminan las categorías con menos de 1000 etiquetas
- Número de categorías que se mantienen: 30
- Se hace una normalización del conjunto de train, buscando tener 200 etiquetas de cada categoría
- Se hace una normalización del conjunto de test, buscando tener 50 etiquetas de cada categoría
- Total de elementos en train tras la normalización: 967
- Total de elementos en test tras la normalización: 378
- Número total de etiquetas en train tras la normalización: 12148
- Número total de etiquetas en test tras la normalización: 2891
- Número total de frases en train tras la normalización: 1255
- Número total de frases en test tras la normalización: 459

En la Figura 4.4 se muestra la distribución de este nuevo corpus. Como se puede observar, ahora hay bastantes menos clases. Respecto a la distribución, se comprueba que no es uniforme el número de etiquetas a lo largo de las diferentes categorías. Esto es debido a lo explicado anteriormente, de que dada la naturaleza del problema, es prácticamente imposible contar con corpus completamente balanceados. Si bien, ahora hay mejor balanceo que en el corpus original.

Todos estos conjuntos de datos creados servirán para crear diferentes modelos, y poder así comprar los resultados que se obtengan.

4.2. Selección y adaptación de un modelo

La primera decisión a la hora de realizar este trabajo fue si entrenar el modelo desde cero, o escoger uno ya creado y entrenado.

Sabiendo que el corpus que se iba a tener no era muy extenso (eran como mucho mil textos), se decidió por no realizar el modelo desde cero, ya que al no contar con un gran volumen de datos, los resultados de este modelo creado nuevo hubieran sido previsiblemente malos.

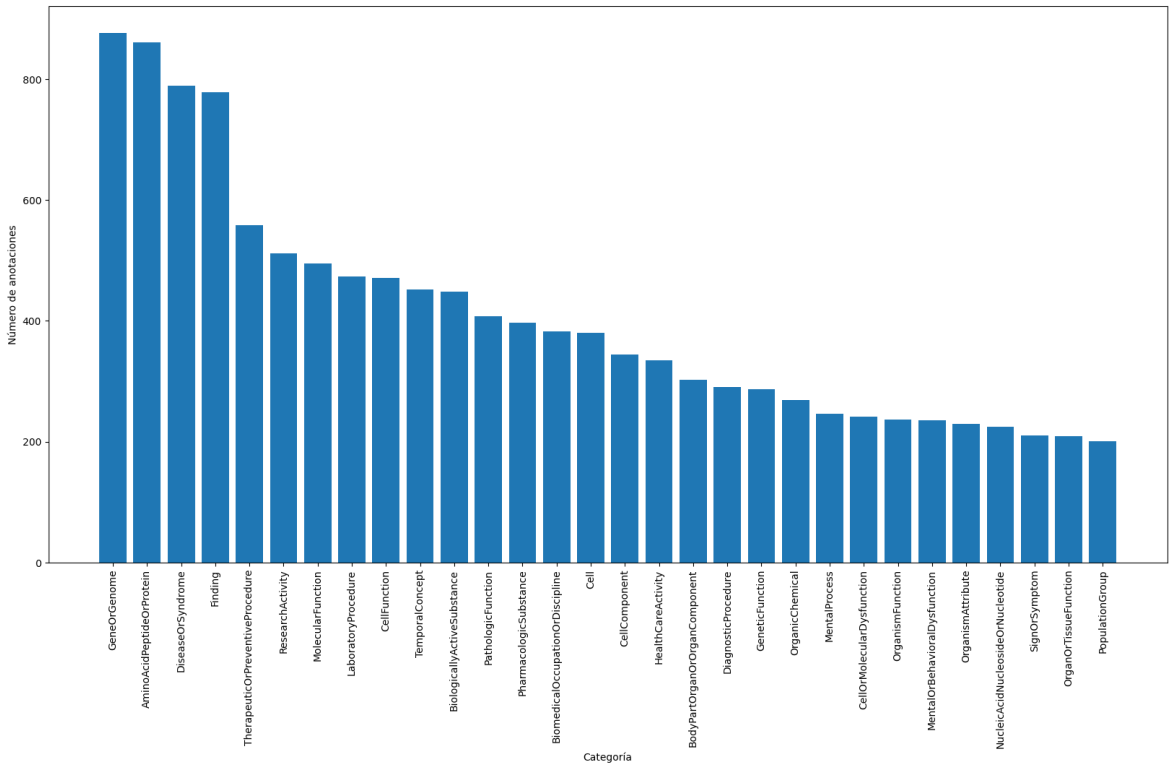

Figura 4.4: Distribución de etiquetas de las diferentes categorías en el conjunto de datos 2

Por tanto, una vez decidido que se iba a partir de un modelo ya entrenado, había que decidir uno.

Una de las ventajas de realizar un reentrenamiento de un modelo ya existente frente a crear uno desde cero, es poder aprovecharse del vocabulario previo que el modelo ya conoce. Es por esto por lo que el modelo elegido debería de haber sido entrenado con vocabulario similar al que se cuenta para este trabajo, vocabulario del ámbito clínico.

Tras realizar una búsqueda en el famoso repositorio de modelos *Hugging Face*, se encontró un modelo creado por Microsoft, llamado "*microsoft/BiomedNLP-PubMedBERT-large-uncased-abstract*", cuyo desarrollo está recogido en el paper de Tinn y cols. (2021). Este modelo, el cual se puede encontrar en esta página, fue creado utilizando como base un modelo BERT.

Este modelo parte del ya creado **BertForTokenClassification**, el cual a su vez parte del modelo *BERT*, tomando en este caso su versión $BERT_{base}$. Este modelo consta de:

- Capa de entrada: Capa a la que le llega la entrada del modelo, como una secuencia de tokens. Un vector de embeddings de palabras representa a cada uno de los tokens.

- Capas transformer: En esta versión de *BERT* se usan 12 capas de transformers. Cada una de estas 12 capas tiene a su vez dos principales capas: la capa de atención multicabeza y una capa con una red neuronal hacia adelante completamente conectada (tal y como se detalló en la Sección 2.3).

- Capa de salida: Esta capa se adapta a cada una de las diferentes tareas para las que se puede usar un modelo *BERT*. En el caso de una tarea de reconocimiento de entidades nombradas, se añade una capa que clasifica tokens.

Ese modelo fue sometido a un proceso de reentrenamiento usando resúmenes (*abstracts*) de textos científicos (*papers*) del ámbito biomédico en inglés. Los documentos mediante los que se ha entrenado este modelo se han extraído íntegramente de *PubMed*, base de datos de acceso libre y gratuito de la Biblioteca Nacional de Medicina de los Estados Unidos. Esta biblioteca contiene más de 32 millones de documentos de todo tipo (resúmenes, artículos científicos, revistas médicas...).

