UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Tratamiento de los datos para un sistema de traducción automática adaptado a un dominio

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Beteta Francisco, Daniel

Tutor/a: Pla Santamaría, Ferran

CURSO ACADÉMICO: 2021/2022

Resumen

La finalidad del presente trabajo es mostrar cuantitativamente el efecto positivo de una correcta limpieza, normalización y clasificación de los datos en la creación de motores de traducción.

Para ello, por un lado, se realizará un preproceso de limpieza y normalización de los datos para la combinación de idiomas inglés-español. Por otro lado, se desarrollará un clasificador de la temática de los textos que nos ayude a mejorar la traducción en un dominio determinado.

El objetivo es poder entrenar dos máquinas de traducción automática: la primera con datos limpios, normalizados y clasificados en el dominio “Health”, y la segunda con datos “sucios”, es decir, sin ninguno de los procedimientos anteriormente nombrados.

Finalmente, se realizará una evaluación cuantitativa para determinar la calidad de la traducción, utilizando las medidas usuales para validar la aproximación propuesta en este trabajo de final de grado.

Palabras clave: preproceso de limpieza y validación, clasificador de dominio, traducción automática neuronal.

Resum

La finalitat del present treball és mostrar quantitativament l'efecte positiu d'una correcta neteja, normalització i classificació de les dades en la creació de motors de traducció.

Per això, d'una banda, es farà un preprocés de neteja i normalització de les dades per a la combinació d'idiomes anglès-espanyol. D'altra banda, es desenvoluparà un classificador de la temàtica dels textos que ens ajudi a millorar-ne la traducció en un domini determinat.

L'objectiu és poder entrenar dues màquines de traducció automàtica: la primera amb dades netes, normalitzades i classificades al domini “Health”, i la segona amb dades “brutes”, és a dir, sense cap dels procediments anteriorment anomenats.

Finalment, es farà una avaluació quantitativa per determinar la qualitat de la traducció, utilitzant les mesures usuals per validar l'aproximació proposada en aquest treball de final de grau.

Paraules clau: preprocesso de neteja i validació, classificador de domini, traducció automàtica neuronal.

Abstract

The purpose of this work is to show quantitatively the positive effect of a correct cleaning, normalization and classification of the data in the creation of translation engines.

To do this, on the one hand, a data cleaning and normalization pre-process will be carried out for the English-Spanish language combination. On the other hand, a classifier of the theme of the texts will be developed to help us improve the translation in a given domain.

The objective is to be able to train two automatic translation machines: the first with clean, normalized and classified data in the "Health" domain, and the second with "dirty" data, that is, without any of the aforementioned procedures.

Finally, a quantitative evaluation will be carried out to determine the quality of the translation, using the usual measures to validate the approach proposed in this final degree project.

Keywords: cleaning and validation preproces, domain classifier, neural machine translation.

Tabla de contenidos

1. Introducción

En este trabajo se lleva a cabo una demostración cuantitativa de la importancia del preprocesado y clasificación de los datos en el contexto de la creación de motores de traducción especializados en un dominio.

En este capítulo de introducción se trata la motivación, objetivos, impacto esperado y metodología del TFG.

1.1 Motivación

Tras haber realizado prácticas en una empresa dedicada al Procesamiento del Lenguaje Natural (NLP) en la que tuve la oportunidad de realizar tareas de limpieza y modelado de datos lingüísticos, tomé verdadera conciencia sobre cómo el preprocesado de los datos lingüísticos es una pieza fundamental y no trivial para cualquier tarea de NLP, como por ejemplo: detección de entidades nombradas (NER), análisis de sentimiento, o máquinas de traducción automática.

De igual manera, tengo la convicción de que, en el ámbito de la limpieza de datos, cada programador va definiendo qué significa tener los datos “limpios” según las necesidades del proyecto y esto no es algo malo per se.
Pero lo que sí considero una mala práctica es que la comunidad tenga que reescribir una y otra vez las mismas funciones para “limpiar” los datos, además de que de esta manera se hace realmente difícil llevar un seguimiento de qué “limpieza” se ha aplicado.

Por todo ello, y bajo mi automotivación para, por un lado, aunar todo lo aprendido en las prácticas, lo cual va, desde la aplicación práctica de conceptos estudiados en el grado de Ingeniería Informática hasta el desarrollo de la pasión por un código limpio y eficiente. Y, por otro lado, para seguir desarrollándome como profesional en el mundo de la Ciencia de Datos considero pertinentes los objetivos descritos en el siguiente apartado.

1.2 Objetivos

El objetivo principal es doble: crear un repositorio open-source en Github en el que se reúnan los principales procesos de limpieza para obtener el máximo valor de un conjunto de datos lingüísticos; y desarrollar un clasificador de dominio para detectar qué partes del conjunto de datos hablan sobre la temática “Health”.

A raíz de dicho objetivo principal, y para comprobar su funcionalidad, el objetivo secundario será: entrenar por un lado una máquina de traducción con los datos limpiados y clasificados y por otro entrenar otro motor de traducción con esos mismos datos, pero sin los procedimientos anteriormente nombrados, para así comparar los resultados de ambos.

1.3 Impacto esperado

El principal impacto que busco causar con el presente trabajo final de grado es en base al desarrollo del repositorio GitHub anteriormente comentado, ya que creo que tiene el potencial de simplificar las tareas de preprocesado lingüístico, permitiendo definir de forma precisa qué entendemos cada vez por "limpieza" y permitiendo llevar un seguimiento de forma sencilla.

De igual forma, me parece una buena oportunidad para que colaborativamente se desarrolle un repositorio open-source, donde poco a poco se vaya completando con más procesos de limpieza adecuados para otros idiomas y otro tipo de proyectos del ámbito NLP.

1.4 Metodología

La metodología usada en el presente trabajo es eminentemente práctica, se busca comprobar la utilidad de las herramientas desarrolladas mediante un enfoque experimental. Más concretamente, la metodología se basa en el aprendizaje automático, para ello se recopilan los datos, se depuran, se realiza el entrenamiento de los modelos y se evalúan de forma automática usando medidas estándar en la literatura.

De igual forma, y para una mayor comprensión de lo realizado, se complementa teóricamente, describiendo el funcionamiento de los modelos.

2. Recursos de texto

En este capítulo se detalla el origen de los datos necesarios para llevar a cabo el proyecto.

Cabe destacar que la facilidad para encontrar dichos corpus ha sido gracias a las prácticas en empresa anteriormente comentadas.

2.1 Corpus monolingües

El corpus monolingüe que se busca es en inglés, se elige este idioma y no el español debido a que nos facilitará la tarea a la hora de generar datos etiquetados para el clasificador de dominio.

De tal forma, tenemos el proyecto "Deutscher Wortschatz" de la Universidad de Leipzig, el cual lleva en activo desde mediados de 1990, y tiene su origen en la recopilación y procesado de textos de habla alemana para desarrollar un diccionario en el que para cada palabra hay disponible una página con información estadística, frases de ejemplo y enlaces a palabras relacionadas.

Sin embargo, en la actualidad se ha ampliado su servicio a más y más idiomas utilizando el nombre de "Leipzig Corpora Collection", llegando así a disponer de corpus para más de 250 lenguas, que además se ofrecen de forma gratuita.

Hablando ahora de su forma de extracción de dicho corpus, el proceso que siguen es el siguiente: extraen textos de distintas páginas (Wikipedia, Noticias, Webs cuidadosamente seleccionadas), separan las diferentes oraciones y las mezclan de forma aleatoria para hacer irreconocible el texto original, y por último, eliminan dichos originales. Todo este flujo se hace también para evitar infringir las leyes sobre el copyright (Wortschatz-Leipzig, 1990).

Gracias a dicho proyecto, fui capaz de conseguir 15 millones de sentencias en inglés. Los conjuntos de datos utilizados están citados según su normativa en el apartado "Origen corpus monolingües".

2.2 Obtención del glosario "Health"

El glosario, al igual que el corpus monolingüe, se requiere en inglés y en este caso se ha elegido la temática "Health", la cual, puede agrupar un gran número de campos, pero que sin embargo, es lo suficientemente concreta como para diferenciarse del resto de dominios.

Así, la forma que se ha empleado para extraer un glosario de alrededor de 1000 términos es la siguiente:

1. Entrar al portal ProZ, el cual, es un lugar de encuentro para que los traductores y otras personas interesadas se ayuden mutuamente con la traducción de términos y frases cortas (ProZ).

