Trabajo Fin de Máster

Análisis de textos médicos mediante NLP

Autor:
Jesús Enrique Cartas Rascón

Tutora:
Rocío Romero Zaliz

Resumen

En el ámbito de la medicina se almacena una gran cantidad de información relevante: desde valores numéricos correspondientes a signos vitales hasta texto plano que realiza un especialista para completar un informe. Muchas veces los datos guardados en el historial médico de un paciente, que no tiene una estructura determinada, son ignorados. Este proyecto propone recuperar texto médico sin formato para extraer nuevo conociemiento que pueda utilizarse para complementar la información estructurada y mejorar en la clasificación y tratamiento de los pacientes.

Índice general

1. Introducción

En este capítulo introduciremos los principales problemas existentes en el contexto de minería de datos en el texto médico y pronpodremos una solución que se desarrollará a lo largo del documento.

1.1. Documentos médicos

Toda atención médica dispone de documentos que recogen toda la información relacionada con el paciente, enfermedad, y un seguimiento de ambos, así como los recursos a utilizar. Los documentos se suelen organizar en un formato de campo-valor, que relaciona los diferentes datos a guardar con su valor esperado. Por ejemplo, nombre, apellidos, o referencias de códigos que se utilicen, como medicamentos o tratamientos.

En estos documentos suele haber una sección en la que el o la profesional en cuestión describe el estado del paciente, así como otros matices que no estuvieran considerados en los campos anteriores. Precisamente por esta razón existe dicha sección en el documento.

La medicina personalizada trata de acercar los tratamientos al paciente lo máximo posible, de forma que los profesionales sanitarios puedan hacer un seguimiento en profundidad de los pacientes sin tener que ello conllevar una enorme carga cognitiva de recordar cada paso en el tratamiento. Esto, entre otras cosas, involucra bases de datos en las que guardar toda esta información, así como el análisis de la taxonomía de dichos documentos. Esto facilita su posterior búsqueda para poder retomar el punto en el que se acabó anteriormente.

Para facilitar esta tarea, se creó un una colección de términos fácilmente procesables por un ordenador, el SNOMED (Systemized Nomenclature of Medicine Clinical Terms). Este conjunto de términos fácilmente indexables ayudan en la inclusión de la informática y el procesamiento automático de los documentos médicos.

El objetivo es centrar nuestra atención en esas secciones de texto sin formato anteriormente mencionadas, con objeto de obtener la mayor cantidad de información posible y anexarla, ahora con formato, al documento del que provienen, enriqueciendo el informe y habilitando nuevas claves de búsqueda, así como mejorando el indexado de los documentos.

1.2. Datos delicados y escasos

Uno de los principales problemas a los que nos enfrentamos es la falta de *datasets* o conjuntos de datos en los que estos documentos estén presentes.

Gran parte de este problema es la delicada naturaleza de estos datos. No estamos hablando de el ancho y el alto de los pétalos de una flor, o de las acciones en bolsa de una determinada empresa. Hablamos de datos profundamente íntimos de personas reales con problemas reales. Por ley, concretamente la Ley Orgánica de Protección de Datos (LOPD), las entidades deben proteger dichos datos y garantizar la privacidad de las personas involucradas.

Una de las posibles soluciones a esto es la anonimización de los datos. Puede parecer una tarea simple pero es muy costosa, proporcionalmente costosa al número de datos que poseamos. Debemos no solo garantizar la anonimidad de los pacientes, sino también la de todos los profesionales involucrados. Esto es información que no aporta nada al modelo, en cualquier caso.

Afortunadamente, ya se ha trabajado en esto y existen bases de datos públicas, precisamente para esto. Uno de nuestros principales objetivos es el de buscar y agregar tantas fuentes de datos como sea posible, unificarlas y crear una herramienta que aproveche todos los datos disponibles públicamente para generar datos nuevos sintéticos, pero suficientemente convincentes.

De ser nuestro proyecto exitoso, podríamos obtener ingentes cantidades de información estructurada de la secciones de texto mencionadas anteriormente, lo que posibilitaría una nueva dimensión en el tratamiento de datos médico y habilitaría a tratamientos mucho más precisos y cercanos al paciente.

1.3. Objetivos

Dado este marco, describiremos en esta sección los objetivos de nuestro trabajo.

- En primer lugar, se agregarán todas las fuentes de información públicas que nos provean con datos de comentarios médicos listos para su minería y análisis.
- Utilizando todos estos datos, se hará una evaluación de las herramientas que ya existen en el estado del arte. Haremos una revisión de cómo se utilizan y del rendimiento de dichas herramientas. Sin embargo, para evaluar dichas herramientas,

no utilizaremos los datos encontrados, sino que efectuaremos un flujo de trabajo alternativo.

- Utilizando técnicas de aprendizaje automático y generativo, crearemos un modelo que sea capaz de generar tantos comentarios médicos como sea necesario. La idea es suplir la carencia de datos con un modelo generativo, de forma que no se tenga que lidiar con aspectos de privacidad o licencia, ya que todos los comentarios serían generados de forma sintética.

Si bien los comentarios son sintéticos, deben ser lo suficientemente convincentes como para que la evaluación de las herramientas sea fiel y rigurosa. Esto ofrece una herramienta esencial para los desarrolladores de los sistemas que habilita a un mejor y más fructífero desarrollo, ya que se disponde de una cantidad, *idealmente infinita* de comentarios sobre los que testear el modelo.

2. Fundamentos de la minería de texto

En este capítulo discutiremos algunas de las técnicas y términos más importantes a la hora de hablar de minería de texto, así como minería de datos en general, con objeto de que todas las consideraciones realizadas posteriormente queden claras.

En *Text Mining Applied to Electronic Medical Records: A Literature Review* [1] se hace una revisión de los diferentes aspectos a tener en cuenta durante el procesamiento de textos médicos. Nos apoyaremos en gran medida en la estructura, contenidos y referencias de este artículo, que resume muy bien todo lo que necesitamos saber para resolver nuestro problema.

2.1. Minería de datos

La minería de datos es una rama de la informática que se dedica a encontrar tendencias y patrones en grandes volúmenes de información. Estas tendencias y patrones crean *conocimiento* a partir de los datos, es decir: información estructurada desde los datos no estructurados. Esta información es muy valiosa y contribuye en las decisiones que se vayan a tomar o a monitorizar algunos aspectos que sean de vital importancia para el interesado.

La minería de datos puede dividirse en un número de técnicas que funcionan de forma diferente en función del tipo de datos que tengamos y la información que busquemos.

1. **Asociación**: esta técnica se centra en encontrar relaciones entre las distintas variables de nuestros datos, con objeto de encontrar muestras que sean estadísticamente dependientes. Una de las técnicas más utilizadas son las reglas de asociación, cuya salida tras el cálculo son un conjunto de reglas con antecedentes y consecuentes, muy fácilmente interpretables por cualquier persona, familiarizada o no con la ciencia de datos. [2]

2. **Clasificación**: el proceso de clasificación trata de asignar una categoría a un conjunto de elementos que tengan algún aspecto en común. La clasificación en la minería

de datos es una de las técnicas más utilizadas, ya que la naturaleza de gran parte de los datos responden bien a este método. [3]

3. **Agrupamiento**: también denominado *clustering* trata de agrupar muestras que tengan características similares. A diferencia de la clasificación, aquí no tenemos una etiqueta o categoría a la que asignar las muestras, sino que las agrupamos *a ciegas*, simplemente basándonos en alguna métrica para evaluar la distancia que haya entre un determinado par de muestras. [4]

4. **Predicción**: la predicción nos ayuda a encontrar tendencias entre variables, generalmente en datos con una componente temporal fuerte. [5] Es común poder predecir si un paciente sufrirá una determinada enfermedad conociendo su historial médico, por ejemplo.

5. **Identificación de patrones secuenciales**: Al igual que la predicción, se trabaja sobre datos con una componente temporal marcada. En este caso, se buscan patrones, es decir, conjuntos o cadenas de muestras que aparecen de forma frecuente en un orden concreto.

2.2. Minería de texto

En esta sección, discutiremos los diferentes aspectos a tener en cuenta en la minería de textos en concreto, tras haber abordado el concepto de minería de datos en un ámbito más general.

2.2.1. Términos

Definiremos algunos de los términos más utilizados en esta disciplina, guiándonos principalmente por el trabajo de Kamran Kowsari, *Text Classification Algorithms: A Survey* [6].