En total, el texto con el que se ha reentrenado consiste en 14 millones de *abstracts*, 3.100 millones de palabras, ocupando un espacio de 21GB de memoria.

Se comenta que se ha reentrenado varias veces, haciendo un reentrenamiento en algunos casos adaptado a un dominio en concreto, y en otros casos para un propósito general. Para cada caso, en el *paper* se comenta la arquitectura con la que cuenta el modelo. En el caso de la detección de entidades nombradas (*NER*), se utilizan como capas de clasificación, capas lineales, *LSTM* y *CRF*.

Viendo los resultados que los investigadores consiguen con este modelo, son muy prometedores, al superar evaluando datasets como *BC5-chem* o *ChemProt*, a modelos como *RoBERTa*, *BioBERT* o *ClinicalBERT*.

Una vez elegido el modelo, este se debe de adaptar para la tarea que persigue este trabajo. El problema al que se está planteando una solución requiere aprender y predecir tokens tanto de manera individual (entendiendo que una palabra o un conjunto de palabras pertenecen a una categoría semántica) como de manera solapada y/o múltiple (una palabra o conjunto de palabras pueden pertenecer a más de una categoría, por lo que se pueden solapar de forma exacta, con el mismo comienzo y final, o de forma inexacta, comienzan y/o acaban en diferentes posiciones las categorías que se solapan). A este tipo de problemas se les llama *Span categorization problems* (problemas de categorización por tramos).

Por tanto, para abordar este problema, se ha creado una nueva clase llamada *BertForSpanClassification*, similar a las implementadas por BERT. De esta clase, se deberá de ajustar la función de pérdida y la función de activación, ya que la clasificación por tramos (*Span categorization*) realiza la clasificación multietiqueta a nivel de token, y la clasificación de tokens (*Token categorization*) realiza la clasificación multiclase (que no multietiqueta) a nivel de token. La diferencia entre la multietiqueta y la multiclase es que la multiclase para un token busca solo una posible clase, entre una lista de más de dos clases. Por otro lado, la multietiqueta busca para un token, la posibilidad de que pertenezca a más de una clase dentro de una lista de más de dos clases.

De la función de pérdida, al estar clasificando tokens de múltiples clases y que se pueden solapar, se usará una *binary cross-entropy loss*. Respecto a la función de activación de la última capa del modelo, se usará una función sigmoide, ya que se debe de tener la posibilidad de obtener más de una respuesta correcta del modelo (es decir, un token puede pertenecer a más de una clase).

En el Código 4.3 se puede ver parte de la clase creada. Se muestra solo la parte en la que se ha modificado la función de pérdida, usando *BCEWithLogitsLoss* (función de *binary cross-entropy loss*). Al usar esta función, la librería *torch* ya entiende que se va a utilizar una función sigmoide como función de activación.

Código 4.3: Clase creada BertForSpanClassification

```
BertForSpanClassification(BertPreTrainedModel):
    ...
    def forward(
    ...
    ) -> Union[Tuple[torch.Tensor], TokenClassifierOutput]:
        ...
        loss = None
        if labels is not None:
            loss_fct = BCEWithLogitsLoss()
            loss = loss_fct(logits, labels.float())
        ...
    ...
```

A continuación se explicará el proceso de reentrenamiento realizado.

4.3. Reentrenamiento

Una vez ya se cuenta con un corpus procesado y un modelo elegido, se puede comenzar la fase de reentrenamiento, en la cual usando los diferentes corpus obtenidos, se podrá reentrenar el modelo para así conseguir nuevas versiones del mismo.

Cabe mencionar que se han definido ciertas funciones y elementos auxiliares de cara al reentrenamiento. Se han creado tres diccionarios que convierten de id a categoría o viceversa. Son los siguientes:

- tag2id: Diccionario que tiene como clave las diferentes categorías y como valores un número asignado a cada categoría de forma secuencial.
- id2label: Diccionario que tiene como clave un número asignado de forma secuencial, y como valor tiene para cada categoría dos elementos, uno que es el texto "B-" más el nombre de la categoría y otro que es el texto "I-" más el nombre de la categoría. Este se ha definido ya que internamente el modelo trabaja con el formato IOB (de las siglas *Inside, Outside, Begenning*), formato utilizado comúnmente en el procesamiento del lenguaje natural para etiquetar y anotar entidades de un texto. La etiqueta que comienza por "B-" representa el inicio de una entidad y la etiqueta que comienza por "I-" indicar que ese token es parte de una entidad.
- label2id: Diccionario cuyas claves y valores son las inversas a i2label. Ahora las claves son las etiquetas de las categorías y los valores los números secuenciales.

Las funciones auxiliares que se definen se utilizan para realizar una correcta tokenización del texto y realizar una correcta gestión de las etiquetas en el formato IOB, haciendo uso de los diccionarios comentados anteriormente.

Se hace uso de la clase *AutoTokizer* para importar el tokenizador que nos da el modelo elegido. Además, en base a ese tokenizador, se importa usando la clase *DataCollatorWithPadding* un agregador de datos (*Data Collator* en inglés) que se encargará de organizar los datos, garantizar que estén en un formato consistente y, en general, prepararlos de forma adecuada de cara al reentrenamiento. Por último, se importa el modelo haciendo uso de la clase *AutoModelForTokenClassification*.

Como parámetros de reentrenamiento, se ha definido un objeto de *TrainingArguments*, en el cual se definen aspectos, estando entre ellos:

- Directorio de salida del modelo: Se define la ruta donde se almacenará el modelo resultante del reentrenamiento.
- Estrategia de evaluación: Se especifica cuándo se debe de realizar la evaluación del modelo. En este caso se define que se realice después de cada época.
- Tasa de aprendizaje: La tasa de aprendizaje define como de rápido o lento un modelo aprende de los datos. Se ha definido con un valor de 0.00025.
- Número de épocas de entrenamiento: Se define el número de épocas (*epochs* en inglés) que se ejecutarán durante el proceso de reentrenamiento. En este caso se ha reentrenado con 100 épocas.
- Métrica para elegir el mejor modelo: Se elige la métrica que se usará a la hora de decidir cuál es el mejor modelo resultante, siendo en este caso la métrica de F1-score.