2. Decidir qué glosarios forman parte del dominio “Health”, en este caso fueron:
    - Biología (biotecnología/química, microbiología)
    - Medicina (general)
    - Medicina: Cardiología
    - Medicina: Odontología
    - Medicina: Salud
    - Medicina: Instrumentos
    - Medicina: Farmacia
    - Nutrición
    - Genética

3. Descargar el html de cada uno de los glosarios elegidos mediante el comando wget:

    wget --recursive --level=inf --no-parent --wait=5 --limit-rate 5k --no-cache --no-cookies --header="Accept: text/html" --reject-regex "(.*)\?(.*)" --no-clobber https://www.proz.com/glossary-translations/english-to-spanish-translations/medical-general

    Pasando a explicar las opciones utilizadas del comando wget se tiene:
    - --recursive: permite activar la descarga recursiva.
    - --level: permite seleccionar la profundidad máxima de recursividad que se quiera, en este caso, como se quiere extraer todas las páginas de los glosarios seleccionados la profundidad será infinita.
    - --no-parent: permite que en la descarga recursiva no se ascienda al directorio padre de la url especificada, esto es muy importante ya que antes hemos especificado que la profundidad máxima de la recursión es infinita.
    - --wait: permite seleccionar el tiempo entre peticiones, lo cual, es muy recomendado para evitar que nuestro proceso de “scrapeo” sea cancelado por el servidor.
    - --limit-rate: permite seleccionar el límite de bytes descargados por segundo, contribuye también a la continuidad de la descarga de los htmls.
    - --no-cache: permite descargar la versión del servicio remoto y no la versión guardada en la caché.
    - --no-cookies: permite desactivar el uso de cookies para almacenar sólo la información relevante para la tarea.
    - --header="Accept: text/html": permite enviar la cabecera seleccionada, en este caso “text/html”, en cada petición.
    - --reject-regex "(.*)\?(.*)": permite rechazar las peticiones de descarga que contengan el carácter “?”, de forma que se descarguen solo las diferentes páginas de los glosarios seleccionados.
    - --no-clobber: evita descargar nuevas copias de lo ya descargado.

4. Juntar los diferentes glosarios y eliminar la estructura html para quedarnos solo con los glosarios en su parte en inglés. Esto se ha conseguido gracias al uso de la librería BeautifulSoup mediante un script ad-hoc para esta tarea (BeautifulSoup).

5. Por último, se ha hecho una revisión manual del archivo conseguido en el punto anterior, incluyéndose nuevos términos básicos del dominio y eliminando aquellos que no fueran lo suficientemente específicos.

2.3 Corpus bilingües

El corpus bilingüe que se busca es en la combinación de idiomas inglés-español, y para ello se ha recurrido al proyecto Opus Corpus, el cual, es una creciente colección de textos traducidos de la web cuyo objetivo principal es apoyar al ámbito del "NLP".

En dicho proyecto, se intenta convertir y alinear los datos de forma gratuita, añadiendo anotaciones lingüísticas y proporcionando los datos en varios formatos para facilitar el trabajo con ellos.
Asimismo, cabe destacar que se proporcionan los datos "tal cual", es decir, sin ninguna garantía, esto es así ya que todo el preprocesamiento y la alineación se realiza de forma automática y no se realizan correcciones manuales.

Se pasa ahora a analizar las distintas fuentes de datos que nos han permitido obtener más de 41 millones de datos alineados:

- **UNPC (United Nations Parallel Corpus)**

  Este corpus tiene una cantidad de más de 25 millones de unidades de traducción provenientes de documentos de la ONU traducidos manualmente durante los últimos 25 años (de 1990 a 2014) para los seis idiomas oficiales de la ONU: árabe, chino, español, francés, ruso e inglés (Ziemski, Junczys-Dowmunt, & Pouliquen, 2016).

- **EUbookshop**

  Este corpus tiene una cantidad de más de 5 millones de unidades de traducción provenientes de documentos del sitio web EUbookshop.

  Más concretamente, el flujo de extracción fue: por un lado y gracias a que la web tenía una funcionalidad de búsqueda que devuelve una lista con todos los documentos y por otra gracias a, una herramienta propia que recorre todas estas páginas y crea una lista completa de publicaciones disponibles del sitio web, fue posible la descarga y guardado de los títulos de todas las publicaciones disponibles en aquel momento. Posteriormente para obtener dichos documentos se usó el comando "wget".

  Una vez obtenidos estos documentos en formato PDF, se extrae el contenido textual de todos ellos convirtiendo los documentos PDF a texto plano o al lenguaje de marcado XML, limpiando y filtrando los datos para así luego proceder a la alineación de las frases.
  De tal forma, se soluciona, en la medida de lo posible, problemas como el no poder detectar bien los límites de las palabras o de los párrafos en algunos idiomas (Tiedemann, 2012).

- **DGT (Directorate-General for Translation)**

  Este corpus tiene una cantidad de más de 5 millones de unidades de traducción provenientes del acervo comunitario de la Unión Europea, esto es, el conjunto de la legislación europea, que comprende todos los tratados, leyes y directivas adoptados por la UE.

Además, cabe destacar dos puntos: el primero es que está disponible en los 24 idiomas oficiales de la comunidad europea, lo cual, tiene gran valor a la hora de crear motores de traducción en idiomas con menos recursos lingüísticos disponibles como puede ser el maltés. Y segundo, que dichas alineaciones están hechas de forma manual, característica poco común en conjuntos de datos de este tamaño (Steinberger, Eisele, Klocek, Pilos, & Schlüter, 2012).

- **TildeMODEL**

  Este corpus tiene una cantidad de más de 3 millones de unidades de traducción provenientes del proyecto de la empresa TILDE, cuyo objetivo es contribuir con uno de los grandes problemas a la hora del desarrollo de tecnología lingüística en Europa: la falta de datos, y en especial, de los idiomas que no son mayoritarios.

  Así, cabe destacar que el origen de su conjunto de datos viene principalmente de 5 fuentes:

  1. RAPID: "Data base" de comunicados de prensa de la Comisión Europea en todas las lenguas de la UE. Dichos comunicados se suelen traducir con precisión.
  2. EMA (Agencia Europea de Medicamentos): documentos y descripciones de medicamentos y prospectos de uso, así como diversas condiciones médicas.
  3. Portal de documentos del Comité Económico y Social Europeo
  4. Web del Banco Central Europeo (BCE)
  5. Web del Banco Mundial: contenido sobre los proyectos y actividades del Banco Mundial en las distintas regiones del mundo.

  (Rozis & Skadiņš, 2017)

- **Wikipedia**

  Este corpus tiene una cantidad de casi 2 millones de unidades de traducción provenientes de Wikipedia, dicho corpus ha sido extraído gracias al siguiente método de alineación:

  Primero se guardan los datos en archivos de formato HTML de Wikipedia de un idioma que no sea el inglés, ya que este es el idioma con más recursos en este sitio web, y luego se clasifican por temas. Posteriormente se le da al rastreador el primer enlace al artículo en el dominio correspondiente (para limitar la búsqueda), y así el mismo rastreador obtendrá automáticamente otros documentos relacionados con el tema.

  Después de obtener los documentos HTML, el "crawler" extrae el texto plano de los mismos, desechando el resto de los elementos: tablas, urls, figuras, imágenes, menús, referencias... Por último, los documentos bilingües se etiquetan con un identificador único como un corpus comparable alineado por temas (Wołk & Marasek, 2014).

- **TED2020**

  Este corpus tiene una cantidad de casi medio millón de unidades de traducción provenientes del rastreó de los subtítulos traducidos de unas 4.000 charlas TED, disponibles en más de 100 idiomas.

  Dicho rastreó fue realizado por el Departamento de Informática de la Universidad Técnica de Darmstadt, en su proyecto para la creación de un método fácil y eficaz para ampliar los modelos de incrustación de frases existentes a nuevos idiomas (Reimers & Gurevych, 2004).

- **Bible-uedin**

  Este corpus tiene una cantidad de casi cien mil unidades de traducción provenientes de la Biblia, el cual, al ser un conjunto de libros tan influyente ha sido traducido a una gran cantidad de idiomas, y además gracias a su única estructura, en la que se identifica las oraciones con un número de libro, capítulo y versículo, facilita en gran medida la tarea de alineación (Christodouloupoulos & Steedman, 2014).

Como se puede observar se ha cogido una variedad suficiente, en términos de sintaxis, origen y temática, para evitar la sobre especialización del modelo de traducción.

3. Preproceso de extracción de valor

En el presente capítulo se analiza el repositorio GitHub creado por y para sacarle el máximo valor posible a los conjuntos de datos analizados anteriormente tanto en su versión monolingüe como bilingüe.

Así, cuando se habla de extracción de valor, lo que se entiende es dar consistencia y fiabilidad al conjunto de datos, ya que como hemos podido observar muchos de los datos que se van a usar no están revisados de forma manual por un traductor profesional, y por tanto están sujetos a problemas de alineación entre otros que no son tan previsibles, pero que definitivamente empeoran la calidad general del conjunto de datos lingüísticos.

Por último, hay que destacar que todo el código ha sido desarrollado con el objetivo de que sea lo más eficiente y claro posible, es por ello que se ha seguido la guía de estilo de Python PEP8.

3.1 Normalización

Los procesos de normalización son aquellos que buscan dar consistencia al conjunto de datos, además de ello también sientan las bases para la validación de estos. De tal forma se tiene:

- **get_text_with_normalized_quotes**

  Con esta función lo que se busca es sustituir las comillas del texto por las adecuadas según su idioma. Para más detalle, ver Figura 3.

- **get_text_with_normalized_spaces**

  Con esta función lo que se busca es eliminar cualquier conjunto de espacios repetidos y sustituirlo por uno solo. Además, también se eliminan los espacios antes y después del texto. Para más detalle, ver Figura 4.

- **get_text_with_normalized_unicode_characters**

  Esta función permite por un lado tener el texto en la forma “Normalización Descomposición Canónica”, esto permite tener en dos caracteres distintos las tildes (entre otros símbolos) de la vocal acentuada. Y, por otro lado, asegurar que todo el texto sigue el formato de codificación “UTF-8”. Para más detalle, ver Figura 5.