Tokens

El término más esencial en minería de textos es *token*. Un token es la mínima unidad en la que dividiremos un cuerpo de texto a la hora de analizarlo. Este elemento suele corresponderse con una palabra, que en el contexto de la mayoría de los idiomas corresponde con un conjunto de letras separado por espacios anterior y posteriormente. Esto da lugar a la creación de *Tokenizers*, algoritmos que toman un cuerpo de texto como una

cadena de caracteres muy larga, y devuelven un vector de palabras. Estos *tokenizers* no han de tomar el espacio en blanco necesariamente ni exclusivamente como criterio divisor, aunque suele ser lo más común. Algunos de los *tokenizers* más famosos son:

- **Tokenizers de palabras**
  - **Standard Tokenizer**: El Standard Tokenizer divide el texto en términos siguiendo los límites de las palabras según están definidos en el algoritmo *Unicode Text Segmentation*. Funciona bien en general.
  - **Letter Tokenizer**: divide el texto en términos cada vez que encuentra un carácter que no es una letra.
  - **Whitespace Tokenizer**: Toma como criterio divisor el espacio en blanco.
  - **Language Tokenizer**: Otros tipos de tokenizers adaptados a diferentes idiomas, como el inglés, que es el idioma más estudiado con diferencia, pero también otros idiomas con caracteres y reglas diferentes a aquellos basados en reglas occidentales, como el tailandés, o el chino.
- **Tokenizers de palabras parciales**
  - **N-Gram Tokenizer**: Este tokenizador incluye un parámetro adicional. Primero divide el texto con alguna de las reglas mencionadas anteriormente, y posteriormente, divide cada término del vector resultante en una ventana deslizante de n elementos, de ahí *N-Gram.* Por ejemplo: *quick fox* devolvería [qu, ui, ic, ck], [fo, ox], dado un $n = 2$. Estos tokenizers también pueden utilizarse a nivel de párrafo, por lo que se devolverían pares de palabras, algo que puede ser muy útil para el análisis de *dichos* o expresiones.
- **Tokenizers de texto estructurado**
  - **Pattern Tokenizer**: este tokenizer utiliza el patrón provisto como parámetro para la división de texto, utilizando expresiones regulares.
  - **Simple Pattern Tokenizer**: este tokenizer utiliza el patrón provisto como parámetro para la división de texto, utilizando expresiones optimizadas para el patrón dado, lo que hace que funcione generalmente más rápido pero también será más específico.

En resumen, un tokenizer es un algoritmo que divide el texto provisto siguiendo los criterios definidos por el usuario, devolviendo un vector con los elementos del texto divididos atendiendo a dichos criterios. Es una de las herramientas más esenciales en la minería de texto, ya que permite generar la mínima unidad de información a partir de la que se extraerá conocimiento.

Palabras vacías

Las palabras vacías o *stopwords* son términos presentes en un idioma que sirven de apoyo para la formulación de oraciones pero que no poseen información en sí. Nos referimos a los artículos, determinantes, preposiciones, etc.

Estos términos son considerados como *ruido* en el procesamiento de texto, por lo que lo más usual es disponer de un diccionario de términos vacíos y filtrar el texto original, eliminando dichos términos. De esta forma, nos quedamos con las palabras más importantes. Los símbolos de puntuación también se suelen considerar como ruido; si bien son esenciales para la comprensión y estructuración de texto para los humanos, suponen un detrimento para algoritmos de clasificación.

Sin embargo, esta operación es delicada y no siempre ofrecerá buenos resultados. Por ejemplo, si tratamos de inferir la intención de la oración *No me gusta el fútbol* y pasamos previamente un filtro de palabras vacías, el texto resultante sería *gusta fútbol*. Dados estos términos, se infiere que se está opinando de forma positiva acerca del tema *fútbol*, cuando no es así.

Este caso particular está descrito en la literatura como *negation handling*, en trabajos como [7] o [8]. Aún así, hay muchos factores que se deben tener en cuenta antes de eliminar términos de una oración.

Stemming y Lematización

Stemming hace referencia a la gestión de palabras con prefijos o sufijos para su integración en una frase, como plurales (casa, casa**s**). Se trata de eliminar los posibles complementos añadidos con objeto de normalizar las palabras y que todas tengan la misma forma. En este caso también han de tenerse en cuenta las negaciones (típico, **a**típico).

La lematización va un paso más allá y trata de encontrar la raíz de las palabras, obteniendo una normalización más estricta. Un buen ejemplo son la conjugación de los verbos: de *estudiando*, *estudiante* o *estudio* obtenemos *estudi-*. [9]

Frecuencias: TF, IDF

Uno de los datos más importantes a obtener de un texto es la frecuencia de palabras. Esta operación es tan simple como suena: contar cuántas veces aparece cada palabra y anotarlo en una estructura similar a un diccionario. Este término se conoce como *Term*

Frequency o TF. Estos valores suelen representarse en una escala logarítmica, con objeto de que las palabras muy dominantes no eclipsen a las menos frecuentes.

Del campo de teoría de la información [10] conocemos que aquellos términos que aparezcan con una frecuencia muy alta poseerán menos información que aquellos que aparezcan menos. Como vimos en la sección 2.2.1, eliminamos las palabras vacías porque aparecían mucho. Es decir, un artículo como *el* o una preposición como *de* tendrían una frecuencia desproporcionada, cuando en realidad no aportan ninguna información.

De forma similar, el valor *Inverse Document Frequency* [11] trata de abarcar esta frecuencia pero en un conjunto de documentos, añadiendo la inversa de la frecuencia por documento. Esta métrica se utiliza mucho en conjunción con la TF, resultando en la TF-IDF, que trata de medir la relevancia de un término en un conjunto de documentos. Esto resulta en un cálculo tal que así:

$$W(d,t) = TF(d,t) * log(\frac{N}{df(t)}) \tag{2.1}$$

donde d es un documento del conjunto de documentos con cardinalidad N, t es el término en concreto y $df(t)$ es el número de documentos que contienen el término t.

Bolsas de palabras

Conociendo el concepto de frecuencias de palabaras, una de las aplicaciones directas son las bolsas de palabras, que recogen en una estructura con forma de diccionario cada término y su frecuencia.

De esta forma, tenemos un *ranking* para cada término. Esto se utiliza extensivamente en sistemas de recomendación, en donde una consulta provista por un usuario se compara con la bolsa de palabras del posible conjunto de documentos, y este conjunto va afinándose conforme se van comparando los conjuntos de palabras. El resultado es una búsqueda más refinada que devuelve documentos más relevantes con respecto a la consulta realizada.

2.2.2. Técnicas

En esta sección discutiremos algunas de las técnicas avanzadas más utilizadas en procesamiento y análisis de texto que además utilizaremos en nuestra implementación directa o indirectamente.

Word Embeddings

Esta técnica esencialmente trata de convertir los diferentes términos en vectores de números reales, ya que esto los convierte en objetos matemáticos fáciles de comparar y procesar. Matemáticamente hablando corresponde con una representación de un espacio n-dimensional a un espacio vectorial continuo de menor tamaño, donde n es el número total de términos presentes en todos los documentos.

Esta técnica ha sido estudiada en profundidad en varios proyectos:

- **Word2Vec**: esta técnica trata de representar las palabras como vectores utilizando una red neuronal con dos capas, haciendo uso de una bolsa de palabras continua (CBOW) y el modelo del Skip Gram. [12]
- **GloVe**: acrónimo de *Global Vectors for Word Representation*, es una técnica muy similar a la *Word2Vec*, con la particularidad de estar preentrenada en grandes corpus de texto, basados en Wikipedia y Gigaword. [13]
- **FastText**: es una técnica desarrollada por Facebook. Esta técnica hace uso de la técnica de los n-gramas para su entrenamiento, obteniendo una representación de los términos mucho más granular. [14]
- **Contextualized Word Representations**: esta técnica hace uso del contexto de las palabras para tratar de encontrar una representación y relación entre ellas. Esta técnica basa su funcionamiento en el uso de Long-Short Term Memory, un tipo de red neuronal recurrente muy utilizada en procesamiento de texto, en la que ahondaremos más en profundidad en secciones posteriores de este documento. [15]

Reducción de dimensionalidad

La reducción de dimensionalidad es una técnica que permite proyectar el espacio en el que se hallan nuestros datos en un subespacio de menor dimensionalidad, con objeto de facilitar el cálculo de las propiedades de dichos datos sin tener que utilizar todas sus características. Es común encontrar conjuntos de datos con un número de dimensiones muy alto, que hace inviable su estudio.