El reentrenamiento se realizará usando la clase *Trainer* desarrollada por Hugging Face. Por tanto, se ha creado un objeto de esa clase, en la cual se especifican los siguientes aspectos:

- Modelo de inicio: Se especifica el modelo base de cara al entrenamiento. En este caso, se ha creado una función auxiliar que retorna un objeto creado por la clase definida anteriormente *BertForSpanClassification*, en la que se especifica el modelo base, y las funciones auxiliares id2label y label2id.
- Argumentos: Es el objeto de *TrainingArguments* definido anteriormente.
- Datos de entrenamiento: Conjunto de datos de entrenamiento.
- Datos de evaluación: Conjunto de datos de evaluación.
- Agregador de datos: Es el agregador de datos definido anteriormente.
- Tokenizador: Tokenizador importado anteriormente.
- Cálculo de métricas: Función definida para calcular las métricas durante el proceso de reentrenamiento. Posteriormente, se explicará cómo se realizan estas métricas, ya que son diferentes de las que se usarán a la hora de evaluar el modelo.

Para realizar el reentrenamiento, se ha hecho uso de Kaggle. Esta plataforma te permite ejecutar cuadernos de Python en una máquina virtual que te proporciona. La principal ventaja encontrada de usar Kaggle respecto a ejecutar el código en una máquina local o en plataformas similares como Google Colab, es que te permite seleccionar la GPU a utilizar en la máquina virtual. En este caso, se ha seleccionado la opción de dos GPUs Tesla T4. Esto sería justamente el doble de los recursos que proporciona Google Colab de manera gratuita, ya que en ese plan solo se puede hacer uso de una Tesla T4, mientras que en Kaggle se pueden usar dos.

El reentrenamiento de cada uno de los modelos se ha realizado en 100 épocas, las cuales tardan aproximadamente una hora y media. Al finalizar, se han guardado los nuevos modelos, para poder importarlos posteriormente y ser usados.

Esos archivos se pueden importar en local para realizar pruebas de rendimiento de los nuevos modelos, así como realizar inferencia de nuevos textos.

En el último apartado de este capítulo se detallará como se ha medido el rendimiento de estos nuevos modelos, así como la explicación de las funciones de métricas implementadas.

4.4. Inferencia

Como breve apartado, se explica aquí como se podría realizar inferencia de nuevos textos usando el modelo reentrenado.

En el Código 4.4 se puede ver como de simple es importar el nuevo modelo y el tokenizador. Para importar el modelo se define la ruta donde se encuentra los archivos del modelo reentrenado (que previamente se han descargado a la máquina local tras el proceso de reentrenamiento realizando en la nube). Respecto al tokenizador, se usa el mismo del modelo base escogido.

Código 4.4: Importación del modelo reentrenado y el tokenizador

```
model = BertForSpanClassification.from_pretrained("./models/finetuned_model")
tokenizer = AutoTokenizer.from_pretrained("microsoft/BiomedNLP-PubMedBERT-base-uncased-abstract")
```

Una vez se han importado, se ha definido una función para dado un texto, el modelo y el tokenizador, se obtenga un fichero de texto que contenga el texto en sí y un fichero de anotaciones, con las anotaciones predichas por el modelo en formato BRAT.

Además, se han definido dos funciones auxiliares que hacen posible la inferencia. La primera de ellas tokeniza el texto y ajusta las etiquetas a los tokens. La segunda, realiza la inferencia del modelo sobre un texto, utilizando la función anterior. La salida de esa segunda función son vectores con posiciones iniciales, finales y etiquetas predichas, por lo tanto, la función principal comentada en el apartado anterior es la que se encarga de procesar estos datos para obtenerlos finalmente en formato BRAT.

Por tanto, gracias a todas esas funciones auxiliares definidas, el proceso de inferencia se simplifica. Se muestra un ejemplo en el Código 4.5

Código 4.5: Inferencia de un texto usando el nuevo modelo

```
example = "Huntington's disease is a neurodegenerative autosomal disease results due to expansion of polymorphic
    CAG repeats in the huntingtin gene. Phosphorylation of the translation initiation factor 4E-BP results in
    the alteration of the translation control leading to unwanted protein synthesis and neuronal function.
    Consequences of mutant huntington (mhtt) gene transcription are not well known. Variability of age of onset
    is an important factor of Huntington's disease separating adult and juvenile types. The factors which are
    taken into account are-genetic modifiers, maternal protection i.e excessive paternal transmission, superior
    ageing genes and environmental threshold. A major focus has been given to the molecular pathogenesis which
    includes-motor disturbance, cognitive disturbance and neuropsychiatric disturbance. The diagnosis part has
    also been taken care of. This includes genetic testing and both primary and secondary symptoms. The
    present review also focuses on the genetics and pathology of Huntington's disease."
inferText(example, model, tokenizer)
```

Epoch	Training Loss	Validation Loss	F1 B-aminoacidpeptideorprotein	F1 I-aminoacidpeptideorprotein	F1 B-biologicallyactivesubstance	F1 I-biologicallyactivesubstance	F1 B-biologicfunction
1	No log	0.039025	0	0	0	0	0
2	No log	0.013812	0	0	0	0	0
3	No log	0.009669	0	0	0	0	0

Figura 4.5: Captura realizada durante el reentrenamiento del modelo

Tras ejecutar la función, se generan dos archivos, uno en formato *.txt* que contiene el texto analizado, y otro en formato *.ann* que contiene las anotaciones en formato BRAT.

4.5. Métricas implementadas

En este apartado final de la parte de desarrollo del trabajo, se explicará como se ha medido el rendimiento de los nuevos modelos creados, mostrando también los resultados de esas mediciones.

Para comenzar, cabe explicar las diferencias de medir a nivel de token (*token-based*) o a nivel de entidad (*entity-based*).

Durante el proceso de reentrenamiento, se ha de medir a nivel de token, ya que como se ha comentado anteriormente, el modelo procesa el corpus en formato IOB (con etiquetas "B-" e "I-"). Por tanto, se desarrolla una función para realizar este tipo de mediciones, que es la que se le pasa al objeto de la clase *Trainer*.

Esta función recibe una tupla, en la que se encuentran las etiquetas predichas y las reales, teniendo para cada elemento, los logits devueltos por el modelo. Para esos logits se define un umbral a partir del cual se tendrán en consideración, siendo en este caso ese umbral 0.5. Una vez se tienen los logits a tomar en consideración, se computa la matriz de confusión para todas las categorías semánticas del modelo (a excepción de la categoría "O", la cual se asigna a los tokens que no son clasificados como de ninguna categoría). Para computar la matriz de confusión, se utiliza el diccionario *id2label* definido anteriormente.