- **get_text_without_initial_index**

  Esta función permite eliminar las posibles modalidades de índices que en algunos conjuntos de datos puedan haber, así no hay que analizar esta característica conjunto por conjunto. Para más detalle, ver Figura 6.

- **get_text_without_repeated_symbols**

  Esta función permite eliminar conjuntos de símbolos que vayan seguidos sean diferentes o no, dejando solamente el primero de ellos. Para más detalle, ver Figura 7.

- **get_text_without_tags**

  Esta función permite eliminar todas las etiquetas provenientes de HTML y TMX, este último, es un formato muy común en la industria de las traducciones.
  Para más detalle, ver Figura 8.

Por último, cabe destacar que el código que se ha mostrado en las distintas Figuras es la lógica principal de cada uno de los normalizadores.
El código se puede analizar en el siguiente repositorio: https://github.com/dbeteta-w/linguistic_data_treatment/tree/main/processes/normalizers

3.2 Validación

Los procesos de validación son aquellos nos permiten distinguir qué partes del conjunto de datos lingüístico cumplen con unos requisitos mínimos de calidad. De tal forma, tenemos:

- **has_a_properly_amount_of_words**

  Esta función permite filtrar las unidades de traducción por su tamaño contado en palabras, así se puede delimitar el número mínimo y máximo de palabras según las necesidades del proyecto. Para más detalle, ver Figura 9.

- **has_parallel_number_val**

  Esta función permite comprobar que hay los mismos números en ambas partes de la unidad de traducción, independientemente de que estén escritos con letras o con números. Para más detalle, ver Figura 10.

- **has_parallel_symbols_val**

  Esta función permite comprobar que hay los mismos símbolos en ambas partes de la unidad de traducción, y al igual que la anterior también se le permite ajustar la tolerancia de errores. Para más detalle, ver Figura 11.

- **has_properly_length_factor_val**

  Esta función es especialmente importante a la hora de detectar malas traducciones o alineaciones, ya que lo que hace es comparar las longitudes relativas de “source” y “target” bajo un valor ajustable que se ha llamado “length_factor”, el cual, dependerá de la combinación de idiomas que se trate. Para más detalle, ver Figura 12.

- **has_too_many_numbers**

  Esta función permite descartar aquellos textos que contengan una cantidad de números relativamente alta frente a la cantidad de letras. De nuevo, la permisibilidad del validador puede ser ajustada, esta vez mediante el parámetro "alpha". Para más detalle, ver Figura 13.

- **is_in_the_accurate_language**

  Esta función juega también un papel importante, ya que, aunque a priori se pueda pensar que el conjunto de datos lingüístico va a estar en la combinación de idiomas que dice estar, esto no suele ser así, al menos, no en la totalidad de las unidades de traducción.

  Así, se ha usado el modelo de identificación de idiomas de "lid.176.bin", el cual, ha sido desarrollado por Facebook en su librería "Fasttext". Para más detalle, ver Figura 14.

- **is_repeated**

  Esta función nos permite eliminar textos repetidos de una forma estricta, esto es, comparándolos una vez se les ha eliminado la puntuación, los espacios y se ha pasado a minúsculas. Para más detalle, ver Figura 15.

  Así se puede ver en la función "get_text_to_be_compared" en la Figura 16.

Por último, cabe destacar que algunos validadores solo pueden tener versión bilingüe y que de los que tienen ambas versiones, el código mostrado es sobre la versión monolingüe, ya que es esta la que tiene la lógica principal.
El código se puede analizar en el siguiente repositorio: https://github.com/dbeteta-w/linguistic_data_treatment/tree/main/processes/validators

3.3 Pruebas unitarias

Una parte que también es muy importante del código es realizar pruebas unitarias de tanto normalizadores como validadores, ya que estos nos permiten pensar antes de escribir el código cuál queremos que sea el comportamiento deseado de las diferentes funciones, es decir, qué casos queremos que cubran. Luego durante la escritura del código, permiten realizar "refactors" sin comprometer la funcionalidad. Y después de escribir el código, sirven a otros programadores para entender en mayor profundidad el código escrito y qué se espera de él.

De tal forma, en este apartado se expone una prueba de un normalizador y otra de un validador:

- **test_get_text_without_repeated_symbols**

  Al tratarse de la prueba unitaria de un normalizador tanto el “input” como el “output” son cadenas de texto, luego el funcionamiento es comparar el resultado que se quiere con el “string” devuelto por la función, en este caso, de la función “get_text_without_repeated_symbols”. Para más detalle, ver Figura 17.

- **test_is_repeated**

  Al tratarse de la prueba unitaria de un validador el “input” es una cadena de texto, pero el "output" es un booleano, luego el funcionamiento es comparar el resultado que se quiere con el “True” o “False” devuelto por la función, en este caso, “is_repeated”. Para más detalle, ver Figura 18.

  Cabe destacar en este test que en la versión bilingüe del validador “is_repeated” no se considera como una repetición si una de las dos partes de la unidad de traducción es igual pero la otra es diferente. Esto es así, ya que se entiende que un texto con varias traducciones aporta valor.

3.4 Scripts principales

Se proponen dos scripts principales, uno para procesar los conjuntos de datos lingüísticos monolingües y otro para los bilingües, pero ambos comparten la misma lógica: agrupar por un lado los normalizador deseados en una función “normalize” y por otro lado agrupar los validadores en una función “validate”, la cual, además generará un informe almacenando qué unidades de traducción han sido descartadas y la función responsable de su descarte.

De igual forma, cabe destacar que como se trabaja con grandes cantidades de datos, los scripts incorporan el uso de paralelismo mediante la librería “multiprocessing”.
Así, el uso ideal que se le quiere dar a este par de scripts es dividir el conjunto de todos los datos en X partes iguales, siendo X la cantidad de procesadores que se puedan usar según las capacidades hardware.
Respecto a este aspecto cabe destacar dos puntos: el primero es la recomendación del comando “cat” para juntar los conjuntos de datos lingüísticos de los distintos orígenes en un solo archivo y luego el comando “split” con la opción “-l” para especificar el número de líneas. Y el segundo, respecto a cuántos procesadores usar, comprobar el número de CPUs mediante el comando “lscpu” y posteriormente usar un máximo aproximado de dos tercios del total, para favorecer el buen funcionamiento del ordenador o servidor que estemos utilizando.

También merece comentar el uso de la librería “argsparse”, gracias a la cual se ha podido crear una descripción detallada de uso mediante la opción “--help”, como se puede ver en la Figura 19.

De tal forma, se ve que el script requiere de seis argumentos obligatorios (ruta con los archivos a procesar, extensión de los archivos, código del primer idioma, código del segundo idioma, número de procesadores, ruta en la que se desea el resultado) y uno voluntario, en el cual, se pueden especificar los parámetros de restrictividad de los procesos de validación.

Un ejemplo de uso sería:

```
python get_bilingual_files_processed.py
-pi
/home/dbeteta/Documents/TFG/linguistic_data_treatment/origin_dat
a/bilingual_data/EN-ES/
-e .tsv
-sc EN
-tc ES
-c 10
-po
/home/dbeteta/Documents/TFG/linguistic_data_treatment/files_proc
essed/bilingual_files_processed/
-opt 2 25 -1 0.5 2.0 -1 0
```

Por último, insistir en la idea de que ambos scripts son propuestas de uso conjunto para la tarea que concierne al presente trabajo, pero el uso del repositorio puede llegar desde el uso individual de los procesos de limpieza hasta la ampliación de estos para sacar el máximo valor a otros conjuntos de datos en otras combinaciones de idiomas.

4. Clasificador de dominio

En este capítulo se detalla en primera instancia, tanto la teoría genérica detrás de los clasificadores, como la teoría específica de la arquitectura utilizada. Y en segunda instancia, todo el ciclo de desarrollo del clasificador de dominio "Health" para posteriormente poder usarlo a la hora de desarrollar el traductor automático especializado en dicha temática.

4.1 El problema de la clasificación

El problema de la clasificación tiene como objetivo principal predecir con la mayor precisión posible la clase correspondiente de un objeto (Godoy, 2021). Con tal finalidad, los clasificadores se pueden expresar mediante una función discriminante del siguiente tipo:

$$g_c: \mathrm{E} \rightarrow \mathrm{R} \mid 1 \leq \mathrm{c} \leq \mathrm{C}$$

Siendo **E** el espacio donde se representan los distintos objetos, **R** el "output" proporcionado por dicha función (un valor real) y **c** un valor comprendido entre 1 y la cantidad de funciones discriminantes de las que se dispone.

La representación de las muestras se alcanza mediante técnicas de preprocesado y extracción de propiedades, las cuales, serán usadas posteriormente para poder discriminar entre las distintas clases.

El número de clases o etiquetas vendrá definido o bien por los datos etiquetados (en caso de tratarse de aprendizaje supervisado) o bien por el propio modelo (en caso de tratarse de aprendizaje no supervisado).

