Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: Ruiz Vallecillo	01/02/2025
	Nombre: Araceli	

Laboratorio: word embeddings y transformers para clasificación de texto

1. Utilizando el tokenizador de spacy, que ya conoces, calcula el número promedio de tokens de una muestra de 15 ficheros de la categoría «com.graphics». Indica el código utilizado y el resultado obtenido. (1 punto).

```
#PREGUNTA 1

import spacy
import en_core_web_sm
nlp = en_core_web_sm.load()

fnames = os.listdir(data_dir / "comp.graphics")

total_tokens=0
for i in (range(15)):
    doc = nlp(pathlib.Path(data_dir / "comp.graphics" / fnames[i]).read_text(encoding="latin-1"))
    doc.__len__()
    total_tokens+=doc.__len__()

average_tokens = total_tokens/15

print(average_tokens)
```

Figura 1: Código para calcular el número medio de tokens para una muestra de 15 ficheros

El número medio de tokens para una muestra de 15 ficheros ha sido de **281.53 tokens.**

2. El código proporcionado lee los ficheros uno a uno y, antes de generar el catálogo de datos de entrenamiento y validación, descarta las diez primeras líneas de cada fichero. ¿Cuál es el trozo de código en el que se realiza dicho descarte?, ¿por qué crees que se descartan dichas líneas?, ¿por qué diez y no otro número? (1 punto).

El código en el que se realiza dicho descarte es el siguiente:

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: Ruiz Vallecillo	01/02/2025
	Nombre: Araceli	

```python
#EJERCICIO 2

samples = []
labels = []
class_names = []
class_index = 0


minima_longitud = 100000
minima_cabecera = 100000
fichero_min_longitud = None


for dirname in list_all_dir:
    class_names.append(dirname)
    dirpath = data_dir / dirname
    fnames = os.listdir(dirpath)
    print("Processing %s, %d files found" % (dirname, len(fnames)))
    for fname in fnames:
        fpath = dirpath / fname
        f = open(fpath, encoding="latin-1")
        content = f.read()
        lines = content.split("\n")

        #Comprobamos si el fichero actual es el fichero con un menor numero de lineas
        numero_lineas_total = len(lines)
        minima_longitud = min(minima_longitud, len(lines))
        if (len(lines) == minima_longitud):
            fichero_min_longitud = fpath

       #Descartamos las 10 primeras lineas del texto y hayamos el valor de la cabecera minima
        lines = lines[10:] #Aquí es donde se descartan las 10 primeras líneas de cada archivo
        cabecera = numero_lineas_total - len(lines)
        minima_cabecera = min(minima_cabecera, cabecera)
        content = "\n".join(lines)
        samples.append(content)
        labels.append(class_index)
    class_index += 1

#Mostramos los resultados
print("-----------------------------------------------")
print("Minima longitud hayada: ", minima_longitud)
print("Fichero con menor longitud: ", fichero_min_longitud)
print("-----------------------------------------------")
print(open(fichero_min_longitud).read())
print("-----------------------------------------------")
print("Cabecera minima: ", minima_cabecera)
print("-----------------------------------------------")
print("Classes:", class_names)
print("Number of samples:", len(samples))
```

Figura 2: Código para leer los archivos y descartar las 10 primeras líneas

Se cree que se descartan 10 líneas, ya que al visualizar los textos, se observa cómo estos se componen de una cabecera con metadatos del texto (referencias, webs,...) y luego el propio texto en sí, ambos separados por al menos un salto de línea.

Para poder responder a las preguntas de ***por que se descartan esas lineas y por que dicho número***, se han visualizado varios de los textos y además, en el propio código se han añadido unas líneas con las que se calcula el texto con longitud mínima y la longitud mínima de cabecera. En el resultado se puede observar como el texto con longitud mínima

es de valor 9, que a su vez coincide con el texto con cabecera mínima, dado que hay un texto que se compone únicamente por su cabecera.

Por tanto, se ha elegido el valor de 10, dado que el valor mínimo de la cabecera es 9, por lo que se está eliminando la mayoría, por no decir todos estos metadatos o cabecera de cada texto. No se pone un número superior dado que al ponerlo es posible que podamos toparnos con parte del texto en sí del propio fichero que es de interés para analizar. Y no se pone un número inferior, ya que se sabe que como mínimo la longitud de estos metadatos es 9, y no es objeto de interés para esta actividad analizarlos.