Las principales técnicas desarrolladas para reducir la dimensionalidad incluyen:

- **Principal Component Analysis (PCA)**: PCA o análisis de componentes principales trata de encontrar un subespacio latente que represente a los datos encontrando aquellas variables que estén menos relacionadas y que maximicen la varianza, para conservar la mayor cantidad de variabilidad posible. [16]

- **Independent Component Analysis (ICA)**: es una técnica similar que trata de expresar los datos con transformaciones lineales. [17]
- **Linear Discriminant Analysis (LDA)**: es otro método muy utilizado cuando los datos son de caracter categórico y no tienen una proporcion uniforme intraclase. [18]

Toda esta familia de algoritmos resulta muy conveniente para *comprimir* datos de alta dimensionalidad y extraer solo las **características principales** de los mismos. Se suelen usar en etapas de preprocesamiento, donde los datos resultantes se pasan a los algoritmos a entrenar, consiguiendo un mejor resultado y rendimiento en comparación con los datos sin preprocesar.

Clasificación de texto

Por último, abordaremos las principales técnicas para clasficar texto, ámbito importante en nuestro proyecto, así como una breve explicación de las mismas. Al igual que en las secciones anteriores, no es nuestro objetivo estudiarlas en profundida, pero más bien ofrecer una vista general del panorama en cuanto a esta tecnología con objeto de que el lector se familiarice con los términos.

Los principales algoritmos de clasificación de texto, entre otros, son los siguientes:

- **Boosting y bagging**: boosting y bagging son dos algoritmos basados en lo que se denomina en la literatura como *ensemble learning*. Esta técnica utiliza un gran número de modelos que funcionan muy bien para casos muy específicos pero no generalizan correctamente. La idea es que la respuesta conjunta de todos los modelos nos acerque a la respuesta correcta. Esta decisión puede hacerse mediante votos u otros métodos. [19]
- **Regresión logística**: la regresión logística es uno de los métodos de aprendizaje más simples, junto con la regresión linear. La regresión logística es una especialización de la regresión lineal, de forma que se utiliza una función logística para predecir categorías discretas, no continuas. [20]
- **Redes neuronales recurrentes**: por último, las redes neuronales recurrentes son una especialización de las redes neuronales en las que un subonjunto de las neuronas reciben su salida como una entrada, generándose ciclos de retroalimentación o *feedback*. Estas redes funcionan especialmente bien con datos con patrones y componentes temporales, características especialmente destacables del lenguaje humano, así como de la música o vídeo. [21]

Existen otras muchas técnicas en clasificación de texto, como *K Nearest Neighbours* [22], *Naïve Bayes* [23], *Support Vector Machines (SVM)* [24], árboles de decisión [25] o *Random Forests* [26], entre muchas otras.

2.3. Estado del arte

Por último, mencionaremos brevemente los tres proyectos más grandes de lenguaje generativo hasta la fecha, con objeto de entender qué es lo que hacen para poder integrar dichas técnicas en nuestro modelo.

2.3.1. Transformers

En esta sección hablaremos del Transformer en más profundidad, sus características principales en contraste con las demás tecnologías y de la configuración escogida para nuestro problema.

Arquitectura y módulos de atención

El Transformer es un tipo de arquitectura de red neuronal creada por ingenieros de Google [27]. Se la presenta como *estado del arte en procesamiento del lenguaje natural*, constituyendo un modelo mucho más potente y **considerado** que las anteriores LSTM.

El transformer está princpalmente basado en redes recurrentes, redes cuya entrada está conectada a la salida y, generalmente, se las construye con un contexto temporal en mente. Esto quiere decir que dada una entrada en el momento n, predeciremos la salida en función de la salida que se obtuvo en el momento $n-1$, además de la entrada del momento n en sí.

Esto es en sí el funcionamiento de una red recurrente estándar. El transformer añade varios conceptos clave a su implementación que lo hacen particularmente poderoso.

Atención

Uno de los principales conceptos que añade el transformer es de la **atención**. Este concepto trata de emular el concepto de atención con el que todos estamos familiarizados: el de ponderar y distrubir los recursos cognitivos de forma que aquellos estímulos más importantes reciban más recursos de cómputo por parte del sistema, de la misma forma

que nuestro cerebro procesa de forma más potente aquello en lo que estamos concentrados, ignorando aquellos estímulos que sean menos importantes en un determinado instante.

Multi-Head Attention

Uno de los principales problemas de las redes neuronales recurrentes clásicas es ls imposibilidad de la paralelización de su ejecución, y el crecimiento de los términos a considerar conforme más avanzado es el análisis de la secuencia en concreto.

La técnica de *multi-head attention* soluciona estos problemas. En primer lugar, hace la paralelización del modelo no solo posible, sino muy fácil. Cada módulo calcula por separado la atención que le corresponda y posteriormente se concatenan y se transforman linealmente en la salida de la dimensión que se espera.

Por otro lado, los modelos tradicionales sufren de no considerar dependencias entre elementos si estos están distantes en la secuencia. Los diferentes módulos del cálculo paralelo calculan la atención en diferentes puntos de la secuencia de forma paralela, como comentamos. Las diferentes zonas pueden efectivamente considerar zonas muy distantes en tiempo constante, gracias a la paralelización. Esto efectivamente soluciona el problema de olvidar características importantes de puntos distantes, así como evitando tener que crear caminos de cálculo cada vez más largos conforme avanzamos en el análisis de la secuencia.

2.3.2. Modelos basados en transformers

Habiendo descrito brevemente la arquitectura de un transformer en general, veamos qué se ha desarrollado con esta nueva tecnología.

BERT

BERT son las siglas en inglés de *Bidirectional Encoder Representations from Transformers* [28]. Es un modelo de lenguaje generativo basado en un encoder bidireccional como su nombre indica. Este modelo fue uno de los primeros en realmente conseguir una fluidez comunicativa convincente. Su arquitectura preentrenada permite, con una sola capa extra, craer modelos para tareas en casi cualquier ámbito, como responder preguntas o inferencia del lenguaje, sin necesidad de modificar particularmente su arquitectura interna.

GPT-3 Open AI

GPT son las siglas de *Generative Pre-trained Transformer* [29] y 3 indica la versión. Este modelo trata de abarcar los problemas que otros transformers solían tener, como es la habilidad de un humano para continuar una tarea lingüística dado muy poco contexto o instrucciones. Escalando los modelos se logra mejorar significativamente la generalización del mismo y se descubre que no es necesario afinar los parámetros de los modelos, sino que esto puede corresponderse más con un problema de meta-aprendizaje.

LaMDA

LaMDA corresponde con las siglas de *Language Model for Dialogue Applications* [30], un proyecto de Google que compite de forma directa con los modelos antes mencionados. Al igual que los dos proyectos anteriores, LaMDA está basado en un transformer, una arquitectura de redes neuronales creada también por Google [27], que explicaremos en más profundidad en los siguientes capítulos. Este proyecto se creó con una aplicación en concreto: un *chatbot* automático lo más natural posible, al que llamaron *Meena*. Según las estadísticas de Google, Meena tiene casi el doble de capacidad de predicción e inferencia que el antiguo GPT-2, y se entrenó en 8 veces más datos. Es el buque insignia de la empresa.