Finalmente, a la hora de usar el modelo y clasificar, el funcionamiento es el siguiente: se obtiene el vector de características, a este se le aplica las distintas funciones discriminantes y en base a la función que devuelva un mayor valor, se le asigna la clase al objeto:

$$\hat{c} = G(y) = argmax\ g_c(y) \mid 1 \leq c \leq C$$

4.2 Arquitectura Fasttext

Se ha elegido el modelo "Fasttext" (Joulin, Grave, Bojanowski, & Mikolov, 2016), para esta tarea de clasificación de dominio, ya que es una herramienta desarrollada por Facebook, ahora Meta, la cual, tiene la misma precisión que los clasificadores basados en redes neuronales, pero necesita menor capacidad hardware además de ser mucho más rápido en términos de entrenamiento y testeo. Prueba de ello es que es capaz de entrenar más de un billón de palabras en menos de 10 minutos usando una CPU estándar.

Así, el modelo se basa fundamentalmente en representar las oraciones en "bags of words" (vector formado por la frecuencia de aparición de las distintas palabras) y el uso de un clasificador lineal, como puede ser "Máquina de Vectores de Apoyo" (SVM) o la regresión logística. Sin embargo, esta simple y eficiente solución, tiene desventajas que se van a ir solucionando hasta alcanzar el modelo "Fasttext".

De tal forma, los modelos lineales no comparten parámetros entre sus clases, lo cual, puede limitar la generalización de las clases con menos ejemplos del conjunto de datos de entrenamiento. Esto se consigue solucionar factorizando el clasificador lineal en matrices de bajo rango, luego el resultado es que las variables se incrustan y se promedian para crear la variable oculta y así ésta pueda ser reutilizada. Se puede ver gráficamente en la Figura 1.

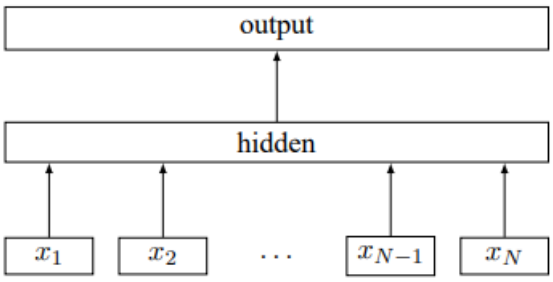

Figura 1: Esquema de modelo lineal con restricción de rango

De la misma manera, también se integra el uso de la función "softmax" para calcular la distribución de probabilidad sobre las clases que ya existen en ese momento, lo cual, nos lleva a minimizar la probabilidad negativa sobre las clases:

$$-1\frac{N}{X}\sum_{n=1}^{N} y_n \log\left((\mathrm{f}(\mathrm{B}\,\mathrm{A}x_n)\right)$$

Siendo $\mathbf{x_n}$ la bolsa normalizada de características de los diferentes datos de entrenamiento, $\mathbf{y_n}$ la etiqueta correspondiente y **A** y **B** las matrices de pesos.

Profundizando sobre la función jerárquica "softmax", hay que decir que su uso presenta ventajas tanto en el tiempo de entrenamiento como en el de prueba:

- Permite pasar en el tiempo de entrenamiento de una complejidad computacional de O(hk) a O(h log2 (k)), siendo k el número de clases y h la dimensión de la representación del texto.

- Permite pasar también en el tiempo de prueba a una complejidad computacional de O(h log2 (k)), esto es así ya que a cada nodo se le asigna la probabilidad del camino desde la raíz hasta dicho nodo, luego:

$$P(n_{l+1}) = \prod_{i=1}^{l} P(n_i)$$

  Estando el nodo en la profundidad "**l+1**" y siendo sus padres $\mathbf{n_1}$ **...,** $\mathbf{n_l}$.

> En otras palabras, lo que esto significa es que la probabilidad de un nodo es siempre menor a la de su padre, luego si usamos el método de búsqueda "DFS" se podrá descartar cualquier rama con una probabilidad de bajo valor, permitiendo así mejorar la complejidad computacional.

Por último, con respecto al uso de la bolsa de palabras decir que dicho método tiene la desventaja de que no se guarda el orden de las palabras, es por ello, que se usa la técnica de la bolsa de n-gramas junto al "hashing trick" para almacenar información parcial del orden de las palabras y así conseguir mejores resultados sin comprometer la eficiencia.

4.3 Obtención de datos etiquetados

Sin datos etiquetados no se puede construir ningún modelo de aprendizaje automático supervisado, es por ello que este apartado es tan importante. Conseguir dichos datos, además, es una tarea que si se hace manualmente es o bien demasiado tediosa si lo hace solo la persona que quiere desarrollar el clasificador o bien demasiado costosa económicamente si ha de contratar a personal externo, sobre todo si se quiere conseguir una cantidad suficiente de datos de entrenamiento.

Por todo ello, se expone la estrategia utilizada en el presente trabajo para conseguir datos etiquetados para el clasificador "Health":

1. Obtener datos monolingües en inglés (se escoge este idioma ya que tiene un léxico más constante que el español). Este punto ya se detalló en el capítulo 2 apartado 1 del presente trabajo.
2. Aplicar el script propuesto para la limpieza del conjunto de datos monolingües planteado en el capítulo 3 apartado 4, cuyo nombre es "get_monolingual_files_processed.py"
3. Obtener un glosario de tamaño razonable en el dominio seleccionado. Este punto ya se detalló en el capítulo 2 apartado 2.
4. Crear un script que filtre las oraciones en base a que algún término del glosario se encuentre en alguna oración del conjunto de datos monolingüe. De esta forma, se consigue tanto datos en el dominio deseado como datos de control para el clasificador.
   Asimismo, cabe destacar que se usa una función que permite comprobar si es un "match" exacto, esto es que antes y después de dicho término haya un espacio o un símbolo, así se evitan falsos positivos que ocurrían al usar simplemente la cláusula de Python "in".

Por un lado, el resultado tras ejecutar la limpieza correspondiente al paso número 2 para conseguir datos etiquetados, fue pasar de 15 millones de líneas a 14,2 millones de líneas, empleando algo menos de 19 minutos y habiéndose generado el siguiente informe debido a los descartes de los validadores:

Proceso	Cantidad descartada	Porcentaje frente al total
has_text_properly_amount_of_words	709.155	89,56%
has_text_too_many_numbers	11.390	1,43%
is_text_in_the_accurate_language	48.327	6,1%
is_text_repeated	22.917	2,91%

Tabla 1: Resultados validación datos monolingües

Por otro lado, el resultado tras ejecutar el script correspondiente al paso número 4 para conseguir datos etiquetados, se obtuvieron de los 14,2 millones de datos limpios alrededor de 400 mil oraciones que contuvieran términos del glosario "Health".

Por último, hay que decir que, aunque esta estrategia no está libre de falsos positivos (puede haber oraciones que contengan términos "Health" pero no hablen realmente sobre esa temática), sí que nos permite de forma sencilla, rápida y a coste zero obtener datos de entrenamiento como los requiere el modelo Fasttext:

__label__HLT The doctor needed more time to make an accurate diagnosis

Siendo HLT la etiqueta designada para el domino "Health" y GEN para el resto de las oraciones que no se identifican en el dominio a detectar.

4.4 Entrenamiento y testeo del clasificador

Una vez obtenidos los datos etiquetados el siguiente paso es balancearlos y separar dicho conjunto de datos en la parte de entrenamiento y de testeo.

Así, se cogieron alrededor de otras 400 mil oraciones de las que no contuvieran ningún elemento del glosario, y se juntaron y barajaron dentro del mismo archivo formado ahora por alrededor de 800 mil oraciones. Tras ello, se separaron los datos en 2 archivos: "training_data.txt" formado por el 90% del conjunto de datos y "test_data.txt" con el resto.

El siguiente paso fue entrenar el modelo de clasificación binaria donde hay dos posibles clases: o la oración habla sobre el dominio "Health" (HLT) o es una oración "genérica" (GEN). Así, para entrenar el modelo solo fue necesario el uso de la función "train_supervised", cuyos argumentos especificados fueron: el conjunto de datos de entrenamiento, el valor de la tasa de aprendizaje o "learning rate" (1.0), el número de épocas (5) y el uso de n-gramas a 2, es decir, bigramas.

Con tal configuración, y en solo unos pocos minutos de entrenamiento, se alcanzó una precisión, esto es, el porcentaje total de elementos clasificados correctamente, del 98,53%. Luego, sin lugar a duda, se puede decir que se trata de un clasificador binario en inglés fiable.

5. Traductor automático neuronal

En este capítulo se detalla en primer lugar, tanto la teoría genérica detrás de los traductores automáticos, como la teoría específica de la arquitectura utilizada. Y en segunda instancia, todo el ciclo de desarrollo de las dos máquinas de traducción (una con los datos limpiados y clasificados, y otra sin ninguno de dichos procedimientos) para posteriormente poder testear los resultados de ambas y probar cuantitativamente los efectos de los nombrados procedimientos.

5.1 El problema de la traducción automática

El problema de la traducción automática se engloba dentro del ámbito del reconocimiento de patrones, más concretamente, en el ámbito de lo que se conoce como el "procesamiento del lenguaje natural" (PLN).

Así, dada una oración en un determinado idioma el objetivo de la traducción automática es transformar dicho "input" en otra oración en otro idioma, conservando el mayor significado posible de la original, en otras palabras, dar como "output" aquella oración más probable de ser la mejor traducción.