Una vez se cuenta con la matriz de confusión, se puede pasar a calcular las métricas de precisión, cobertura y la métrica F1-score a nivel de token, mostrando una tabla con todos los resultados por categoría.

Se muestra en la Figura 4.5 una captura realizada durante el reentrenamiento del modelo, en la que se puede ver como por cada iteración realizada, se obtienen las métricas F1-score a nivel de token.

Todo esto en cuanto a las métricas que se muestran durante el reentrenamiento, las cuales van dando una idea de como va progresando (o no) el modelo conforme aprende del corpus mostrado.

Pero lo verdaderamente importante son las métricas definidas para evaluar el modelo reentrenado. Estas se usarán para conocer la destreza a la hora de predecir las categorías semánticas del corpus entero, tanto del conjunto de entrenamiento como del de prueba, y comparar así los resultados que se arrojan de textos que el modelo sí que conoce y los textos que el modelo no ha visto previamente.

Como este modelo es un tanto especial, al poder etiquetar una entidad con varias anotaciones, haber solapamiento de etiquetas con otras entidades y otras características, no se han

encontrado métricas ya definidas para este tipo de casos. Es por ello por lo que se ha creado una métrica propia para poder medir el modelo. Esta nueva métrica creada se ha basado en la expuesta en el paper de Piad-Morffis y cols. (2020).

Este proceso de medición se ha realizado en dos pasos. Primero se mide la destreza del modelo en la tarea de detectar entidades. Una vez se obtiene esa métrica, se mide el modelo en la tarea de clasificar entidades en las diferentes categorías.

Para la detección de entidades, se diferenciarán cuatro casos:

- Correcta (C_d): Una detección será correcta cuando sobre en una frase se encuentra una anotación con el mismo inicio y final.
- Parcial (P_d): Una detección es parcial si en una frase se solapan las detecciones. Por ejemplo, que comiencen en la misma posición de inicio, pero finalicen en una posición final diferente. Uno de los casos en los que se puede producir es cuando el modelo nuevo predice una entidad en varias partes, mientras que en el *gold standard* es solo una.
- *Missing* (M_d): Una detección será clasificada como *missing* (falta), cuando se encuentra en el *gold standard* pero no en las detectadas por el nuevo modelo.
- *Spurious* (S_d): Este último caso sería la inversa del *missing*. Se da cuando el modelo ha detectado una entidad que no está en el *gold standard*.

Con estos cuatros elementos, se calcularán la precisión (*precision*), exhaustividad (*recall*), puntuación F1 (*F1 score*) y la exactitud (*accuracy*) del modelo. Para ello, se utilizan las siguientes fórmulas.

$$Precision_{deteccion} = \frac{C_d + 0.5 * P_d}{C_d + P_d + S_d}$$

$$Recall_{deteccion} = \frac{C_d + 0.5 * P_d}{C_d + P_d + M_d}$$

$$F1 - Score_{deteccion} = \frac{Precision_{deteccion} * Recall_{deteccion}}{Precision_{deteccion} + Recall_{deteccion}}$$

$$Accuracy_{deteccion} = \frac{C_d}{C_d + M_d + S_d}$$

Por otro lado, para la clasificación de entidades, se diferenciarán otros tres casos:

- Correcta (C_c): Una clasificación será correcta cuando en una frase se encuentra una categoría del *gold standard* en las anotaciones generadas por el modelo.
- *Missing* (M_c): Una clasificación será clasificada como *missing* (falta), cuando se encuentra en el *gold standard* pero no en las detectadas por el nuevo modelo.
- *Spurious* (S_c): Este último caso sería la inversa del *missing*. Se da cuando el modelo ha clasificado una entidad que no está en el *gold standard*.

Con estos tres elementos, se calcularán la precisión (*precision*), exhaustividad (*recall*), puntuación F1 (*F1 score*) y la exactitud (*accuracy*) del modelo. Para ello, se utilizan las siguientes fórmulas.

$$Precision_{clasificacion} = \frac{C_c}{C_c + S_c}$$

$$Recall_{clasificacion} = \frac{C_c}{C_c + M_c}$$

$$F1 - Score_{clasificacion} = \frac{Precision_{clasificacion} * Recall_{clasificacion}}{Precision_{clasificacion} + Recall_{clasificacion}}$$

$$Accuracy_{clasificacion} = \frac{C_c}{C_c + M_c + S_c}$$

Respecto al código, la función para realizar la detección de entidades recibe un único parámetro, *textTags*. Es un vector que contiene todas las anotaciones reales y las predichas por el modelo para todos los textos, así como el vector *sentences*, que indica las posiciones en las que se delimitan las frases del texto. Para cada texto, se cuenta con un vector de las anotaciones reales y otra para las anotaciones predichas.

Lo primero que realiza esta función es crear un *dataframe*, el cual contiene 8 columnas: C_d, P_d, M_d, S_d, $Precision$, $Recall$, $F1$ y $Accuracy$.

Después, se itera sobre todos los textos, para así crear una fila en el dataframe por cada texto. Por cada frase en el texto, se calculan las detecciones, anotando cada detección en uno de los 4 casos definidos (C_d, P_d, M_d y S_d). Importante volver a resaltar el hecho de que la ventana de posibilidades se ha limitado a una frase, para que así las mediciones realizadas sean más precisas. Aumentar esa ventana a todo el texto puede dar lugar a errores, como intentar buscar la anotación de una determinada frase en otra diferente.

Una vez se han recorrido todos los textos, se calculan para todas las filas del dataframe las métricas, dadas las fórmulas definidas anteriormente.

Respecto a la clasificación, el proceso seguido es similar al de clasificación, pero en este caso, en vez de mirar las posiciones inicio y final, se miran las etiquetas.

Dadas esas dos funciones definidas, para facilitar el proceso de obtención de métricas sobre el conjunto de entrenamiento y de test, se ha creado una función auxiliar, que recibe cuatro parámetros: el dataset, el modelo, el tokenizador, y el diccionario *tag2id*. La función crea el diccionario de anotaciones reales-predichas que necesitan las funciones de detección y clasificación, llama a esas dos funciones, y calcula las métricas globales para cada tarea. Una vez obtenido todo eso, se devuelven las métricas y los dataframes.

Con todo ello, obtener las métricas se realiza en una simple llamada a la última función definida. En el Código 4.6 se puede ver como se han obtenido las métricas para el conjunto de entrenamiento y para el conjunto de test.