3. Datos: EDA y preprocesamiento

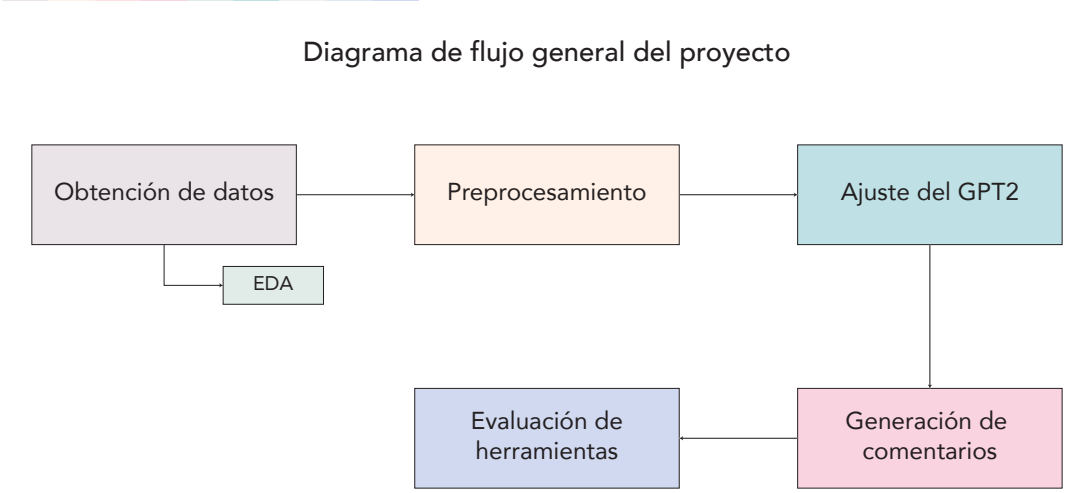


Figura 3.1: Diagrama de flujo general de todo el proyecto

En este capítulo discutiremos los distintos detalles de la implementación de nuestro sistema: desde el preprocesamiento de los datos hasta la elección y entrenamiento del modelo.

En la figura 3.1 podemos observar una vista general de los diferentes pasos de los que consta nuestro proyecto. En cada paso, cuando proceda, expandiremos el nodo concreto para ver de qué subtareas se compone cada una de estas tareas más generales.

3.1. Datos: fuente y forma

Los datos escogidos provienen del dataset Medical Text publicado por Chaitanya Krishna Kasaraneni, y de Medical Transcriptions, publicado por Tara Boyle. La naturaleza de los mismos es ligeramente diferente así que explicaremos el proceso de preprocesamiento y unificación posteriormente.

En la figura 3.5 podemos ver un pequeño resumen del preprocesamiento que se acometerá a los datos. En la siguiente sección se detallan cada uno de estos pasos.

3.1.1. Medical Text

El dataset tiene formato `.dat`, estructurado como un `.tsv` (Tab Separated Values). La primera columna corresponde con una categoría determinada –ya que el dataset estaba diseñado para clasificación– y la segunda columna contiene fragmentos de documentos médicos.

El dataset está descompuesto en un archivo `train.dat` y otro `test.dat`. El archivo de entrenamiento contiene 14438 comentarios, y el de evaluación, 14442. En total, disponemos de 28880 comentarios.

Análisis de datos exploratorio

En esta subsección, analizaremos en más profundidad la forma de los datos, para saber qué esperar de cara al entrenamiento de nuestros modelos.

Como podemos observar en las figuras 3.2a y 3.2b, la distribución de los distintos elementos de nuestro dataset de entrenamiento está muy normalmente distribuída.

El número medio de caracteres por comentario es de 1230, y el número medio de tokens por comentario es de unos 180, correspondiéndose con las líneas amarillas en las figuras.

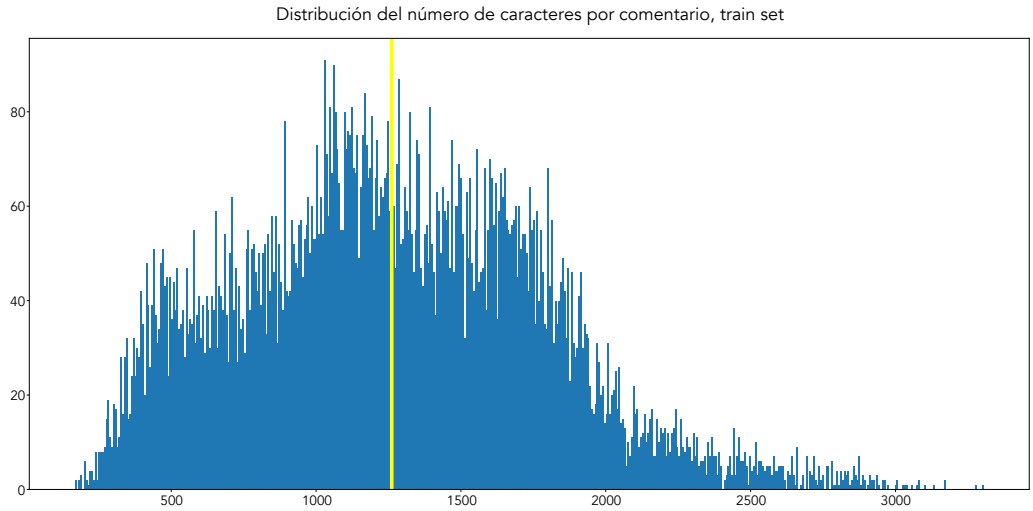
Se pueden apreciar, aún así, algunos valores atípicos de comentarios particularmente largos. Esto, sin embargo, no es necesariamente malo en nuestro caso. En definitiva, cuanto más texto tengamos a nuestra disposición, mejor para el modelo.

Una media de casi 200 palabras por comentario con comentarios alcanzando las 500 corresponde con comentarios relativamente largos. Esto nos vendrá bien de cara al entrenamiento de nuestro modelo, para poder formar oraciones con más sentido.

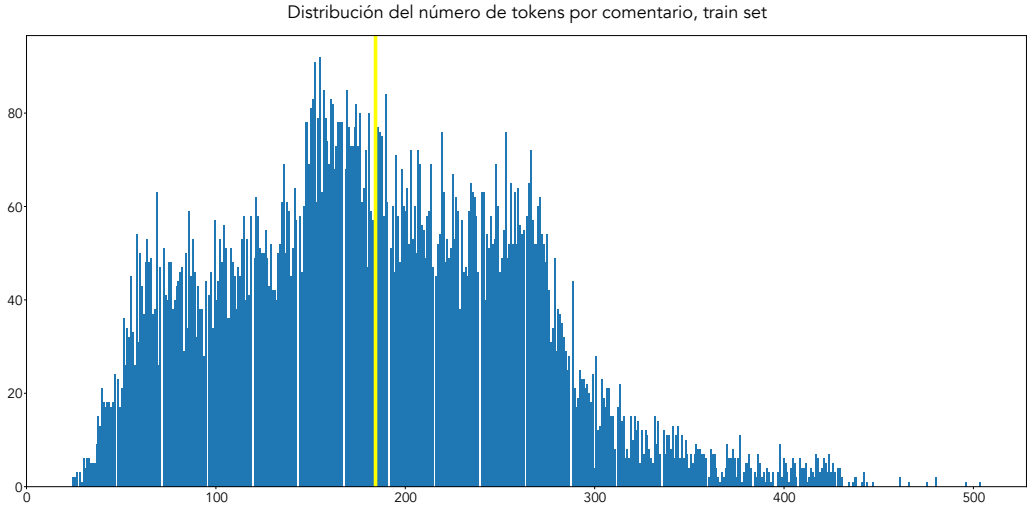
3.1.2. Medical Transcriptions

Este dataset es en realidad una extracción de la página web `mtsamples.com`, donde se halla una respetable cantidad de trasncripciones médicas. La autora extrajo todos los comentarios, así como los diferentes metadatos que los acompañaban mediante *web scraping* y los provee en la columna `trasncription`.

En este caso, como podemos apreciar en la figura 3.4, las distribuciones son ligeramente asimétricas, predominando comentarios más cortos. Aún así, disponemos de comentarios excepcionalmente largos, con alrededor de 18000 caracteres.



