Tarea 7.1

April 23, 2021

0.1 # Tarea 7

Por: Miguel Angel Soto Hernandez

0.2 Introducción a la arquitectura del modelo del transformador

El lenguaje es la esencia de la comunicación humana. Las civilizaciones no habrían nacido nacido sin las secuencias de palabras que forman el lenguaje. Ahora vivimos principalmente en un mundo de representaciones digitales del lenguaje. Nuestra vida cotidiana depende de las funciones lingüísticas digitalizadas del Procesamiento del Lenguaje Natural (PLN). Procesamiento del Lenguaje Natural (PLN) funciones lingüísticas digitalizadas: buscadores web correos electrónicos, redes sociales, mensajes, tuits, mensajes de texto de teléfonos inteligentes, traducciones, páginas web, voz a texto en sitios de streaming para transcripciones, texto a voz en servicios de línea telefónica, y muchas más funciones cotidianas.

En diciembre de 2017, se publicó el artículo seminal Vaswani et al. Attention Is All You Need, escrito por los miembros de Google Brain y Google Research. Había nacido el Transformer. El Transformer superó los modelos de PNL existentes de última generación. El Transformer se entrenó más rápido que las arquitecturas anteriores y obtuvo resultados de evaluación más altos. Los Transformers se han convertido en un componente clave de la PNL.

El mundo digital nunca habría existido sin la PNL. El Procesamiento del Lenguaje Natural sin la PNL. El procesamiento del lenguaje natural habría seguido siendo primitivo e ineficiente sin la inteligencia artificial. inteligencia artificial. Sin embargo, el uso de redes neuronales recurrentes (RNN) y redes neuronales convolucionales (CNN) tiene un coste tremendo en términos de de cálculos y potencia de la máquina.

0.2.1 Los antecedentes del Transformer

A lo largo de los últimos 100 años, muchas grandes mentes han trabajado en la transducción de secuencias y el modelado del lenguaje. Las máquinas han aprendido progresivamente a predecir secuencias probables de palabras. Haría falta un libro entero para citar a todos los gigantes que hicieron posible esto.

A principios del siglo XX, Andrey Markov introdujo el concepto de valores aleatorios y creó una teoría de procesos estocásticos. En la inteligencia artificial (IA) los conocemos como procesos de decisión de Markov (MDP), cadenas de Markov y procesos de Markov. En 1902, Markov demostró que se podía predecir el siguiente elemento de una cadena, una secuencia, utilizando sólo el último elemento pasado de esa cadena. En 1913, lo aplicó a un conjunto de datos de 20.000 letras utilizando secuencias pasadas para predecir las letras futuras de una cadena. Hay que tener

en cuenta que no disponía de ningún ordenador, pero consiguió demostrar su teoría, que todavía se utiliza hoy en día en la IA.

En 1948, se publicó La teoría matemática de la comunicación de Claude Shannon de Claude Shannon. Cita la teoría de Andrey Markov en múltiples ocasiones cuando construye su enfoque probabilístico de la modelización de secuencias. Claude Shannon sentó las bases de un modelo de comunicación basado en un codificador de origen, un transmisor y un decodificador decodificador o decodificador semántico.

En 1950, Alan Turing publicó su artículo seminal Computing Machinery and Intelligence. Alan Turing basó este artículo sobre la inteligencia de las máquinas en la inmensamente éxito de la máquina de Turing que descifra los mensajes alemanes. La expresión inteligencia artificial fue utilizada por primera vez por John McCarthy en 1956. Sin embargo, Alan Turing implementaba la inteligencia artificial en la década de 1940 para descifrar mensajes mensajes codificados en alemán.

En 1954, el experimento Georgetown-IBM utilizó ordenadores para traducir frases rusas ruso al inglés utilizando un sistema de reglas. Un sistema de reglas es un programa que ejecuta una lista de reglas que analizan las estructuras del lenguaje. Los sistemas de reglas siguen existiendo. Sin embargo, crear listas de reglas para los miles de millones de combinaciones lingüísticas de nuestro mundo digital es un reto aún no superado. Por el momento, parece imposible. Pero £quién sabe lo que pasará?

En 1982, John Hopfield in trodujo las redes neuronales recurrentes (RNN), conocidas como redes de Hopfield o redes neuronales "asociativas". John Hopfield se inspiró en W.A. Little, que escribió La existencia de estados persistentes en el cerebro en 1974. Las RNN evolucionaron y surgieron las LSTM tal y como las conocemos.

En la década de 1980, Yann Le Cun diseñó la red neuronal convolucional (CNN) multi-propósito. Aplicó las CNN a las secuencias de texto, y se han utilizado ampliamente para la transducción y el modelado de secuencias. También se basan en estados persistentes que recogen información capa a capa. En los años 90, resumiendo varios años de trabajo, Yann Le Cun produjo la LeNet-5, que dio lugar a los numerosos modelos de CNN que conocemos hoy. La arquitectura de la CNN, por lo demás eficiente, se enfrenta a problemas cuando se trata de dependencias a largo plazo en secuencias muy largas y complejas.