Código 4.6: Obtención de las métricas de detección y clasificación

```
1 dfDetectionTrain, dfClassificationTrain, detectionMetricsTrain, classificationMetricsTrain = evalDataset(trainData,
      model, tokenizer, tag2id)
```

```
dfDetectionTest, dfClassificationTest, detectionMetricsTest, classificationMetricsTest = evalDataset(testData,
    model, tokenizer, tag2id)
```

Finaliza aquí el capítulo que explica cómo se ha realizado el proceso de reentrenamiento para cada uno de los modelos que se querían obtener. En la posterior sección, los resultados obtenidos serán expuestos.

5. Resultados y discursión

En este nuevo capítulo, y una vez teniendo desarrollado el nuevo modelo, se exponen los resultados de las métricas implementadas para medir la calidad del modelo. Además, se aporta una discusión final sobre los resultados obtenidos de los diferentes modelos.

5.1. Modelos entrenados y métricas obtenidas

Por cada corpus creado (detallados en la Sección 4.1.1), se ha realizado un reentrenamiento del modelo, para ver así como se comportaba. Todos los procesos se han realizado con los mismos parámetros, iterando 100 veces en cada caso, tal y como se ha explicado en la Sección 4.3. Todos los resultados que se mostrarán en las tablas sucesivas son valores entre el 0 y el 1.

Se detallan a continuación los resultados de los diferentes modelos.

5.1.1. Modelo 1

Este primer modelo se ha creado utilizando el corpus original (Conjunto de datos 1). Se detallan en la Tabla 5.1 los resultados obtenidos para el conjunto de entrenamiento y de test para las tareas de clasificación y detección. El conjunto de entrenamiento y test medidos son los utilizados para el reentrenamiento del modelo (es decir, el Conjunto de datos 1).

Métrica	Detección		Clasificación	
	Train	Test	Train	Test
Precision	0.68	0.58	0.54	0.43
Recall	0.74	0.65	0.48	0.40
F1-Score	0.70	0.60	0.56	0.48
Accuracy	0.63	0.47	0.34	0.25

Tabla 5.1: Resultados del primer modelo entrenado

Comenzando a debatirlos, nos fijamos en la tarea de detección. Aquí, según la precisión, el modelo es capaz de detectar en el 68% de las ocasiones correctamente las entidades en el caso del conjunto de entrenamiento, y en un 58% en el caso del conjunto de prueba. Se podría generalizar diciendo que la mitad de las veces detecta correctamente las entidades.

Respecto a la clasificación, los datos empeoran. Volviéndonos a fijar en la métrica de la precisión, en el caso del conjunto de entrenamiento, el modelo es capaz de clasificar correctamente en el 53% de los casos, valor que desciende hasta el 43% en el caso del conjunto de prueba.

Como se puede observar, los resultados son muy pobres. Esto era esperable, dado al gran desbalanceo que presentaba este corpus, tal y como se ha comentado en la Sección 4.1.1.

Además, había muchas categorías con apenas datos, por tanto, es difícil que el modelo aprenda de ellas. Pese a esto, podrían haber sido bastante peores, ya que haber conseguido que en el conjunto de prueba el 58% de los casos de detecte correctamente y que el 43% se clasifique a una entidad en una categoría concreta, es un resultado malo, pero no desastroso, ya que sabiendo el gran desbalanceado y falta de datos que hay, podría haber sido mucho peor.

Se prueba otra aproximación en el siguiente modelo.

5.1.2. Modelo 2

En este caso, se ha realizado otro reentrenamiento del modelo original, pero en este caso con el corpus procesado previamente para tener menos categorías y estar mejor balanceado, tal y como se ha explicado en la Sección 4.1.1.2.

Los datos sobre los que se han obtenido las métricas son los del Conjunto de datos 2, corpus utilizado para el reentrenamiento de este modelo.

En la Tabla 5.2 se pueden ver los resultados obtenidos.

Métrica	Detección		Clasificación	
	Train	Test	Train	Test
Precision	0.98	0.63	0.96	0.50
Recall	0.95	0.58	0.93	0.46
F1-Score	0.96	0.58	0.95	0.46
Accuracy	0.96	0.45	0.90	0.31

Tabla 5.2: Resultados del segundo modelo obtenidos sobre los datos del Conjunto de datos 2

Viendo esos datos, se comenzará discutiendo los resultados de la detección. Sobre el conjunto de entrenamiento, el modelo arroja resultados prácticamente perfectos, estando todos los valores cercanos al 1. En cuanto al conjunto de prueba, los resultados bajan, estando todas las métricas en torno al 0.5. Viendo la precisión, se puede decir que, aproximadamente, 6 de cada 10 etiquetas son detectadas correctamente.

Respecto a la clasificación, al igual que pasaba en la tarea de detección, los resultados sobre el conjunto de entrenamiento son muy buenos, pero sobre el conjunto de prueba el rendimiento baja. Fijándonos en la precisión, se puede decir que la mitad de las veces el modelo categoriza de forma correcta las etiquetas detectadas.

Sabiendo que los datos medidos en la tabla 5.1 y los de la Tabla 5.2 son diferentes (cada uno solo tiene datos para las categorías medidas), se puede hacer una valoración global, pudiendo ver que conforme se han reducido las categorías con menos etiquetas (y por tanto, las categorías que hay ahora tienen más etiquetas por cada una de ellas), se mejoran los resultados, tanto en la tarea de detección como en la de clasificación.

En la próxima sección sí se podrá hacer una comparación justa (con los mismos datos) sobre ambos modelos.

5.1.3. Comparación de resultados

Las tablas mostradas anteriormente tienen una interpretación aislada, sin poder ser comparadas entre sí al haber sido utilizados corpus diferentes para la obtención de las métricas.

Métrica	Modelo original				Modelo 1				Modelo 2			
	Detección		Clasificación		Detección		Clasificación		Detección		Clasificación	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Precision	0.31	0.28	0.01	0.01	0.68	0.58	0.54	0.43	0.78	0.71	0.03	0.02
Recall	0.49	0.53	0.01	0.00	0.74	0.65	0.48	0.40	0.48	0.48	0.01	0.01
F1-Score	0.37	0.36	0.03	0.01	0.70	0.60	0.56	0.48	0.59	0.56	0.06	0.12
Accuracy	0.21	0.20	0.00	0.00	0.63	0.47	0.34	0.25	0.43	0.41	0.00	0.00

Tabla 5.3: Resultados del baseline y modelos reentrenados sobre el Conjunto de datos 1

Es por ello que se crea esta sección. En ella, los dos modelos serán evaluados usando el mismo corpus, el corpus original (Conjunto de datos 1). Se recuerda que este corpus se encontraba fuertemente desbalanceado, con muchas categorías, teniendo algunas de ellas apenas datos.