(a) Distribución del número de caracteres por comentario, en el conjunto de entrenamiento



(b) Distribución del número de tokens por comentario en el conjunto de entrenamiento

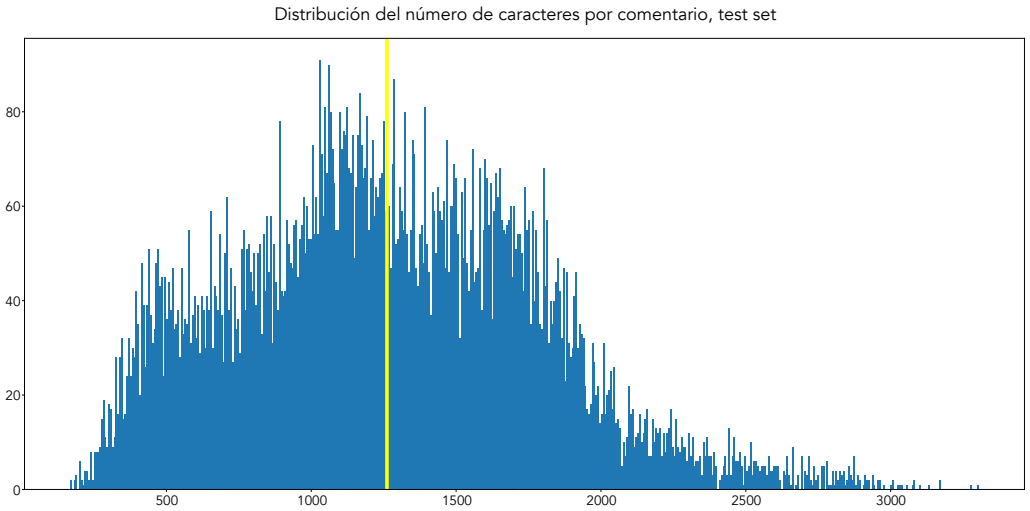
Figura 3.2: Visualización de la distribución de nuestro conjunto de entrenamiento

3.2. Preprocesamiento

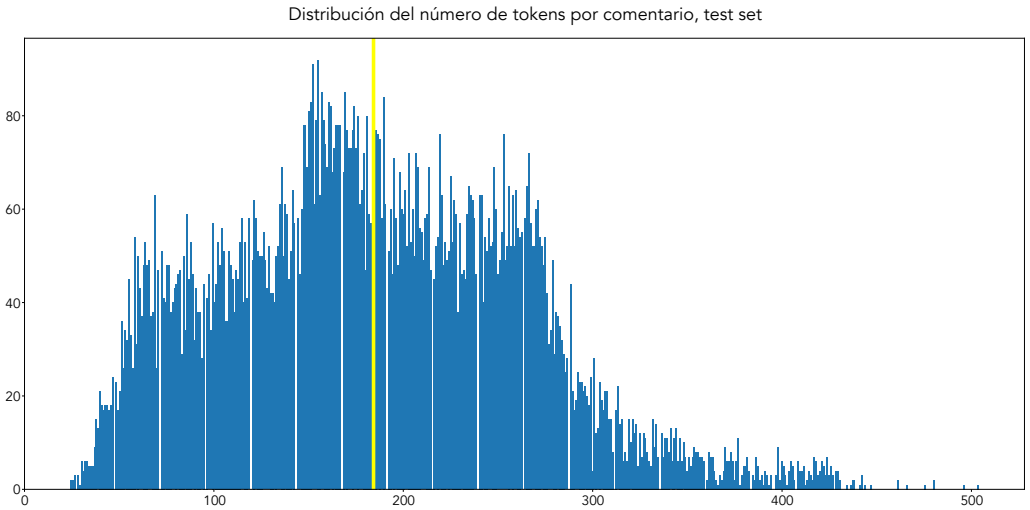
En esta sección, describiremos el preprocesamiento acometido en cada uno de los datsets. Provienen de fuentes diferentes así que cada uno recibirá un trato diferente, con objeto de normalizar y unificar el formato de todos de cara al entrenamiento.

3.2.1. Medical Text

Este conjunto de datos, siendo específicamente texto, el formato, ortografía y en general formato de los archivos es muy bueno. Simplemente hemos de eliminar las categorías adjuntas a cada comentario, para obtener una lista de comentarios crudos en sí. Por lo



(a) Distribución del número de caracteres por comentario, en el conjunto de evaluación



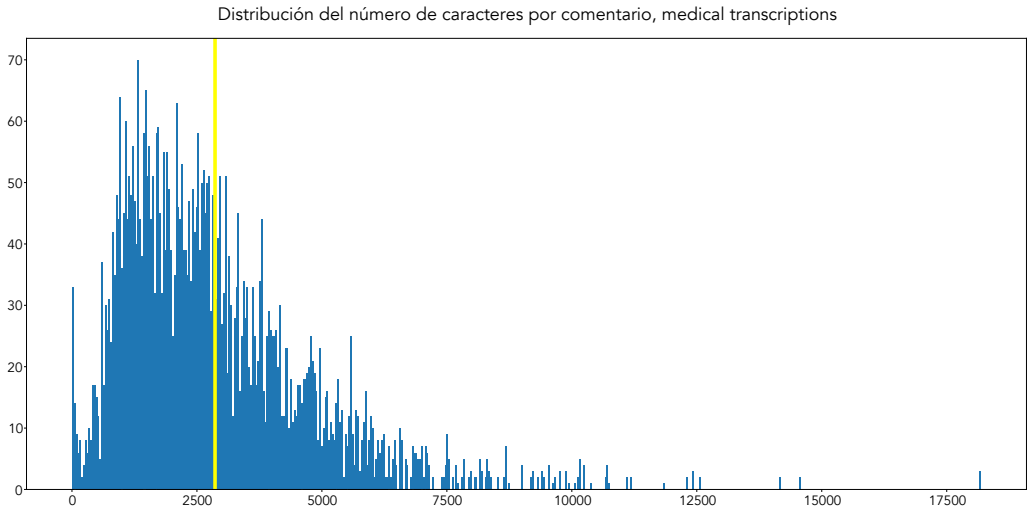
(b) Distribución del número de tokens por comentario en el conjunto de evaluación

Figura 3.3: Visualización de la distribución de nuestro conjunto de evaluación

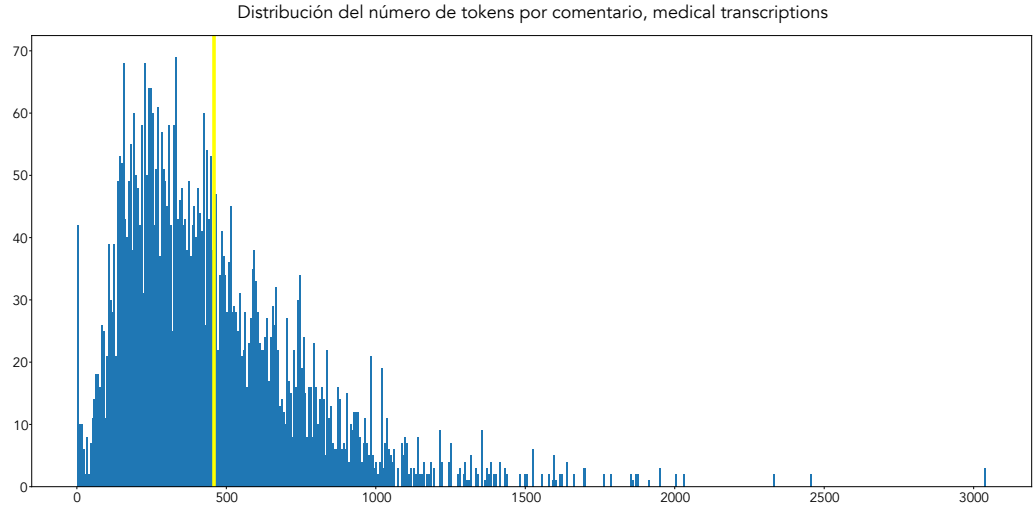
demás, los comentarios carecen de problemas de formato, codificación o cualquier otra cosa que pudiera molestar. Probablemente el autor ya hiciera esto por nosotros antes de publicarlo.

3.2.2. Medical Transcriptions

En el caso de las trancripciones médicas, la tarea es considerablemente más compleja. El conjunto de datos proviene de la página web `mtsamples.com`, como especificamos anteriomente. La autora efectuó un proceso de *scraping* para obtener toda la información y recogerla en el archivo `.csv`.



(a) Distribución de caracteres en el dataset Medical Transcriptions



(b) Distribución de palabras en el dataset Medical Transcriptions

Figura 3.4: Visualización del dataset Medical Transcriptions

Esto facilita las cosas, pero desde luego los comentarios deben ser profundamente esculpidos antes de pasarlos a cualquier modelo. Los trazos de formato HTML se dejan entrever en los comentarios con signos de puntuación o tabulaciones fuera de lugar, así que debemos arreglarlo previo entrenamiento.

Para ello, se ha hecho un fuerte uso de expresiones regulares, y se ha creado un *pipeline* para procesar todo el texto a la vez.

El pipeline elimina todas las posibles trazas o residuos que hubieran quedado del *scraping*. Podemos ver el pipeline diseñado en la figura A.1.