3. ¿Qué se controla con el parámetro validation_split?, ¿por qué se ha elegido ese valor?, ¿qué ocurre si lo modificas? (1 punto).

```
#EJERCICIO 3, si se reduce ese numero cambia el tamaño de la división que se realizan los datos.

# Extract a training & validation split
validation_split = 0.2
num_validation_samples = int(validation_split * len(samples))
train_samples = samples[:-num_validation_samples]
val_samples = samples[-num_validation_samples:]
train_labels = labels[:-num_validation_samples]
val_labels = labels[-num_validation_samples:]
```

Figura 3: Código para separar los datos en los conjuntos de Train y Test

A la hora de realizar un entrenamiento y una predicción, es fundamental separar los datos, dejando un tanto por ciento para el conjunto de datos de Train y otro tanto por ciento para el conjunto de datos de Test o Validación.

El primer conjunto (train) es utilizado para poder entrenar el modelo, en este caso, se suele coger un alto porcentaje del dataset (entre el 70% y 80%). Mientras que el segundo conjunto (test o validación), es un conjunto que nunca ha visto el modelo entrenado, por lo que es utilizado para realizar predicciones y saber cómo de bien se ha entrenado el modelo y cómo de bien predice. Para este conjunto se suele coger un el porcentaje restante que no se cogió en el conjunto de train del conjunto total de datos (30% o 20%).

Por tanto, en este caso, el parámetro ***validation_split***, controla esos tantos porcientos, siendo el valor de este parámetro la cantidad de tanto por ciento que se va a coger del conjunto de datos total para hacer el dataset de test o validación (20% ~ 0.2). Por lo que para el conjunto de entrenamiento se cogerá un **80%** del conjunto de los datos totales.

En el caso de que se modificase dicho valor aumentándolo, podría afectar negativamente, ya que se estaría reduciendo la cantidad de datos para el conjunto de entrenamiento y por tanto su capacidad de generalizar. Mientras que, si se disminuyese dicho valor, se quedaría

un conjunto muy pequeño con el que validar ese entrenamiento, por lo que se podría elevar el riesgo de haber sufrido sobreajuste (overfitting) durante la fase de entrenamiento.

4. Imprime por pantalla un ejemplo (es decir, un elemento del *array*) de train_samples, val_samples, train_labels y val_labels. A tenor de las etiquetas que se utilizan, ¿qué tarea crees que se está intentando entrenar? (1 punto).

```
Alan Carter writes:

>> 3.  On April 19, a NO-OP command was sent to reset the command loss timer to
>> 264 hours, its planned value during this mission phase.

> This activity is regularly reported in Ron's interesting posts. Could
> someone explain what the Command Loss Timer is?

The name is rather descriptive.  It's a command to the spacecraft that tells
it "If you don't hear from Earth after 264 hours, assume something is wrong
with your (the spacecraft) attitude, and go into a preprogrammed search mode
in an attempt to reacquire the signal from Earth."

The spacecraft and Earth are not in constant communication with each other.
Earth monitors the telemetry from the spacecraft, and if everything is fine,
there's no reason to send it any new information.  But from the spacecraft's
point of view, no information from Earth could mean either everything is
fine, or that the spacecraft has lost signal acquisition.  Just how long
should the spacecraft wait before it decides that something is wrong and
begins to take corrective action?  That "how long" is the command loss timer.
During relatively inactive cruise phases, the command loss timer can be set
to rather long values.  In this case, Earth is telling Galileo "expect to
hear back from us sometime within the next 264 hours".

-------------------------------------------
```

```
-------------------------------------------------------
10
```

Figura 4: Resultados de train_samples[0] y train_labels[0]

```
Lines: 14


        Is there anyone out there running a Chicago National
        League Ballclub list?  If so, please send me information
        on it to...
                        andrew@aardvark.ucs.uoknor.edu

        Thanks!

|\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/|
|o|   _    |  Chihuahua Charlie              |  OU is not responsible   |o|
|o| | |    |  Academic User Services         |  for anything anywhere,  |o|
|o| ||||   |  The University of Oklahoma     |  except for that one     |o|
|o|  |_|   |  andrew@aardvark.ucs.uoknor.edu |  incident where 200...   |o|
|o|____________________________________________________________________|o|

-------------------------------------------
```

```
-------------------------------------------------------
6
```

Figura 5: Resultados de val_samples[0] y val_labels[0]

Se puede observar en base de los resultados de cada una de estas variables que se trata de una **tarea de clasificación**, concretamente de la clasificación de un texto en función de su **categoría o topic**. En este caso los **samples**, se corresponden con los textos, mientras que los **labels**, se corresponden con las categorías de los textos.