Además, se usará como punto de partida (lo que en inglés se le llama *baseline*), los resultados que aporte el modelo original, sin haber sido sometido a un proceso de reaprendizaje.

Viendo los resultados de la Tabla 5.3, se comenzará prestando atención a los datos arrojados por el modelo base, resultados que son utilizados como *baseline*. En la tarea de detección, los resultados son pobres, teniendo un 38% de precisión en el conjunto de entrenamiento y un 28% en el caso del conjunto de prueba. Si bien, esos resultados no son cercanos a 0. Los que sí lo son, son los de la tarea de clasificación. Esto era previsible, ya que el modelo original no ha visto nunca ejemplos de las categorías que se quieren entrenar, por tanto, su tarea de clasificación sobre estas categorías es prácticamente nula.

Respecto al modelo 1, cabe recordar que los datos que se están midiendo son los mismos con los que se ha realizado en reentrenamiento. Tal y como se ha debatido en la Sección 5.1.1, los resultados que se aportan no podrían ser considerados como buenos, pero tampoco desastrosos, sabiendo que la calidad del corpus es bastante pobre. Estas medias aportadas se deben de analizar con cuidado, debido al desbalanceado de datos que hay entre clases. Si bien, se mejoran en todos los aspectos a los resultados *baseline*, por lo cual, ya se puede decir que se ha mejorado.

Por último, los resultados del modelo 2. Comenzando por ver los datos de la clasificación, estos son mejores que los del modelo 1, y por tanto, mejores que los del modelo original. Aquí cabe destacar la gran diferencia entre la precisión y la exactitud que se aprecia tanto en el conjunto de entrenamiento y en el de test, hecho que se podría achacar a que muchas de las clases del modelo no se conocen, por tanto, la cantidad de detecciones parciales aquí será mayor, hecho que hace que el valor de precisión aumente. También hará que disminuya la exactitud el creciente número de elementos que el modelo no ha detectado (clasificados como *missing*). Respecto a la clasificación, como cabría esperar debido a haber entrenado menos de la mitad de las clases, los resultados son fatídicos, estando todas las métricas muy cercanas al 0.

6. Conclusiones

Como conclusión a este Trabajo de Fin de Máster, se comienza hablando del corpus.

El primer objetivo marcado fue el de contar con 90 categorías a predecir, un número muy amplio. Sin embargo, el número de textos era limitado, siendo aún más limitado el número de textos revisados. Este hecho hace que el corpus sea de una calidad mejorable, tanto en cuanto a la calidad de las anotaciones como a la cantidad y variedad de las mismas.

Pese a esta limitación, se ha trabajado para poder refinar toda la información que se tenía, y así poder utilizarla al máximo posible.

Es por esto por lo que se rebalanceó el corpus. Aquí cabe resaltar que ahora se cuentan con una amplia variedad de funciones que hacen posible que este proceso se realice de forma ágil y sencilla, pudiendo obtener nuevos corpus con una serie de parámetros a configurar, como el número de etiquetas por clase que se quieren, o el mínimo de etiquetas que debe de tener una clase para ser considerada.

Respecto al modelo, al principió se comentó que la tarea que debía de realizar no era tan común como otras en el ámbito del PLN. El modelo objetivo debía de poder detectar entidades y categorizarlas en una o varias categorías, pudiendo estas estar solapadas. Tras conocer los requisitos, se ha adaptado con satisfacción un modelo previamente BERT para que sea capaz de realizar esta tarea. Se ha ajustado sus parámetros, además de haber creado funciones que permiten tomar como entrada datos en formato BRAT, y que se obtengan como salida también estos datos, creando funciones que transforman este formato al formato IOB, el cual el modelo sí que entiende.

En cuanto a las métricas, se ha comprobado que en la actualidad no existían métricas ya establecidas para poder medir los resultados de un tipo de modelo como el deseado. Es por ello por lo que se han desarrollado nuevas, basándose en algunas definidas para tareas similares. Por ello, ahora se cuentan con funciones ya definidas que miden los datos para un modelo dado y obtienen valores para parámetros de medición comunes, como pueda ser el *Accuracy* o el *F1-Score*.

Por último, cabe destacar que los resultados de los modelos no son muy alentadores, si bien, son un muy buen punto de partida de cara a continuar las investigaciones, ya que han detectado los puntos débiles del reentrenamiento, por lo que estos se podrían solucionar en un futuro para obtener mejores modelos. Todos estos problemas detectados, juntos con propuestas de soluciones, son trabajo futuro, el cual se detalla a continuación.

6.1. Trabajo futuro

Vistas las conclusiones, se pueden definir alguna serie de aspectos que serían de interés de cara a mejorar los resultados que el presente trabajo aporta. Estos serían de utilidad y servirían como guía de cara a futuras investigaciones.

- Ajuste del corpus: Como se comentó al principio del trabajo, tener un corpus de buena calidad es vital para poder tener buenos modelos. Tal y como se ha discutido anteriormente, el corpus con el que se ha desarrollado este trabajo era limitado, teniendo además pocas anotaciones revisadas. Por tanto, como trabajo futuro, se podría por un lado aumentar el corpus, por ejemplo, escogiendo artículos enteros, y no solo sus resúmenes. Y por otro, se deberían de contar con muchas más anotaciones revisadas, para estar seguros de que estas son de una calidad lo mejor posible.

- Categorías a detectar: El objetivo inicial era la detección de 90 categorías. Esto es un número excesivamente grande. Por ello, como trabajo futuro, se debería de explorar la posibilidad de reducir el número de categorías a anotar, viendo cuáles son realmente de interés o que puedan ser más significativas. Si la voluntad es mantener las 90, se debería entonces de contar con un corpus tremendamente extenso, para ser capaz de tener unos datos balanceados, donde al realizar el reentrenamiento, el modelo pueda ver un número de diferentes anotaciones de cada categoría suficiente para aprenderlas bien y poder posteriormente predecirlas de la forma lo más correcta posible.

- Balanceado del corpus: Como se ha mostrado, se han creado nuevas funciones que permiten obtener corpus más balanceados que el original. Al describir el proceso, se vio que no era una tarea trivial. Como trabajo futuro, se podrían explorar otras vías de realizar el balanceado de corpus, investigando como minimizar el impacto que tiene eliminar una categoría en una frase sobre el resto de categorías contenidas en esa frase.