El resumen del proceso es eliminar signos de puntuación mal colocados, eliminar títulos o cabeceras de secciones de la página web, sustituir múltiples espacios por uno solo o

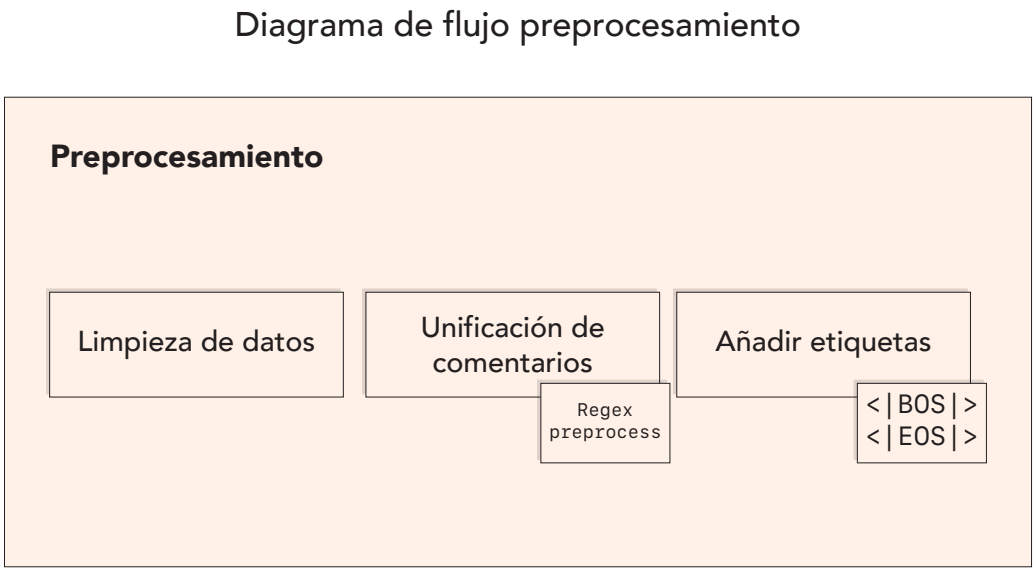


Figura 3.5: Diagrama resumen del preprocesamiento efectuado en los datos.

eliminar los números de listas enumeradas (1., 2., etc). Finalmente, se añaden las etiquetas que vemos en la figura. Se explicará su funcionamiento en la sección de la experimentación.

El resultado es un texto muy limpio y claro, mucho más apto para la fase de entrenamiento.

4. Experimentación: modelos y entrenamiento

En este capítulo usaremos todos los conceptos vistos en los capítulos anteriores para justificar las elecciones de los diferentes modelos, hablaremos de las características principales de los mismos y finalmente entraremos en la fase del entrenamiento y los resultados de dichos modelos.

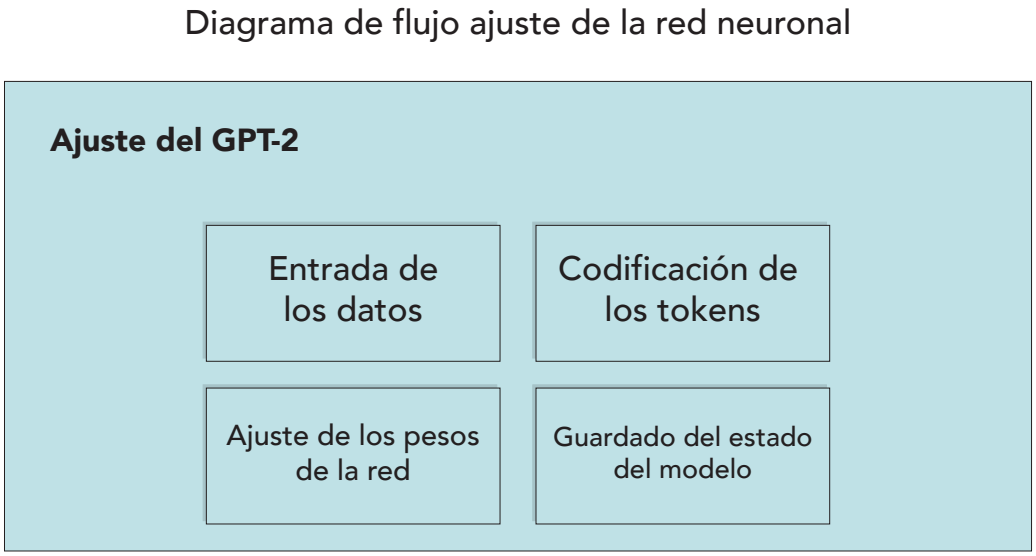


Figura 4.1: Diagrama resumen del ajuste de los pesos del GPT2

En la figura 4.1 podemos ver un breve resumen de las diferentes tareas de las que se compone el entrenamiento del modelo. Pasaremos a explicar en profundidad cada una de esas subtareas en las siguientes secciones.

4.1. Experimentación

En esta sección describiremos los diferentes pasos que fue necesario acometer para el entrenamiento del modelo del GPT2, el transformer que hemos utilizado para nuestro proyecto.

4.1.1. Entrada de los datos

En primer lugar, debemos ofrecer los datos de entrada al modelo. Estos datos están previamente formateados y unificados, como vimos en la fase de preprocesamiento.

Es importante la adición de las etiquetas `<|BOS|>` y `<|EOS|>`. Estas etiquetas son las siglas de *begin of sentence* y *end of sentence*. Son marcadores que indican, como es obvio, el inicio y fin de una oración. Estas etiquetas son necesarias para que el modelo entienda cuál es el criterio a partir del cual empieza y acaba una oración.

4.1.2. Codificación de los tokens

Antes de pasar los datos al modelo, debemos codificarlos. Codificar significa obtener un código único en función del token que estemos considerando. La codificación es esencial para estos modelos, ya que ofrece una representación matemática de las palabras. La codificación de los tokens se efectúa mediante un diccionario preentrenado, de forma que la codificación decodificación sean consistentes, dado un modelo determinado.

4.1.3. Ajuste de los pesos

Nuestro modelo ya está construido. Esto es, los diseñadores ya han decidido las distintas capas y el orden de estas según hemos visto en las secciones anteriores. Este modelo se nos ofrece preentrenado de forma bastante simple. Tal y como viene en el paquete, es capaz de generar frases con relativo sentido, es decir, de algún modo *sabe hablar*. El ajuste se efectúa como un proceso de aprendizaje no supervisado, en la que exponemos a la red a un conjunto de comentarios de forma que ésta podrá generar nuevos comentarios de la nada que caigan bajo la función de distribución de los comentarios de entrada.

En nuestro caso, esta *función de distribución* la define nuestro conjunto de datos de informes médicos. Este proceso efectivamente ajusta los pesos de la red de forma que se familiarice al modelo con el vocabulario y expresiones comunes encontradas en nuestro conjunto de datos.

Como sabemos, este proceso es, computacionalmente, extraordinariamente costoso. Debemos calcular el peso de millones de parámetros (117 millones en nuestro caso particular), debido a la magnitud del modelo. Para ello, hemos de tener disponible un equipo con, como mínimo, una tarjeta gráfica decente que nos permita hacer cálculos matriciales en paralelo, operaciones muy comunes en el entrenamiento de las redes neuronales.

Dichos equipos pueden ser muy caros. Para ello, hicimos uso del servicio de clústeres de la UGR, enviando nuestros datos mediante `ssh`, entrenando el modelo, y descargándonoslos de vuelta.

4.1.4. Guardado del estado del modelo

Finalmente, una vez entrenado el modelo, lo más importante es guardar su estado. Esto es muy fácil gracias a los métodos provistos por las librerías de deep learning. Este estado se guarda en un archivo, que, generalmente, consta de un diccionario en el que las claves corresponden a los nodos de las capas en sí, es decir, a sus parámetros entrenables, y cuyo valor es el peso de dicho nodo.

De esta forma, es fácil cargar un modelo *vacío*, un modelo inicializado con pesos nulos o no relevantes, y sustituir dichos valores por los incluidos en el archivo. Esto nos permite volver al estado en el que dejamos al modelo tras el costoso entrenamiento.