De tal forma, para que el modelo de traducción automática pueda aprender a discriminar cuál es la oración más probable, necesita de corpus alineados. Esto es un conjunto de datos lingüísticos en los que, o bien "source" (oración en el idioma a ser traducido) y "target" (oración en el idioma que se quiere la traducción) están separados por algún separador como una coma o una tabulación (formato ".csv" y ".tsv" respectivamente), o bien dicho "source" y "target" se hayan diferenciado por un sistema de etiquetas como puede ser en el formato XML. En ambos casos, se generará un sistema en una sola dirección, es decir, si entrenamos una máquina para que traduzca de inglés a español, esta misma máquina no sabrá traducir de español a inglés, para ello se necesitaría entrenar otra distinta.

Entrando más en detalle, históricamente se han usado 3 tipos de modelos en la traducción automática:

1. Modelos basados en palabras: esta clase de modelos trabajan bajo la hipótesis de que la traducción se realiza de cada una de las palabras individualmente (Brown, y otros, 1990). En concreto, se trabaja en base a 2 conceptos:
    - Modelo de traducción léxica, cuya función es devolver la probabilidad de cada una de las traducciones posibles para una determinada palabra.
    - Modelo de reordenamiento o alineación, cuya función es indicar qué tan probable es que las frases traducidas estén ordenadas de dicha manera.

Matemáticamente se puede expresar como:

$$p(x|y) = \sum_{a} p(x, a|y)$$

Siendo **y** el "source", **x** la traducción y **a** la alineación.

2. Modelos basados en oraciones: esta clase de modelos trabajan descomponiendo las oraciones en conjuntos de palabras o frases y traduciéndolos hasta recomponer finalmente la oración completa (Koehn, Josef Och, & Marcu, 2003). Esta forma de realizar la traducción automática supone una gran mejora frente a los modelos basados en palabras ya que al traducir por conjunto de palabras se tiene más información contextual que palabra por palabra.

   Este modelo además está compuesto en base a 3 conceptos:
   - Modelo de lenguaje, cuya función es estimar la probabilidad a priori de que una frase se produzca en un idioma determinado.
   - Modelo de traducción, el cual viene definido matemáticamente como:

     $$p_t(\bar{x}_1^I|\bar{y}_1^I) = \prod_{i=1}^{I} \varphi(\bar{x}_i|\bar{y}_i)$$

     Siendo $\boldsymbol{\varphi(x_i\ |y_i)}$ una tabla de traducciones que permiten calificar la calidad de una traducción.
   - Modelo de reordenamiento, cuya función ya ha sido explicada en modelo basado en palabras.

3. Modelos basados en redes neuronales: esta clase de modelos se basan en el uso de redes neuronales, más concretamente, se basan en el perceptrón multicapa, el cual, está formado por una capa de entrada, un número variable de capas ocultas y una capa de salida (Goodfellow, Bengio, & Courville, 2016). Pero, además, al tratarse de una tarea donde hay dependencias entre las palabras de la oración "input", es decir, al tratarse de datos secuenciales el modelo que mejor se adapta a dicha tarea es la red neuronal recurrente. Así, la ecuación para las capas ocultas es:

   $$h_t^{(i)} = g^{(i)}(W^{(i)}h_t^{(i-1)} + U^{(i)}h_{t-1}^{(i)} + b^{(i)})$$

   Siendo $\mathbf{W^{(i)}}$ la matriz de pesos de la capa, $\mathbf{b^{(i)}}$ el sesgo de la capa, $\mathbf{g^{(i)}}$ la función de activación de la capa y, por último, y principal diferencia con el perceptrón multicapa anteriormente mencionado, se tiene $\mathbf{U^{(i)}}$ que es la suma de la matriz de pesos con las que se controla el estado anterior al actual.

   Otro factor a tener en cuenta es el "problema de explosión de gradiente", esto es, las dependencias de largo alcance, es por ello que se usan también unidades "Long short-term memory", lo cual significa que la capa oculta depende además de $h^{(i-1)}$ en el momento t y de $h^{(i)}$ en el momento t-1, del estado interno c_t en el que se almacena información entre cada etapa.

5.2 Arquitectura Transformer

Se ha elegido la arquitectura Transformers (Vaswani, y otros, 2017) ya que forma parte del estado del arte actual y por tanto va a permitir extraer los mejores resultados posibles para la tarea que se va a realizar.

De tal forma, se tiene que dicha arquitectura sigue la estructura "encoder-decoder", en la que el codificador asigna una secuencia de entrada de representaciones de símbolos (x1, ..., xn) a una secuencia de representaciones continuas z = (z_1, ..., z_n). Dado z, el decodificador genera una salida secuencia (y_1, ..., y_m) de símbolos elemento por elemento. Y como el modelo es autorregresivo, en cada paso que se da, se va retroalimentando dicho modelo con los símbolos generados previamente como entrada.

Además, esta arquitectura se completa utilizando autoatención apilada y capas totalmente conectadas tanto para el codificador como para el decodificador. Se puede ver una representación gráfica en la Figura 2.

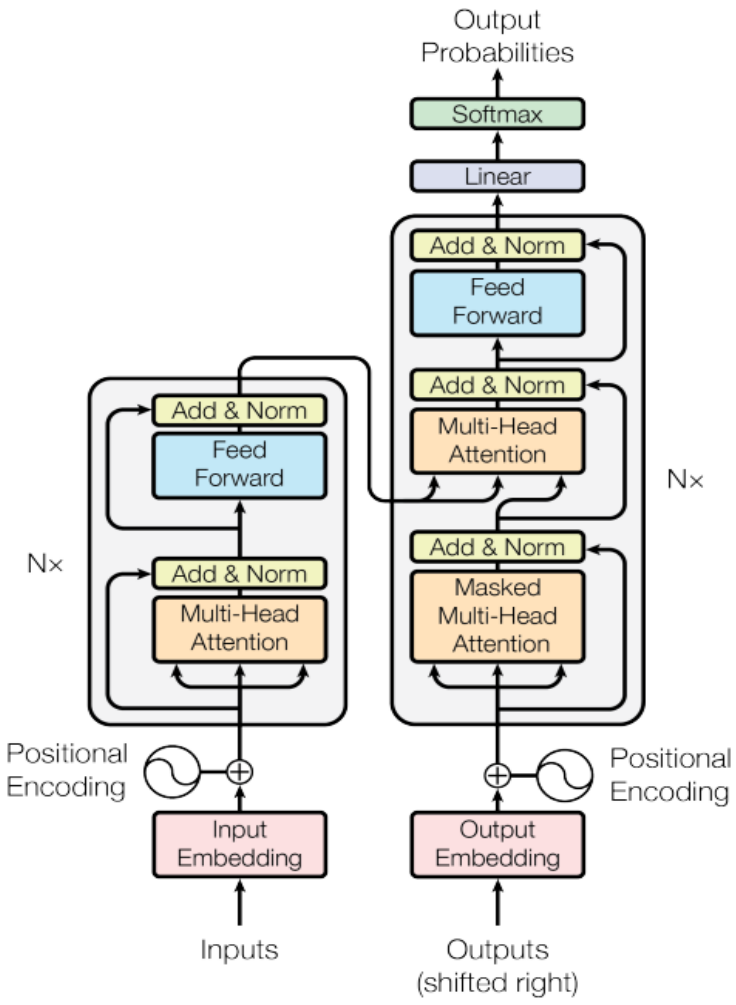

Figura 2: Arquitectura Transformer

Así, por una parte, el codificador está formado por 6 capas idénticas, las cuales, están subdividas en 2: una con el mecanismo de autoatención de múltiples cabezas y la otra con una red de realimentación por posición. Además, cada una de las subcapas a su vez usa conexiones residuales seguidas de un proceso de normalización de capas. Por otra parte, el decodificador comparte estructura con el codificador, con 2 diferencias: la primera la tenemos en la cantidad de subcapas, ya que el decodificador cuenta con una extra cuya función es la atención de múltiples cabezas en los resultados de la pila de los resultados el codificador; y la segunda está en la modificación de la subcapa de autoatención de forma que se asegure que solo se pueden realizar traducciones en base a elementos conocidos anteriormente.

Pasando a describir lo que se entiende por atención, esta se puede definir como la asignación de una consulta y un conjunto de pares clave-valor a una salida, siendo cada uno de estos elementos un vector. De tal forma, y poniendo el foco en los mecanismos de las subcapas descritas:

- Autoatención de múltiples cabezas, su función es permitir al modelo atender conjuntamente la información de diferentes subespacios de representación en diferentes posiciones.
- Red de realimentación por posición, su función consiste en dos transformaciones lineales con una activación de un rectificador en medio de ambas. Cabe añadir que para cada transformación se usan parámetros distintos en función de cada capa.

También cabe destacar que a diferencia de lo descrito en el capítulo 5 apartado 1 del presente trabajo, la arquitectura Transformers no usa redes neuronales recurrentes (ni convolucionales), luego para almacenar información sobre el orden del texto, se usa "positional encodings", los cuales, viene definidos por las siguientes funciones:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

Siendo **i** la dimensión y **pos** la posición.