- Cambios en el reentrenamiento: En este TFM, el enfoque dado para conseguir un modelo mejor ha sido el de estructurar mejor el corpus. En el futuro, se podrían probar diferentes aspectos a la hora de realizar el reentrenamiento, como probar diferentes hiperparámetros, probar con otros modelos base o con otras arquitecturas.

- Revisión de las métricas: En el futuro se podrían revisar las métricas implementadas, buscando más intensamente si hay proyectos similares en los que se hayan desarrollado o usado métricas. Además, se pueden explorar otras vías de medir cómo el modelo va aprendiendo en cada iteración del reentrenamiento, ya que por el momento, solo se mide usando el *F1-Score*.

Bibliografía

Asale, R. (s.f.). *inteligencia | Diccionario de la lengua española.* Descargado de https://dle.rae.es/inteligencia?m=form#2DxmhCT

Devlin, J., Chang, M.-W., Lee, K., y Toutanova, K. (2019). *Bert: Pre-training of deep bidirectional transformers for language understanding.*

Morales, M., Morales, M., y Morales, M. (2022, 12). La palabra del año son dos: inteligencia artificial según la FundéuRAE. Descargado de https://elpais.com/cultura/2022-12-29/la-palabra-del-ano-son-dos-inteligencia-artificial-segun-la-fundeurae.html

Piad-Morffis, A., Gutiérrez, Y., Almeida-Cruz, Y., y Muñoz, R. (2020). A computational ecosystem to support ehealth knowledge discovery technologies in spanish. *Journal of Biomedical Informatics*, *109*, 103517. Descargado de https://www.sciencedirect.com/science/article/pii/S1532046420301453 doi: https://doi.org/10.1016/j.jbi.2020.103517

Posada, M., MartÃn Arribas, C., RamÃrez, A., Villaverde, A., y Abaitua, I. (2008, 00). Enfermedades raras: Concepto, epidemiologÃa y situaciÃactual en EspaÃ. *Anales del Sistema Sanitario de Navarra*, *31*, 9 - 20. Descargado de http://scielo.isciii.es/scielo.php?script=sci_arttext&pid=S1137-66272008000400002&nrm=iso

Tinn, R., Cheng, H., Gu, Y., Usuyama, N., Liu, X., Naumann, T., … Poon, H. (2021). *Fine-tuning large neural language models for biomedical natural language processing.* arXiv. Descargado de https://arxiv.org/abs/2112.07869 doi: 10.48550/ARXIV.2112.07869

UNESCO. (2021). Recomendación sobre la Ética de la Inteligencia Artificial .. Descargado de https://unesdoc.unesco.org/ark:/48223/pf0000380455_spa

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is all you need. *CoRR*, *abs/1706.03762*. Descargado de http://arxiv.org/abs/1706.03762

Lista de Acrónimos y Abreviaturas

BERT	*Bidirectional Encoder Representations from Transformers.*
BRAT	*Brat Rapid Annotation Tool.*
CNN	*Convolutional neural network.*
CRF	*Conditional Random Field.*
GPT	*Generative Pre-trained Transformer.*
GPU	*Graphics Processing Unit.*
IA	Inteligencia Artificial.
IOB	*Inside–Outside–Beginning.*
JSON	*JavaScript Object Notation.*
LSTM	*Long Short-Term Memory.*
NER	*Named Entity Recognition.*
NLP	*Natural Language Processing.*
PLN	Procesamiento del Lenguaje Natural.
POS	*Part-Of-Speech.*
RNN	*Recurrent Neural Networks.*
UMLS	Unified Medical Language System.
WSD	*Word Sense Desambiguation.*

A. Anexo I: Ejemplo de anotación

En este anexo se mostrará un ejemplo de un fichero de texto y de un fichero de anotaciones en BRAT.

En el Código A.1 se muestra el contenido de uno de los archivos de texto (concretamente, el archivo *text0.txt*).

Código A.1: Contenido del archivo *text0.txt*

```
Huntington's disease is a neurodegenerative autosomal disease results due to
    expansion of polymorphic CAG repeats in the huntingtin gene. Phosphorylation
    of the translation initiation factor 4E-BP results in the alteration of the
    translation control leading to unwanted protein synthesis and neuronal
    function. Consequences of mutant huntington (mhtt) gene transcription are not
    well known. Variability of age of onset is an important factor of Huntington's
     disease separating adult and juvenile types. The factors which are taken into
     account are-genetic modifiers, maternal protection i.e excessive paternal
    transmission, superior ageing genes and environmental threshold. A major focus
     has been given to the molecular pathogenesis which includes-motor disturbance
    , cognitive disturbance and neuropsychiatric disturbance. The diagnosis part
    has also been taken care of. This includes genetic testing and both primary
    and secondary symptoms. The present review also focuses on the genetics and
    pathology of Huntington's disease.
```

Ahora, en el Código A.2 se muestran las diez primeras anotaciones del texto anterior, presentes en el archivo de anotaciones de ese texto (archivo *text0.ann*).

Código A.2: Contenido del archivo *text0.ann*

```
T1 DiseaseOrSyndrome 0 20 Huntington's disease
T2 FunctionalConcept 0 20 Huntington's disease
T3 AminoAcidPeptideOrProtein 0 20 Huntington's disease
T4 DiseaseOrSyndrome 26 61 neurodegenerative autosomal disease
T5 LaboratoryOrTestResult 62 69 results
T6 IntellectualProduct 62 69 results
T7 ResearchActivity 62 69 results
T8 Classification 62 69 results
T9 Finding 62 69 results
T10 CellFunction 77 86 expansion
```

Todas estas anotaciones se encuentran en formato BRAT. Se detallan a continuación los diferentes elementos que componen la anotación:

- Número de elemento: Se otorga en cada línea un número de elemento secuencial, precedido de una ”T”.