4.2. Generación de comentarios

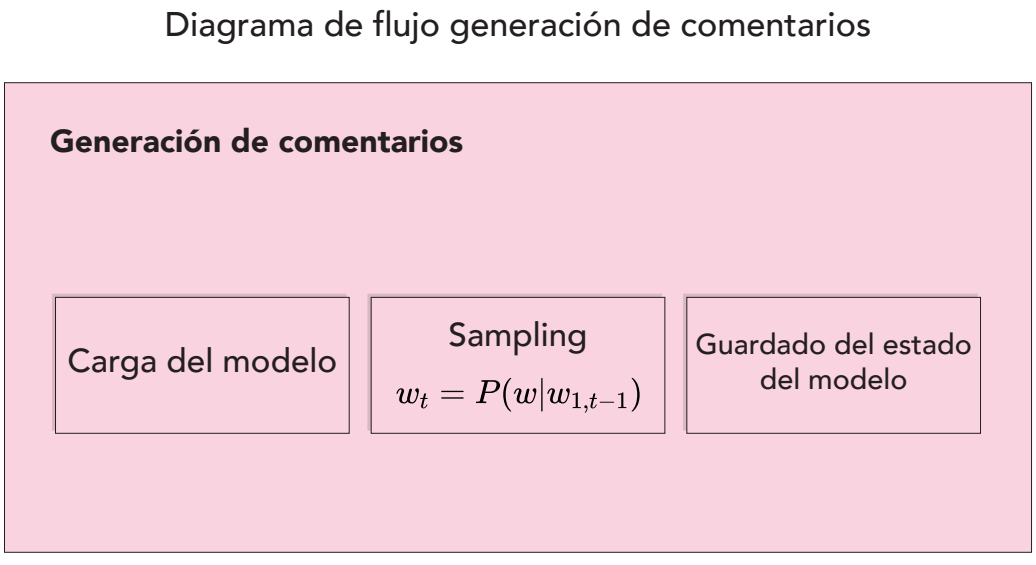


Figura 4.2: Diagrama resumen de la generación de comentarios

En esta sección hablaremos de la generación de comentarios, una vez nuestro modelo está entrenado y listo para funcionar.

4.2.1. Carga del modelo

Como indicamos anteriomente, la carga del modelo es realmente sencilla gracias a las librerías que lo hacen posible. Con nuestro archivo con los pesos localizado, podemos cargar de vuelta los pesos relevantes en los parámetros correspondientes de la red, recuperando el estado original. Este estado es el que nos permitirá generar comentarios de forma automática.

4.2.2. Sampling y otros métodos de generación de texto

Ahora que nuestro modelo está listo, podremos generar comentarios de la nada, casi como por *arte de magia*. Aún así, este proceso no es trivial y merece ser aclarado.

Debemos gran parte de esta sección a la magnífica explicación de Patrick von Platen en su publicación How to generate text. Precisamente es uno de los integrantes de Hugging Face, una empresa de código abierto que se dedica a la creación de los diferentes modelos de los que hemos hablado anteriormente, y que nos ha provisto con una manera fácil y accesible de poder utilizar estas herramientas tan potentes.

Existen varias maneras de generar lo que en la jerga se denomina *texto abierto*, es decir, texto libre sin restricciones. Muchos modelos son capaces de hacerlo, aunque nosotros nos centraremos en los métodos que conciernen al GPT2, que es un modelo autorregresivo.

Los modelos autorregresivos asumen que la siguiente palabra a generar se calcula como una función de probabilidad de todas las palabras anteriores, dada una palabra de contexto inicial.

Greedy and beam search

La búsqueda voraz obtiene la palabra con mayor probabilidad de todas las opciones dada una palabra inicial.

Dada una palabra inicial que tomamos del usuario, por ejemplo, siempre tomaremos aquella palabra de todas las posibles opciones que más probabilidad tenga de aparecer, dada la palabra anterior. Dichas probabilidades se calculan en el entrenamiento del modelo.

El algoritmo toma siempre la palabra más probable y esto funcionará bien, aunque este algoritmo peca de empezar a repetirse bastante pronto. Esto es, generará secuencias

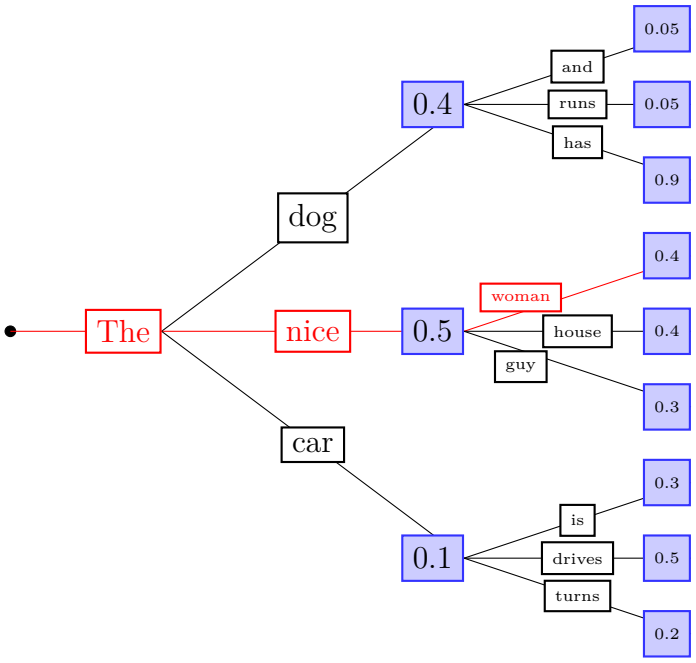


Figura 4.3: Ilustración de los pasos que daría el algoritmo greedy, tomando siempre la posibilidad más alta

que contienen palabras finales e iniciales similares, por lo que el ciclo empieza de nuevo al escogerse siempre la palabra más probable.

Una solución planteada es la *beam search*, algo así como una búsqueda distribuida. Dadas las probabilidades de cada palabra, el algoritmo calcula varios pasos en avanzadilla para varias alternativas, y devuelve aquella secuencia que más probabilidad general tenga. Esto soluciona algunos problemas, pero no termina de lidiar con el factor de repetitividad anterior.

Ari Holtzmann sugiere en [31] que la generación del lenguaje humano no se guía por la elección de las palabras más probables. El lenguaje humano real es mucho menos predecible, ya que de otra forma sería *aburrido* de leer o escuchar.

Es por ello que los métodos alternativos han de evitar las técnicas deterministas de generación de texto, y apostar por los métodos con gran influencia aleatoria.

Sampling

Sampling es una técnica de generación de lenguaje autorregresiva, que, como sabemos, asume la probabilidad de una palabra dada la anterior.

En la forma más básica, el muestreo simplemente toma una palabra de forma aleatoria, ponderándolas con una distribución de probabilidad tal que:

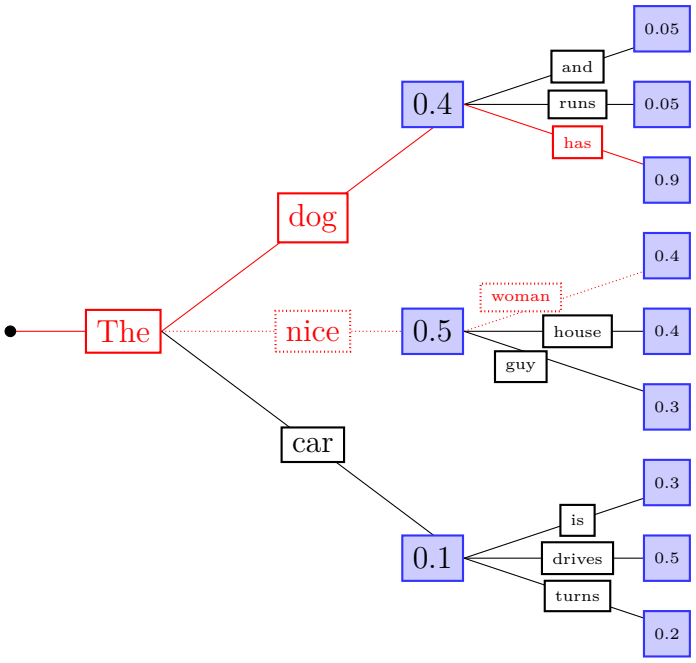


Figura 4.4: Ilustración demostrando el funcionamiento del *beam search*, con 2 beams en este caso. Vemos como se toma un camino que a priori es menos probable pero resulta en una probabilidad más alta al final.

$$w_t = P(w|w_{1,t-1}) \tag{4.1}$$

es decir, la probabilidad de escoger la palabra w_t depende de todas las palabras anteriores.