Por último, y en virtud de la motivación de usar la autoatención frente a las redes neuronales recurrentes se tiene que:

- Se rebaja la complejidad computacional por capa de O(n d^2) a O(n^2 d), siendo n la longitud del texto y d la representación de la dimensión.
- Se rebaja la complejidad de las operaciones secuenciales de O(n) a O(1), permitiendo así una mayor capacidad de paralelización dado su coste constante en operaciones secuenciales.
- Se rebaja la longitud máxima de la ruta de O(n) a O(1), facilitando así el aprendizaje de dependencias de largo alcance.

5.3 Obtención de datos de entrenamiento

Como bien se ha presentado en el primer apartado del presente capítulo, para entrenar cualquier máquina de traducción se necesitan corpus bilingües. Por tanto, la estrategia seguida para conseguir dichos datos ha sido:

1. Descargar datos bilingües, en este caso, alineaciones de inglés con el español. Este punto ya se detalló en el capítulo 2 apartado 3 del presente trabajo.
2. Aplicar el script propuesto para la limpieza del conjunto de datos bilingües planteado en el capítulo 3 apartado 4, cuyo nombre es "get_bilingual_files_processed.py".
3. Aplicar a los datos ya limpiados el clasificador desarrollado en el capítulo 4 para finalmente conseguir datos bilingües en el dominio "Health".
4. Como se quiere desarrollar además una segunda máquina de traducción con los datos sin limpiar ni clasificar, se "balancean" los datos de forma que la cantidad para ambas máquinas sea el mismo, es decir, lo que se ha hecho es ir al conjunto de datos bilingües iniciales y repartir de forma directamente proporcional los datos según el origen y teniendo en cuenta la cantidad obtenida en el punto 3.

Por un lado, el resultado tras ejecutar la limpieza correspondiente al paso número 2, fue pasar de 41 millones de unidades de traducción a 18,3 millones, empleando alrededor de 50 minutos y habiéndose generado el siguiente informe debido a los descartes de los validadores:

Proceso	Cantidad descartada	Porcentaje frente al total
has_a_properly_amount_of_words	12.580.765	53,98%
has_too_many_numbers	1.118.991	4,8%
is_in_the_accurate_language	3.884.427	16,66%
has_properly_length_factor_val	798.740	3,42%
has_parallel_symbols_val	93.187	0,4%
has_parallel_number_val	1.395.259	5,98%
is_repeated	3.432.148	14,76%

Tabla 2: Resultados validación datos bilingües

Por otro lado, tras realizar la clasificación con una probabilidad mínima del 75% para ser aceptado como unidad de traducción valida en el dominio "Health" se obtienen alrededor de medio millón de unidades de traducción, las cuales, han sido repartidas en 3 archivos distintos: "train_HLT_en-es.tsv" con el 90% de los datos, "check-learning_HLT_en-es.tsv" con otro 5% de los datos y "test_HLT_en-es.tsv" con el restante 5%.

Por último, siguiendo con el procedimiento del paso 4, se generan 2 archivos más, esta vez, para el entrenamiento de la máquina de traducción con los datos sin limpiar ni clasificar: "train_raw_en-es.tsv" y "check-learning_raw_en-es.tsv".

5.4 Entrenamiento del traductor automático neural

Lo primero que se ha de destacar es que se usa el Transformer pre entrenado en 101 idiomas por parte de "Google Research" llamado "mt5", sin embargo, dicho modelo ha sido entrenado de forma no supervisada con el conjunto de datos "mC4", esto quiere decir que dicho modelo ha de ajustarse y especializarse en una determinada tarea para que sea funcional (Xue, y otros, 2021).

Dicho modelo, más concretamente su versión "mt5-base", permite partir de una cierta base para las máquinas de traducción que vamos a crear y también permite tener un "tokenizador", al cual le vamos a añadir la etiqueta "<es>" ya que este es el idioma al que queremos traducir.

El siguiente paso consiste en cargar los datos, y crear un generador de "batches" en los que tanto "source" como "target" se codifican con la única diferencia que al "source" se le añade previamente a la codificación la etiqueta "<es>".

En cuanto a este paso cabe destacar, por un lado, que si no se crearan los denominados "batches" y se intentara cargar todos los datos de golpe, la RAM se llenaría y por tanto el propio sistema operativo mataría este proceso. Y, por otra parte, respecto a la codificación, hay que detallar que a lo que se refiere es a convertir los distintos "tokens" encontrados por el "tokenizador" en un número y formar así un vector de las mismas dimensiones para todo el conjunto de datos.

Posteriormente, se han de definir los parámetros del entrenamiento:

- Nº de épocas: 8 (más épocas produce, en nuestro caso, sobre especialización).
- Tamaño del "batch": 8 (un mayor número genera problemas de RAM en el entorno de Google Colab Pro).
- Tasa de aprendizaje: 5e-4 (experimentalmente es el que mejores resultados ha traído).
- Número de pasos de calentamiento: el 1% del total de pasos (número de "batches" multiplicado por el número de épocas).

  Hay que recalcar el uso del calentamiento para la tasa de aprendizaje, ya que este mecanismo lo que permite es mantener de forma más estable el aprendizaje del modelo.

- Optimizador: se usa "AdamW", el cual, es dado por "Hugging Face" y usa la regularización de la descomposición del peso para reducir de esta manera la posibilidad de sobre entrenamiento.

Ahora, una vez cargados los datos tanto de entrenamiento como de chequeo de aprendizaje y definido los parámetros del entrenamiento, es momento de iterar sobre los distintos "batches" e ir actualizando la matriz de pesos del modelo de forma que aprenda y se especialice en la tarea que se quiere realizar.

De igual forma, y cada cierta frecuencia se ejecutarán dos operaciones: la primera una pequeña prueba que permite comprobar la evolución del aprendizaje modelo y la segunda el guardado del actual estado del modelo.

En la práctica, haciendo uso de la GPU de Google Colab Pro, este proceso supuso alrededor de unas 18 horas de entrenamiento para cada máquina de traducción.

Por último, hay que destacar que para evitar que el proceso entrara en modo inactivo y se parase el entrenamiento se ejecutó el código de la Figura 20 en la consola del navegador.

5.5 Testeo del traductor automático neuronal

Igual que ocurría en el capítulo 4 apartado 3 del presente trabajo, la forma más fiable para realizar esta clase de tarea es usar la revisión humana, sin embargo, por los problemas ya descritos en dicho capítulo, hay que buscar alternativas más eficientes.

Es por ello, que la métrica que se ha usado para comprobar la calidad de las traducciones de ambas máquinas es el BLEU, cuyo valor oscila entre 0 y 1, siendo contra más alto reflejo de traducciones más fiables (Doshi, 2021).

Dicha métrica se calcula por un lado con la media geométrica de la precisión:

$$Media\ geométrica\ de\ la\ presición\ (N) = exp\ (\sum_{n=1}^{N} w_n\ log\ p_n)$$

Siendo $\mathbf{w_n}$ el peso asignado y $\mathbf{p_n}$ la precisión.

Y, por otro lado, también se incluye una penalización por brevedad:

$$Penalización\ por\ bevedad = \begin{cases} 1 & : si\ c\ >\ \mathrm{r} \\ e^{(1-r/c)} & :\ si\ c \leq\ \mathrm{r} \end{cases}$$

Siendo **c** la longitud del "output" del traductor y **r** la longitud del "target" o referencia.

Dando como resultado que el BLEU se calcule:

$$BLEU(N) = Penalización\ por\ brevedad * Media\ geométrica\ de\ la\ precisión$$

Así, dicha métrica puede utilizarse con varios valores para N:

- BLEU-1: Se calcula la precisión a nivel de uni-grama.
- BLUE-2: Se calcula la media geométrica de la precisión de uni-grama y bi-grama.
- BLUE-3: Se calcula la media geométrica de la precisión de uni-grama, bi-grama y tri-grama.
- BLEU-4: Se calcula la media geométrica de la precisión de uni-grama, bi-grama, tri-grama y cuatri-grama.

También, merece la pena expresar que la métrica BLEU tiene también defectos, entre ellos, destaca la ausencia de mecanismos para detectar sinónimos o expresiones equivalentes y la ausencia de control de la importancia de las palabras, no es lo mismo que falta un sustantivo clave para el significado de la oración que falte un artículo.

Por todo ello, es por lo que se considera que un BLEU-4 de entre 0.6 y 0.7 es lo más alto a lo que se puede aspirar.

Una vez definida la métrica a utilizar, solo hace falta aplicarla a ambas máquinas de traducción con el archivo "test_HLT_en-es.tsv" que creamos en el apartado 3 de este capítulo.

De tal forma, y gracias a la librería NLTK, se obtienen los siguientes resultados:

Métrica	**HLT_mt5_translation.h5**	**Raw_mt5_translation.h5**
BLEU-1	0.702489	0.355973
BLEU-2	0.615270	0.252762
BLEU-3	0.582082	0.218852
BLEU-4	0.492037	0.139069

Tabla 3: Resultados BLEU de ambas máquinas de traducción automática

Luego si comparamos dichos valores con la indicación que se encuentra en la cita (Lavie, 2011) se ve cómo el modelo "HLT_mt5_translation.h5" se encuentra en el rango del BLEU-4 de entre 0.4 y 0.5 "Traducciones de alta calidad", mientras que el modelo "Raw_mt5_translation.h5" se encuentra en el rango de entre 0.1 y 0.19 "Difícil de captar la esencia".