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: Ruiz Vallecillo	01/02/2025
	Nombre: Araceli	

Por tanto, el texto del conjunto de entrenamiento ha sido clasificado en la categoría 10, mientras que el texto utilizado en el conjunto de test o validación se ha clasificado en la categoría 6.

5. Con output_sequence_length se establece un tamaño fijo para la salida de Vectorizer. ¿Por qué se necesita un tamaño fijo y por qué se ha elegido el valor 200? (1 punto).

Al trabajar con modelos de aprendizaje automático, en este caso con **redes neuronales**, es común que la entrada y salida tengan una dimensión fija. Esto se debe a que los modelos requieren entradas de tamaño constante para realizar operaciones eficientes y tener una representación consistente de los datos.

En el caso específico de ***output_sequence_length***, se establece un tamaño fijo para la salida del *Vectorizer* para garantizar que todas las secuencias de texto tengan la misma longitud en su representación vectorial. Esto es necesario para alimentar adecuadamente los datos a un modelo, ya que generalmente se requiere que todas las entradas tengan la misma dimensión.

En cuanto al **valor elegido de 200**, se ha visto conveniente calcular el número completo de tokens, así como el número medio del número total de tokens (no de una muestra), los percentiles y visualizar la distribución de los mismos.

De los resultados obtenidos, tenemos que la **media de tokens** es **395.79,** mientras que la **mediana o el segundo percentil** es de **242 y** el **tercer percentil**, que hace representación del **75 %** de los datos tiene un valor de **415**. Podría parecer que lo lógico fuese coger el valor del tercer percentil, puesto que representa un alto porcentaje del número de tokens de todos los textos, sin embargo, el valor de la mediana no se muestra tan alejado, siendo además un valor que representa a gran parte de los datos.

Por lo tanto, es probable que para este caso, el valor de dicha variable haya sido elegida en función de la **mediana** de la longitud de las secuencias de texto en el conjunto de datos o consideraciones específicas del problema, de esta forma se asegura que la mayoría de los textos de menor longitud a 200 se procesen completos y se reduzca la longitud promedio, haciéndolo más eficiente. Además, es usual elegir un valor lo suficientemente grande como para abarcar la mayoría de las secuencias, pero no tan grande como causar problemas de rendimiento o memoria. Es posible que se hayan realizado experimentos previos para determinar si un tamaño de 200 era óptimo y adecuado para capturar la información relevante en el contexto del problema.

Es importante tener en cuenta que el valor de ***output_sequence_length*** puede variar según el conjunto de datos y el problema específico que se esté abordando. En algunos casos, puede ser necesario ajustar este valor y realizar pruebas para encontrar el tamaño óptimo que funcione mejor para el modelo y los datos específicos.