Podemos graduar la ponderación de la que hablamos con lo que los autores denominan la *temperatura* de la función de activación softmax de la última capa del modelo.

Al bajar la temperatura por debajo de 1 (1 estando efectivamente desactivada esta técnica), las palabras más probables se saturan y las menos probables disminuyen. De ser el caso de ajustar la temperatura a límites muy cercanos a 0, nos encontraríamos con la búsqueda voraz de la que hablamos antes.

La adición de esta técnica ayuda a que la generación no sea *extremadamente* aleatoria. Si bien en la sección anterior comentábamos que debíamos añadir aleatoriedad, todo en exceso es malo. Una generación completamente aleatoria es, en su mayoría, poco coherente. La bajada sutil de la temperatura del modelo provoca que se elijan, normalmente, palabras bastante probables, pero que de vez en cuando se escojan palabras poco probables. Dichas palabras ahora abren nuevos caminos por los que continuar generando, que no hubieran sido considerados de otra forma, pero continuamos eligiendo, por lo general, combinaciones de palabras más probables, para garantizar la coherencia del texto.

Top K Sampling

Dadas las premisas anteriores, las siguientes técnicas son mejoras incrementales que tratan de solucionar los problemas que podamos encontrarnos.

El muestreo de los K mejores, tal y como su nombre indica, solo considera las K mejores opciones de toda la batería de palabras posibles. Dado este subconjunto K, se muestrea una palabra como en el Sampling original, tomándola aleatoriamente de forma ponderada.

Esta técnica elimina posibles elecciones bastante malas, lo que Holtzmann denomina como "*the unreliable tail*", que podemos acuñar como *la cola traicionera*. Holtzmann afirma que existe una enorme cantidad de palabras con muy baja probabilidad en cada elección que, en realidad, están sobrerepresentadas cuando hablamos de términos agregados. Es decir, la **probabilidad agregada** de todas esas palabras con una bajísima probabilidad puede corresponder a **grandes secciones de la distribución de probabilidad**, lo que, efectivamente, va en contra de nuestro objetivo.

Tomando solo aquellas K mejores, se elimina dicha cola y se redistribuye la probabilidad de los términos elegidos de forma oportuna, creando un modelo de decodificación mucho más natural que los otros.

Top P nucleus

Finalmente, esta última técnica se denomina Top P nucleus, refiriéndose a un valor diferente al K anterior.

En este caso, en lugar de tomar los K mejores términos, tomamos el conjunto más pequeño de términos cuya probabilidad acumulada supere el p establecido por el usuario. Esto es, dado un $p = 0{,}9$, tomamos las n mejores palabras que, dada la suma de su probabilidad, se supere el p determinado.

Esto provoca que en lugar de tomar un subconjunto de palabras de cardinalidad constante durante todo el proceso, ahora el tamaño varía.

Estos dos últimos métodos son los más utilizados en el estado del arte, y este último es el que utilizamos nosotros concretamente en nuestro proyecto a la hora de generar palabras.

4.3. Resultados

A. Apénice

Incluiremos en este apéndice todos los bloques de código o cualquier otro tipo de contenido necesario para comprender la estructura del documento, pero evitando que interfiera con la lectura del mismo.

```python
def regex_processing(text):
    # Remove capital letters surrounded by 0 or more `,` and a colon,
    ↪ i.e. the titles
    no_caps = re.sub(r',*([A-Z\s]+):', '', text)

    # Remove weirdly positioned commas. Find commas that dont have any
    ↪ letter before and some space after them.
    weird_commas = re.sub(r'(?<!\w),\s+', '', no_caps)

    # Remove commas that dont have spaces around them. (Commas should
    ↪ always have a trailing space after them)
    more_commas = re.sub(r'(?<!\s),(?!\s)', ' ', weird_commas)

    # Remove digits adyacent to dots or commas, as in enumerated lists.
    no_digits = re.sub(r'[\.,]*\d[\.,]+', ' ', more_commas)

    # Remove any other commas left behind the process. Particularly these
    ↪ cases: Hello. ,How are you?
    trailing_commas = re.sub(r'\s,(?=[A-Z\d])', '', no_digits)

    # Substitute any number of spaces for 1 single space.
    no_double_spaces = re.sub(r'\s+', ' ', trailing_commas)

    # Solve these problems: Hello .How are you? => Hello. How are you?
    final_text = re.sub(r'(?<!\s)\.(?!\s)', '. ', no_double_spaces)

    # Finally, strip the text from any trailing commas or white spaces.
    # The result is hopefully a clean version of the text, ready to be
    ↪ tokenized
    # and passed to the models.
    return final_text.strip(', ')
```

Figura A.1: Pipeline para el procesamiento de los comentarios de Medical Transcriptions

Bibliografía

[1] Luis Pereira, Rui Rijo, Catarina Silva, and Ricardo Martinho. Text mining applied to electronic medical records: A literature review. *International Journal of E-Health and Medical Communications (IJEHMC)*, 6:1–18, 07 2015.

[2] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In *Knowledge Discovery in Databases*, 1991.

[3] R. Kumar and Rajesh Verma. Classification algorithms for data mining: A survey. *International Journal of Innovations in Engineering and Technology (IJIET)*, 1:7–14, 2012.

[4] Anil K. Jain, M. N. Murty, and P. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31:264–323, 1999.

[5] Jiawei Han, Micheline Kamber, and Jian Pei. 6 - mining frequent patterns, associations, and correlations: Basic concepts and methods. In Jiawei Han, Micheline Kamber, and Jian Pei, editors, *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 243–278. Morgan Kaufmann, Boston, third edition edition, 2012.

[6] Kamran Kowsari, K. Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura E. Barnes, and D. Brown. Text classification algorithms: A survey. *Inf.*, 10:150, 2019.

[7] Umar Farooq, Hasan Mansoor, A. Nongaillard, Y. Ouzrout, and M. Qadir. Negation handling in sentiment analysis at sentence level. *J. Comput.*, 12:470–478, 2017.

[8] Omar Ali, A. Gegov, Ella Haig, and R. Khusainov. Conventional and structure based sentiment analysis: A survey. *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[9] Vimala Balakrishnan and Lloyd-Yemoh Ethel. Stemming and lemmatization: A comparison of retrieval performances. *Lecture Notes on Software Engineering*, 2:262–267, 01 2014.

[10] Thomas Cover and Joy Thomas. *Elements of Information Theory*, volume 36, pages i – xxiii. 10 2001.

[11] K. Jones. A statistical interpretation of term specificity and its application in retrieval. *J. Documentation*, 60:493–502, 2004.

[12] Tomas Mikolov, Kai Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *ICLR*, 2013.

[13] Jeffrey Pennington, R. Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

[14] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[15] Oren Melamud, J. Goldberger, and Ido Dagan. context2vec: Learning generic context embedding with bidirectional lstm. In *CoNLL*, 2016.

[16] I.T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.

[17] A. Hyvärinen. Topographic independent component analysis. In *Encyclopedia of Computational Neuroscience*, 2014.

[18] Stan Z. Li and Anil Jain, editors. *LDA (Linear Discriminant Analysis)*, pages 899–899. Springer US, Boston, MA, 2009.

[19] Eric Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–139, 2004.

[20] Cox Dr and EJ Snell. The analysis of binary data, 1989.

[21] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. A c-lstm neural network for text classification. *CoRR*, abs/1511.08630, 2015.

[22] Xiao-Peng Yu and Xiao-Gao Yu. Novel text classification based on k-nearest neighbor. In *2007 International Conference on Machine Learning and Cybernetics*, volume 6, pages 3425–3430, 2007.

[23] Eibe Frank and Remco R. Bouckaert. Naive bayes for text classification with unbalanced classes. In *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, ECMLPKDD'06, page 503510, Berlin, Heidelberg, 2006. Springer-Verlag.

[24] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398, pages 137–142, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.

[25] Wan Noormanshah, Puteri Nohuddin, and Zuraini Zainol. Document categorization using decision tree: Preliminary study. *International Journal of Engineering and Technology*, 7:437–440, 12 2018.

[26] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[28] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[29] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[30] Daniel Adiwardana, Minh-Thang Luong, David R. So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, and Quoc V. Le. Towards a human-like open-domain chatbot. *CoRR*, abs/2001.09977, 2020.

[31] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. 2020.