0.3 Generación de textos con los modelos GPT-2 y GPT-3 de OpenAI

En 2020, Brown et al. (2020) describieron el entrenamiento de un modelo GPT-3 de OpenAI con 175 mil millones de parámetros entrenados con aproximadamente un trillón de palabras en 50 petaflops/días. Esto representa unas 50*1020 operaciones al día para 400.000 millones de tokens codificados por pares de bytes. Al mismo tiempo, supimos que OpenAI tenía acceso a un super-ordenador hecho a medida que contenía 280.000 CPUs y 10.000 GPUs.

La inteligencia de la GPT-3 de OpenAI y la potencia de los superordenadores llevaron a Brown et al. (2020) a realizar experimentos de tiro cero. La idea era utilizar un modelo entrenado para tareas posteriores sin entrenar más los parámetros. El objetivo sería que un modelo entrenado pasara directamente a la producción de tareas múltiples. OpenAI decidió restringir el uso de los modelos GPT-3 a usuarios específicos. El futuro de la IA bien podría limitarse a los usuarios de la nube. El tamaño y la potencia de GPT-2 y GPT-3 parecen haber llevado la PNL a otro nivel. Sin embargo, podría no ser el único camino para aumentar el rendimiento de los transformadores.

0.3.1 Ejemplo: Ejecutando BertViz

Installando BertViz Aquí importaremos BertViz, los transformadores Hugging Face y los demás requisitos básicos para implementar el programa:

[ ]:
```
import sys
!test -d bertviz_repo && echo
!test -d bertviz_repo || git clone https://github.com/jessevig/bertviz␣
↪bertviz_repo

if not 'bertviz_repo' in sys.path:
  sys.path += ['bertviz_repo']

!pip install regex
!pip install transformers
```

Importando los módulos

[3]:
```
from bertviz import head_view
from transformers import BertTokenizer, BertModel
```

Definiendo la función HTML

[4]:
```
def llamar_html():
  import IPython
  display(IPython.core.display.HTML(
      '''
      <script src="/static/components/requirejs/require.js"></script>
      <script>
        requirejs.config({
          paths: {
            base: '/static/base',
            "d3": "https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.8/d3.min",
            jquery: '//ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min',
          },
        });
      </script>
      '''
  ))
```

0.3.2 Procesamiento y visualización de las cabezas de atención (attention heads)

Ahora vamos a procesar una traducción para mostrar las cabezas de atención. Podríamos utilizar cualquier modelo de transformación o cualquier tarea. Seguiríamos llegando a las mismas conclusiones.

El cuaderno utiliza un BERT preentrenado, procesa las frases a traducir y muestra la actividad de las cabezas de atención:

[5]:
```
version_modelo = 'bert-base-uncased'
hacer_minusculas = True
```

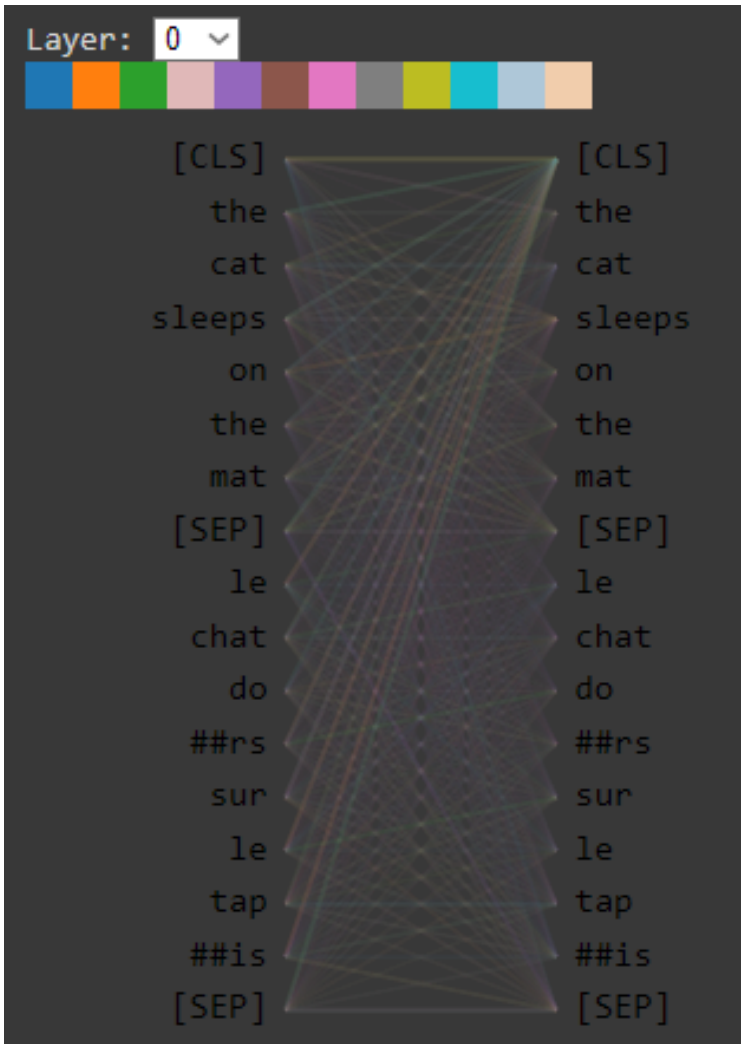
[6]:
```
modelo = BertModel.from_pretrained(version_modelo, output_attentions=True)
tokenizador = BertTokenizer.from_pretrained(version_modelo,
                                            do_lower_case=hacer_minusculas)
```