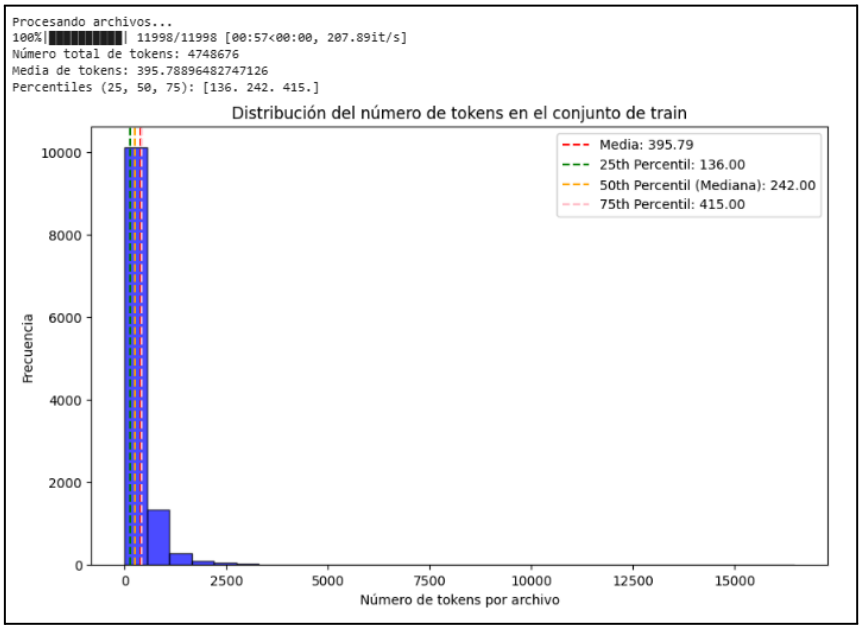

Figura 6: Distribución del número de tokens por archivo y su frecuencia, además de sus percentiles.

Si se observa la distribución, se puede ver como la mediana se encuentra en medio de la cantidad de datos más frecuente, por lo que cojer un valor cercano a este, como lo es 200, solo reafirma que es un buen valor del tamaño de textos a procesar, asegurándonos que los textos más cortos y que son los más frecuentes, se procesan.

6. Indica cuál es la precisión del modelo en el conjunto de datos de entrenamiento y en el conjunto de datos de validación. ¿Qué interpretación puedes dar? Haz, en este punto, un análisis comparativo de los dos modelos ejecutados (1.5 puntos).

	Accuracy Train	Accuracy Test o Validation
Modelo Red Clásica	0.9569	0.6742
Modelo Transformers	0.9615	0.8279

Observando los valores obtenidos para ambos modelos, se puede ver como para el **modelo de red clásica** el conjunto de entrenamiento obtiene unos resultados muy buenos, mientras que para el conjunto de validación o de test hay un notable empeoramiento. Esto indica, que no ha tenido una buena capacidad de generalización, por lo que se ha producido **overfitting,** de tal forma que el modelo se ha 'aprendido' los datos proporcionados en el entrenamiento.

Por otro lado, para el **modelo de transformers**, se puede ver cómo se obtienen unos valores de precisión (accuracy) similares tanto para el conjunto de entrenamiento como para el de test. Por lo que se puede concluir que este modelo (transformers) generaliza mejor y por tanto se adapta mejor que el modelo clásico.

7. En la parte final del código se hace un análisis cualitativo de la salida. Explica el funcionamiento de este análisis e interpreta los resultados. Haz también, en este punto, un análisis comparativo de los dos modelos ejecutados (1 punto).

Figura 7: Predicción de 3 ejemplos con red clásica (izquierda) y red de transformers (derecha).

En la parte final del código, se realizan tres ejemplos para evaluar cómo el modelo clasifica diferentes textos en sus respectivas categorías.

Se proporcionan al modelo textos que no ha visto antes, como:

- "this message is about computer graphics and 3D modeling"
- "politics and federal courts law that people understand with politician and elects congressman"
- "we are talking about religion"

Los textos se convierten en representaciones numéricas utilizando **TextVectorization**, que transforma las palabras en secuencias de enteros basados en el vocabulario aprendido durante el entrenamiento. El modelo procesa estas secuencias y devuelve **probabilidades** asociadas a cada una de las categorías en **list_all_dir**. Finalmente, la función **np.argmax(probabilities[0])** identifica la categoría con la mayor probabilidad, que es la categoría que el modelo predice como la más adecuada para el texto.

Realizando un análisis comparativo de ambos modelos:

Texto 1:

Input: "this message is about computer graphics and 3D modeling"
Predicción red clásica: comp.graphics
Interpretación red clásica: Correcta. El modelo ha identificado con precisión que el texto trata sobre gráficos por computadora.

En el caso del **modelo de transformers**, obtenemos el mismo resultado.

Texto 2:

Input: "politics and federal courts law that people understand with politician and elects congressman"
Predicción red clásica: talk.politics.guns
Interpretación red clásica: Parcialmente correcta. El texto habla de política y leyes federales, pero el modelo lo ha relacionado específicamente con el tema de las armas (*talk.politics.guns*). Esto podría indicar una confusión dentro de las categorías políticas debido a similitudes en el vocabulario.

En el caso del **modelo de transformers**, obtenemos el mismo resultado.

Texto 3:

Input: "we are talking about religion"
Predicción red clásica: : sci.med
Interpretación red clásica:: Incorrecta. El texto habla de religión, pero el modelo lo ha clasificado como *sci.med* (medicina). Esto sugiere que el modelo no ha capturado bien el contexto de términos relacionados con la religión o que necesita más datos de entrenamiento en esas categorías.