6. Conclusión

En conclusión, se puede afirmar con rotundidad que los objetivos del presente Trabajo de Final de Grado han sido cumplidos.

Esto es así, ya que se ha desarrollado un repositorio Github con un sistema de extracción de valor de conjuntos de datos lingüísticos (https://github.com/dbeteta-w/linguistic_data_treatment) y un clasificador de dominio para la temática "Health" con una precisión del 98,53%, que conjuntamente han permitido tratar "datasets" públicos para crear una máquina de traducción automática neuronal con un BLEU-4 del 0.49, cifra significativamente mayor frente al traductor automático neuronal creado sin ninguno de los procedimientos anteriormente mencionados, ya que su BLEU es del 0.13.

Por lo tanto, queda demostrado cuantitativamente el efecto de una correcta limpieza y clasificación de los datos previa a la construcción de una máquina de traducción especializada en un dominio.

6.1 Relación del trabajo desarrollado con los estudios cursados

Aunque en el presente trabajo se ha ampliado profundamente en los conceptos y tecnologías utilizadas con respecto a las asignaturas impartidas en el grado, cabe mencionar las siguientes asignaturas como fundamento a partir del cual he trabajado:

- Estructura de datos y algoritmos
- Fundamentos de sistemas operativos
- Computación paralela
- Sistemas inteligentes
- Sistemas de información estratégicos

Aprovecho también está sección para agradecer a la ETSINF por un lado por su plataforma PEIX para encontrar prácticas en empresa y por otro lado a su sistema de dispensas que me ha permitido compaginar estudiar y trabajar a la vez, eso sí con mucho esfuerzo y sacrificio. Sin ese primer empujón, muy probablemente, no hubiera sido posible este TFG.

Anexo

Código

```
def get_text_with_normalized_quotes(text: str, lang_desired: str) -> str:
    rest_quotes = _get_rest_quotes(lang_desired)
    amount_quotes, text = _get_amount_quotes_and_replace_wildcard(rest_quotes, text)
    return _replace_wildcard_with_proper_quotes(amount_quotes, text, lang_desired)
```

Figura 3: Normalizador de comillas

```
def get_text_with_normalized_spaces(text: str) -> str:
    return re.sub(r"\s+", " ", text).strip()
```

Figura 4: Normalizador de espacios

```
def get_text_with_normalized_unicode_characters(text: str) -> str:
    # NFD form (Normalization Form Canonical Decomposition) => accented letters = 2 character
    # For more info: https://en.wikipedia.org/wiki/Unicode_equivalence
    return unicodedata.normalize("NFD", text).encode("ascii", "ignore").decode("UTF-8")
```

Figura 5: Normalizador de caracteres unicode

```
def get_text_without_initial_index(text: str) -> str:
    not_number_found = re.findall(r"[^\d|,\t. ]", text)
    if not_number_found:
        index_of_first_non_number = text.index(not_number_found[0])
        return re.sub(r"\d+[|,\t. ]?", "", text[:index_of_first_non_number]) + text[index_of_first_non_number:]
    else:
        return text
```

Figura 6: Normalizador de índices iniciales

```
def get_text_without_repeated_symbols(text: str) -> str:
    symbols_found = _get_symbols(text)
    indexes_symbols_found = _get_indexes_of_symbols(text, symbols_found)
    subset_of_indexes_followed = _get_subset_of_indexes_followed(indexes_symbols_found)
    return _remove_repeated_symbols(text, subset_of_indexes_followed)
```

Figura 7: Normalizador de símbolos repetidos

```python
def get_text_without_tags(text: str) -> str:
    # Remove html tags, tmx tags among other
    return re.sub(r"([<{]\\?/?[A-Za-z\d]+/?[>}]|[&%][A-Za-z\d]+;?)", "", text)
```

Figura 8: Normalizador de etiquetas

```python
def has_text_properly_amount_of_words(text: str, min_words=2, max_words=35) -> bool:
    list_of_cleaning_processes = [
        get_text_without_punctuation,
        get_text_with_normalized_spaces
    ]

    for funct in list_of_cleaning_processes:
        text = apply(funct, text)

    list_of_words = text.split(" ")
    len_list_of_words = len(list_of_words)
    if len_list_of_words > max_words or len_list_of_words < min_words:
        return False
    return True
```

Figura 9: Validador por cantidad de palabras

```python
def has_parallel_number_val(src: str, src_lang: str, tgt: str, tgt_lang: str, tolerance=0) -> bool:
    src_alpha2digit = alpha2digit(src, src_lang)
    tgt_alpha2digit = alpha2digit(tgt, tgt_lang)
    for number in range(10):
        string_number = str(number)
        if abs(src.count(string_number) - tgt.count(string_number)) > tolerance and \
                abs(src_alpha2digit.count(string_number) - tgt_alpha2digit.count(string_number)) > tolerance:
            return False
    return True
```

Figura 10: Validador por números paralelos

```python
def has_parallel_symbols_val(src: str, tgt: str, tolerance=0, symbols_to_check=None) -> bool:
    if symbols_to_check is None:
        symbols_to_check = {'[', ']', '{', '}', '<', '>', '@', '+', '...', '#'}

    for symbol in symbols_to_check:
        if abs(src.count(symbol) - tgt.count(symbol)) > tolerance:
            return False
    return True
```

Figura 11: Validador por símbolos paralelos

```
def has_properly_length_factor_val(src: str, tgt: str, length_factor=2.0, min_len=6) -> bool:
    len_src_to_be_compared = len(get_text_to_be_compared(src))
    len_tgt_to_be_compared = len(get_text_to_be_compared(tgt))

    if len_src_to_be_compared < min_len and len_tgt_to_be_compared < min_len:
        return True
    if len_src_to_be_compared > len_tgt_to_be_compared * length_factor \
            or len_tgt_to_be_compared > len_src_to_be_compared * length_factor:
        return False
    return True
```

Figura 12: Validador por longitud relativa

```
def has_text_too_many_numbers(text: str, alpha=2) -> bool:
    list_of_numbers = re.findall(r"\d", text)
    list_of_letters = re.findall(r"[A-Za-z]", text)
    if len(list_of_numbers) * alpha >= len(list_of_letters):
        return True
    return False
```

Figura 13: Validador por cantidad de números

```
def is_text_in_the_accurate_language(text: str, lang: str, fasttext_model, minimum_probability=0.4) -> bool:
    AMOUNT_OF_GUESSES = 2
    LABEL_TYPO = "__label__"
    text_with_normalized_spaces = get_text_with_normalized_spaces(text)

    guess_array = fasttext_model.predict(text_with_normalized_spaces, k=AMOUNT_OF_GUESSES)
    if len(guess_array) >= AMOUNT_OF_GUESSES \
            and len(guess_array[0]) >= AMOUNT_OF_GUESSES \
            and len(guess_array[1]) >= AMOUNT_OF_GUESSES:
        first_lang_guess = guess_array[0][0].replace(LABEL_TYPO, "")
        first_lang_prob = guess_array[1][0]
        if first_lang_guess != lang or \
                (first_lang_guess == lang and
                 first_lang_prob < minimum_probability):
            second_lang_guess = guess_array[0][1].replace(LABEL_TYPO, "")
            second_lang_prob = guess_array[1][1]
            if second_lang_guess != lang or \
                    (second_lang_guess == lang and
                     second_lang_prob < minimum_probability):
                return False
    return True
```

Figura 14: Validador por idioma detectado

```python
def is_text_repeated(text: str, set_texts: Set[str]):
    text_to_be_compared = get_text_to_be_compared(text)
    if text_to_be_compared in set_texts:
        return True
    return False
```

Figura 15: Validador por repeticiones estrictas

```python
def get_text_to_be_compared(text: str) -> str:
    list_of_cleaning_processes = [
        get_text_without_punctuation,
        lambda x: x.casefold(),
        lambda x: re.sub(r"\s", "", x),
    ]

    for funct in list_of_cleaning_processes:
        text = apply(funct, text)

    return text
```

Figura 16: Función de limpieza auxiliar

```python
class TestGetTextWithoutRepeatedSymbols(unittest.TestCase):
    def test_normalizer(self):
        test_golds = {
            "": "",
            "Hola que tal": "Hola que tal",
            "Hola,, que tal": "Hola, que tal",
            "Hola que tal..": "Hola que tal.",
            "Hola que tal;.": "Hola que tal;",
            "Hola,:, que tal": "Hola, que tal",
            "Hola,, que tal??": "Hola, que tal?"
        }

        for test, gold in test_golds.items():
            test_without_repeated_symbols = get_text_without_repeated_symbols(test)
            self.assertEqual(test_without_repeated_symbols, gold)
```

Figura 17: Test normalizador de símbolos repetidos

```python
class TestIsRepeated(unittest.TestCase):
    def test_validator(self):
        test_set = {
            tuple(["holaquetal", "hihowareyou"])
        }

        test_golds = {
            tuple(["Hola que tal", "Hi how are you doing"]): False,
            tuple(["Hola que tal", "Hi how are you"]): True,
            tuple(["Hola, que tal?", "Hi, how are you?"]): True,
            tuple(["HOLA QUE TAL", "HI HOW ARE YOU"]): True,
            tuple([" Hola que     tal", " Hi how    are you"]): True
        }

        for test, gold in test_golds.items():
            src, tgt = test
            self.assertEqual(is_repeated(src, tgt, test_set), gold)
```