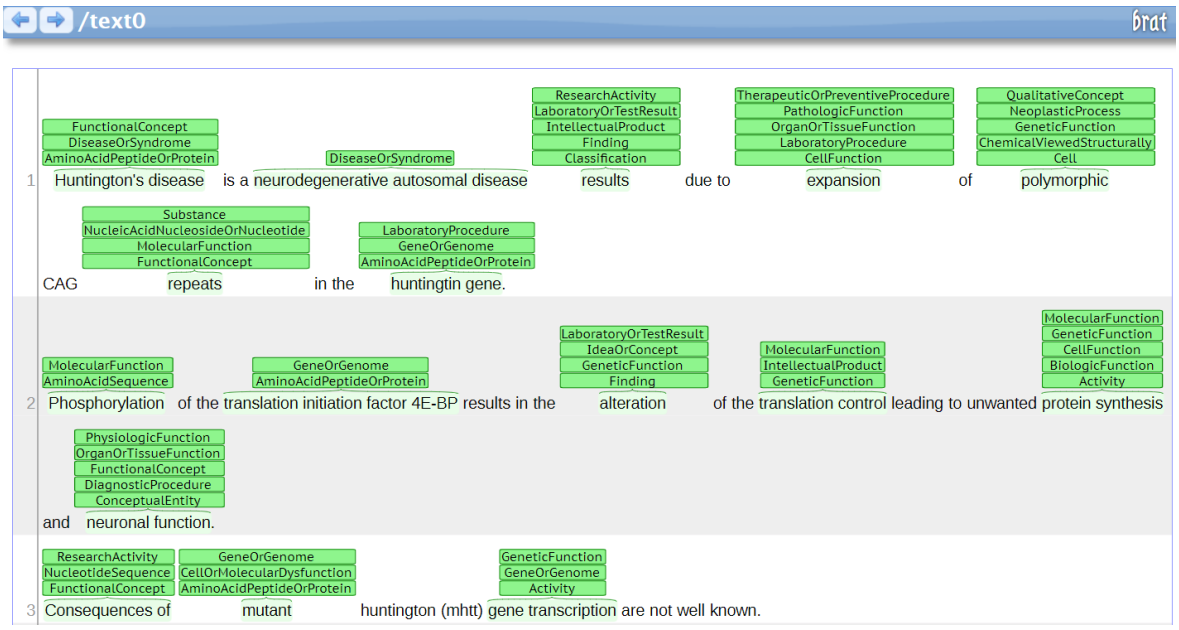

Figura A.1: Representación del texto y anotaciones en el servidor BRAT

- Categoría semántica: Se muestra la categoría semántica (sin espacios en blanco) a la que pertenece el texto anotado.
- Posición inicial: Posición de inicio del texto anotado.
- Posición final: Posición de fin del texto anotado.
- Texto anotado: Texto que se ha anotado.

Todos estos elementos se encuentran separados los unos de los otros mediante el carácter de espacio o el carácter TAB.

Para ver visualmente la representación de las anotaciones, se puede iniciar el servidor de BRAT. Se ejecuta este servidor en local, accediendo a él a través del navegador. Este servidor detecta pares de archivos, es decir, busca el archivo con el texto más el archivo con las anotaciones.

En la Figura A.1 se muestra la representación del texto y anotaciones anteriormente mostrados. Para mejorar la visualización, se muestran solo las tres primeras frases.

Se puede ver como las etiquetas se superponen unas con otras, habiendo algunas que son relativas a una sola palabra, mientras otras se aplican a grupos de palabras.

Para hacer esta representación posible en el servidor de BRAT, se han realizado modificaciones en él, concretamente en el archivo *annotation.conf*. En él, primero se han incluido todas las categorías semánticas con las que se cuenta. Además, se han especificado todos los grupos de anotaciones. Es decir, por cada par de posibles anotaciones (por ejemplo, *FunctionalConcept* y *DiseaseOrSyndrome*), se ha creado una línea de configuración, para así especificarle al sistema que todos los solapamientos de categorías son posibles.

B. Anexo II: Categorías semánticas

Este segundo anexo está dedicado a explicar las categorías semánticas de UMLS que se han utilizado en este proyecto.

Originalmente, UMLS cuenta con 127 categorías semánticas. A continuación se listarán todas ellas, mostrando en la columna de la izquierda las que se han escogido y en la de la derecha las que se han descartado para este trabajo.

Categorías escogidas	**Categorías descartadas**
Amino Acid, Peptide, or Protein	Activity
Acquired Abnormality	Amphibian
Age Group	Animal
Amino Acid Sequence	Behavior
Anatomical Abnormality	Bird
Anatomical Structure	Biomedical or Dental Material
Antibiotic	Chemical
Archaeon	Classification
Biologically Active Substance	Conceptual Entity
Bacterium	Daily or Recreational Activity
Body Substance	Education Activity
Body System	Entity
Biologic Function	Event
Body Location or Region	Fully Formed Anatomical Structure
Biomedical Occupation or Discipline	Fish
Body Part, Organ, or Organ Component	Food
Body Space or Junction	Geographic Area
Cell Component	Governmental or Regulatory Activity
Cell Function	Group Attribute
Cell	Group
Congenital Abnormality	Human-caused Phenomenon or Process
Chemical Viewed Functionally	Human
Chemical Viewed Structurally	Idea or Concept
Clinical Attribute	Language
Clinical Drug	Manufactured Object
Cell or Molecular Dysfunction	Occupational Activity
Carbohydrate Sequence	Organization
Diagnostic Procedure	Plant
Drug Delivery Device	Professional or Occupational Group
Disease or Syndrome	Professional Society
Environmental Effect of Humans	Qualitative Concept
Element, Ion, or Isotope	Quantitative Concept

Categorías escogidas	**Categorías descartadas**
Experimental Model of Disease	Reptile
Embryonic Structure	Regulation or Law
Enzyme	Self-help or Relief Organization
Eukaryote	Social Behavior
Family Group	Vertebrate
Finding	
Fungus	
Functional Concept	
Genetic Function	
Gene or Genome	
Health Care Related Organization	
Health Care Activity	
Hazardous or Poisonous Substance	
Hormone	
Immunologic Factor	
Individual Behavior	
Inorganic Chemical	
Injury or Poisoning	
Intellectual Product	
Indicator, Reagent, or Diagnostic Aid	
Laboratory Procedure	
Laboratory or Test Result	
Mammal	
Molecular Biology Research Technique	
Machine Activity	
Medical Device	
Mental Process	
Mental or Behavioral Dysfunction	
Molecular Function	
Molecular Sequence	
Neoplastic Process	
Nucleic Acid, Nucleoside, or Nucleotide	
Natural Phenomenon or Process	
Nucleotide Sequence	
Occupation or Discipline	
Organic Chemical	
Organism Attribute	
Organism Function	
Organism	
Organ or Tissue Function	
Pathologic Function	
Physical Object	
Phenomenon or Process	
Physiologic Function	

Categorías escogidas	**Categorías descartadas**
Pharmacologic Substance	
Patient or Disabled Group	
Population Group	
Receptor	
Research Activity	
Research Device	
Substance	
Sign or Symptom	
Spatial Concept	
Tissue	
Temporal Concept	
Therapeutic or Preventive Procedure	
Virus	
Vitamin	