Predicción red transformers: alt.atheism
Interpretación red transformers: Incorrecta. El texto habla de religión, pero el modelo lo ha clasificado como *alt.atheism* (atletismo). Esto sugiere que el modelo no ha capturado bien el contexto de términos relacionados con la religión o que necesita más datos de entrenamiento en esas categorías.

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: Ruiz Vallecillo	01/02/2025
	Nombre: Araceli	

8. Explica algunas de las limitaciones que puedes encontrar al modelo entrenado (1.5 puntos).

Las limitaciones que se encuentran para ambos modelos entrenados son las siguientes:

- El modelo ha sido entrenado con datos en inglés, lo cual restringe su capacidad de predicción en otros idiomas. Un buen método para poder solucionar esta limitación sería aplicar técnicas de adaptación como ***fine-tuning***.
- Puede haber la existencia de textos que no coincidan con ningún tema o tópic declarado. En este caso el modelo le asignaría una categoría de manera incorrecta o forzada. Esto último, afectaría considerablemente a la predicción del modelo, dado que no ha sido entrenado con esa categoría adicional y los textos correspondientes a dicho tópic quedarían fuera del dominio, haciendo el modelo poco confiable.
- Por último, la restricción de procesar únicamente secuencias de 200 tokens puede limitar la capacidad del modelo, ya que para textos con una mayor longitud se perdería parte de la información contenida, y puede que de interés, en esos textos. Una solución posible sería segmentar el texto en fragmentos más pequeños, donde posteriormente se procesarían de manera individual y se combinarían sus probabilidades de clasificación. De esta forma, se mejoraría el rendimiento para los textos de una mayor longitud a 200, sin embargo se podría perder la coherencia entre textos.

9. ¿Qué sería necesario para que este modelo pueda interpretar textos en español? (1 punto).

Para que los modelos puedan interpretar textos en español, las estrategias varían según se trate del modelo clásico o del modelo basado en transformers.

En el caso del **modelo basado en transformers**, es recomendable utilizar modelos de lenguaje preentrenados en español, como **BETO** (la versión de BERT en español) o **XLM-R** (XLM-Roberta multilingüe). Estos modelos ya han sido entrenados con grandes volúmenes de texto en español, lo que les permite comprender mejor la semántica, la gramática y las estructuras lingüísticas propias del idioma. Posteriormente, se puede realizar un **ajuste fino (fine-tuning)** con conjuntos de datos más específicos en español, adaptando el modelo a tareas concretas como la clasificación de texto, análisis de sentimientos o detección de tópicos. Además, en el caso de los transformers, es fundamental utilizar **modelos de tokenización adaptados al español**, ya que estos afectan directamente la manera en que el texto es procesado antes de ser analizado por el modelo.

Por otro lado, para el **modelo clásico**, que probablemente emplea técnicas como TF-IDF o bag-of-words junto con un clasificador tradicional, es esencial contar con un **conjunto de datos en español** para construir el vocabulario y los vectores de características adecuados. También sería importante utilizar **recursos lingüísticos específicos del español**, como herramientas de **lematización**, **etiquetadores gramaticales** y **diccionarios**, para mejorar la calidad del preprocesamiento del texto. A diferencia del modelo de transformers, en el modelo clásico no se depende tanto de modelos preentrenados, sino que el rendimiento se mejora mediante un buen preprocesamiento y un vocabulario representativo del idioma.

Una opción viable para ambos modelos, aunque más común en contextos clásicos, es incorporar una capa de **embeddings multilingües** como **LASER** o **MUSE**, que permiten representar textos en diferentes idiomas en el mismo espacio vectorial. También se podría utilizar **modelos de traducción automática** para convertir los textos en español al inglés antes de procesarlos, aunque esto podría introducir errores de interpretación debido a posibles fallos en la traducción. Esta estrategia es más relevante en el modelo clásico, donde los métodos son menos flexibles con múltiples idiomas, mientras que en transformers multilingües como XLM-R esta necesidad se reduce.

Finalmente, para ambos modelos es fundamental realizar una **evaluación continua del rendimiento** en español, ajustando hiperparámetros, corrigiendo errores y recopilando retroalimentación para mejorar la precisión en contextos específicos.