Figura 18: Test validador por repeticiones estrictas

```
(venv) (base) dbeteta@dbeteta-OMEN-by-HP-Laptop:~/Documents/TFG/linguistic_data_treatment$ python get_bilingual_files_processed.py --help
usage: get_bilingual_files_processed.py [-h] -pi INPUT -e {.tsv,.csv} -sc
                                        SRCCODE -tc TGTCODE -c CPUS -po OUTPUT
                                        [-opt MIN_WORDS MAX_WORDS ALPHA MIN_PROB LENGTH_FACTOR TOLERANCE_SYMBOLS TOLERANCE_NUMBERS]

Normalize and validate bilingual files

optional arguments:
  -h, --help            show this help message and exit
  -pi INPUT, --input INPUT
                        Path to the input files
  -e {.tsv,.csv}, --extension {.tsv,.csv}
                        Extension of the files
  -sc SRCCODE, --srccode SRCCODE
                        Source language ISO 639-1 Code => en
  -tc TGTCODE, --tgtcode TGTCODE
                        Target language ISO 639-1 Code => es
  -c CPUS, --cpus CPUS  Amount of cpus desired to use in the execution.
                        Recommendation: use a max of 2/3 of the total
  -po OUTPUT, --output OUTPUT
                        Path to the desired output folder
  -opt MIN_WORDS MAX_WORDS ALPHA MIN_PROB LENGTH_FACTOR TOLERANCE_SYMBOLS TOLERANCE_NUMBERS, --optional MIN_WORDS MAX_WORDS ALPHA MIN_PROB LENGTH_FACTOR TOLERANCE_SYMBOLS TOLERANCE_NUMBERS
                        Introduce the seven values like => 2 35 2 0.4 2.0 0 0
                        1. Minimum words => has_a_properly_amount_of_words -
                        2. Maximum words => has_a_properly_amount_of_words -
                        3. Alpha value => ------ has_too_many_numbers ------
                        4. Min probability => - is_in_the_accurate_language -
                        5. Length factor => - has_properly_length_factor_val
                        6. Tolerance wrong symbol => has_parallel_symbols_val
                        7. Tolerance wrong number => has_parallel_number_val
                        If you want the default value just introduce -1 in the
                        desired variable => 2 35 -1 0.5 2.0 -1 0
```

Figura 19: Ayuda para el uso de “get_bilingual_files_processed.py”

```javascript
var rand = 90000;

function ClickConnect() {
console.log("Working");
rand = Math.round(Math.random()*(90000-30000))+30000;
document
  .querySelector('#top-toolbar > colab-connect-button')
  .shadowRoot.querySelector('#connect')
  .click()
}
setInterval(ClickConnect, rand)
```

Figura 20: Script para evitar la inactividad de Google Colab

Objetivos de desarrollo sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**			X	
ODS 9. **Industria, innovación e infraestructuras.**				X
ODS 10. **Reducción de las desigualdades**.			X	
ODS 11. **Ciudades y comunidades sostenibles**.				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

El presente TFG no presente excesiva relación con ningún objetivo ODS, sin embargo, con los objetivos que más podríamos encajarlo serían el ODS 8 “Trabajo decente y crecimiento económico” y con el ODS 10 “Reducción de las desigualdades”. Esto es así, ya que crear un repositorio open-source para la extracción de valor de conjuntos de datos lingüísticos permite por un lado apalancarse en él, evitando reinventar el fuego, y así dedicar esos esfuerzos en otros ámbitos, favoreciendo la innovación y por tanto el crecimiento económico. Y, por otro lado, la creación de dicho repositorio permite que cualquier persona independientemente de su renta pueda desarrollar proyectos (lucrativos o no) en base a un código fiable y en el que se ha comprobado su utilidad mediante el actual trabajo final de grado.

Bibliografía

BeautifulSoup. (s.f.). Obtenido de https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Brown, P. F., Cocke, J., A Della Pietra, S., J Della Pietra, V., Jelinek, F., D Lafferty, J., . . . S Roossin, P. (1990). A statistical approach to machine translation. Obtenido de https://aclanthology.org/J90-2002.pdf

Christodouloupoulos, C., & Steedman, M. (2014). A massively parallel corpus: the Bible in 100 languages. Obtenido de https://link.springer.com/content/pdf/10.1007/s10579-014-9287-y.pdf

Doshi, K. (2021). *Medium*. Obtenido de https://towardsdatascience.com/foundations-of-nlp-explained-bleu-score-and-wer-metrics-1a5ba06d812b#:~:text=It%20is%20based%20on%20the,the%20best%20you%20can%20achieve.

Godoy, F. E. (2021). Metodos clásicos de clasificación: comparación y aplicación. Obtenido de https://rdu.unc.edu.ar/bitstream/handle/11086/19768/Godoy%2C%20F.%20E.%20M%C3%A9todos%20cl%C3%A1sicos%20de%20clasificaci%C3%B3n.pdf?sequence=1&isAllowed=y

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning textbook. Obtenido de https://www.deeplearningbook.org/

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of Tricks for Efficient Text Classification. Obtenido de https://arxiv.org/pdf/1607.01759.pdf

Koehn, P., Josef Och, F., & Marcu, D. (2003). Statistical Phrase-Based Translation. Obtenido de https://aclanthology.org/N03-1017.pdf

Lavie, A. (2011). *Evaluating the Output of Machine Translation Systems*. Obtenido de https://www.cs.cmu.edu/~alavie/Presentations/MT-Evaluation-MT-Summit-Tutorial-19Sep11.pdf

ProZ. (s.f.). Obtenido de https://www.proz.com/

Reimers, N., & Gurevych, I. (2004). Making Monolingual Sentence Embeddings Multilingual using. Obtenido de https://arxiv.org/pdf/2004.09813.pdf

Rozis, R., & Skadiņš, R. (2017). Tilde MODEL - Multilingual Open Data for EU Languages. Obtenido de https://tilde-model.s3-eu-west-1.amazonaws.com/nodalida2017_Tilde_MODEL.pdf

Steinberger, R., Eisele, A., Klocek, S., Pilos, S., & Schlüter, P. (2012). DGT-TM: A freely Available Translation Memory in 22 Languages. Obtenido de http://www.lrec-conf.org/proceedings/lrec2012/pdf/814_Paper.pdf

Tiedemann, J. (2012). Parallel Data, Tools and Interfaces in OPUS. Obtenido de http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., N. Gomez, A., . . . Polosukhin, I. (2017). Attention Is All You Need. Obtenido de https://arxiv.org/pdf/1706.03762.pdf

Wołk, K., & Marasek, K. (2014). Building Subject-aligned Comparable Corpora and Mining it for Truly Parallel Sentence Pairs. Obtenido de https://www.sciencedirect.com/science/article/pii/S2212017314005453

Wortschatz-Leipzig. (1990). *Leipzig Corpora Collection*. Obtenido de https://wortschatz.uni-leipzig.de/en

Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., . . . Raffel, C. (2021). mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer. Obtenido de https://arxiv.org/pdf/2010.11934.pdf

Ziemski, M., Junczys-Dowmunt, M., & Pouliquen, B. (2016). The United Nations Parallel Corpus. Obtenido de https://aclanthology.org/L16-1561.pdf

Origen corpus monolingües

Leipzig Corpora Collection (2002): *English from Australia web corpus based on material from 2002*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng-au_web_2002_1M.

Leipzig Corpora Collection (2002): *English from Canada web corpus based on material from 2002*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng-ca_web_2002.

Leipzig Corpora Collection (2002): *English from New Zeland web corpus based on material from 2002*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng-nz_web_2002_1M.

Leipzig Corpora Collection (2002): *English from United Kingdom web corpus based on material from 2002*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng-uk_web_2002_1M.

Leipzig Corpora Collection (2007): *English from Wikipedia corpus based on material from 2007*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng_wikipedia_2007_1M.

Leipzig Corpora Collection (2010): *English from Wikipedia corpus based on material from 2010*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng_wikipedia_2010_1M.

Leipzig Corpora Collection (2012): *English from Wikipedia corpus based on material from 2012*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng_wikipedia_2012_1M.

Leipzig Corpora Collection (2015): *English from Newscrawl corpus based on material from 2015*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng_newscrawl_2015_1M.

Leipzig Corpora Collection (2016): *English from Wikipedia corpus based on material from 2016*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng_wikipedia_2016_1M.

Leipzig Corpora Collection (2016): *English from News Typical corpus based on material from 2016*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng_news-typical_2016_1M.

Leipzig Corpora Collection (2018): *English from News corpus based on material from 2018*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng_news_2018_1M.

Leipzig Corpora Collection (2018): *English from Newscrawl corpus based on material from 2018*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng_newscrawl_2018_1M.

Leipzig Corpora Collection (2018): *English from United Kingdom Web Public corpus based on material from 2018*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng-uk_web-public_2018_1M.

Leipzig Corpora Collection (2019): *English from News corpus based on material from 2019*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng_news_2019_1M.

Leipzig Corpora Collection (2020): *English from News corpus based on material from 2020*. Leipzig Corpora Collection. Dataset. https://corpora.uni-leipzig.de?corpusId=eng_news_2020_1M.

Índice de figuras

Índice de tablas