[7]:
```
frase_a = 'The cat sleeps on the mat'
frase_b = 'Le chat dors sur le tapis'
entradas = tokenizador.encode_plus(frase_a, frase_b, return_tensors='pt',
                                   add_special_tokens=True)
```

[11]:
```
tipo_tokens_ids = entradas['token_type_ids']
ids_entrada = entradas['input_ids']
```

[12]:
```
atencion = modelo(ids_entrada, token_type_ids=tipo_tokens_ids)[-1]
lista_id_entrada = ids_entrada[0].tolist()
tokens = tokenizador.convert_ids_to_tokens(lista_id_entrada)
```

[13]:
```
llamar_html()
head_view(atencion, tokens)
```



```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.Javascript object>
```

Nuestra experimentación empírica con las cabezas de atención nos lleva a algunas conclusiones: - El proceso de atención tiene en cuenta todos los pares de palabras posibles para aprender las conexiones entre ellas. Cuanto mayor sea la ventana de contexto, más pares serán analizados. - Si un texto tiene 100.000 palabras, esto conduce a 100.000 veces 100.000 pares de palabras. Eso se traduce en 10.000 millones de pares por cada paso. La potencia de los ordenadores para llevar a cabo este proceso es alucinante y se necesitan superordenadores para conseguir un rendimiento aceptable. - El número de capas conlleva unos requisitos de memoria considerables para almacenar la información, incluida la hinchazón de las capas feedforward que alcanzan los terabytes para los modelos que contienen miles de capas. - Una de las razones por las que podríamos analizar secuencias largas es para la generación de música transformada.

0.4 Deja que tus datos hablen: Historia, preguntas y respuestas

La comprensión lectora requiere muchas habilidades. Cuando leemos un texto, nos fijamos en las Cuando leemos un texto, nos fijamos en las palabras clave y en los acontecimientos principales y creamos representaciones mentales del contenido. A continuación, podemos responder a las preguntas utilizando nuestro conocimiento del contenido y nuestras representaciones. También examinamos cada pregunta para evitar trampas y cometer errores.

Los transformers, por muy poderosos que sean, no pueden responder a las preguntas abiertas. Un entorno abierto significa que alguien puede hacer cualquier pregunta sobre cualquier tema, y un transformador respondería correctamente. Eso sigue siendo imposible. Los transformadores suelen utilizar conjuntos de datos de entrenamiento de dominio general en un entorno cerrado de preguntas y respuestas. Por ejemplo, las respuestas críticas en la atención médica y la interpretación de la ley requieren una funcionalidad de PNL adicional. adicional.

Sin embargo, los transformadores no pueden responder correctamente a ninguna pregunta, independientemente de que el entorno de entrenamiento esté cerrado con secuencias de pregunta-respuesta preprocesadas o no. Si una secuencia contiene más de un sujeto y proposiciones compuestas, un modelo transformador puede hacer predicciones erróneas.

0.4.1 Ejemplo: Preguntas y respuestas

[ ]:
```
!pip install transformers
```

[2]:
```
from transformers import pipeline
```

[ ]:
```
nlp_qa = pipeline('question-answering')
```

Ejemplo 1:

The traffic began to slow down on Pioneer Boulevard in Los Angeles, making it difficult to get out of the city. However, WBGO was playing some cool jazz, and the weather was cool, making it rather pleasant to be making it out of the city on this Friday afternoon. Nat King Cole was singing as Jo and Maria slowly made their way out of LA and drove toward Barstow. They planned to get to Las Vegas early enough in the evening to have a nice dinner and go see a show.

[4]:
```
secuencia = "The traffic began to slow down on Pioneer Boulevard in Los␣
↪Angeles, making it difficult to get out of the city. However, WBGO was␣
↪playing some cool jazz, and the weather was cool, making it rather pleasant␣
↪to be making it out of the city on this Friday afternoon. Nat King Cole was␣
↪singing as Jo and Maria slowly made their way out of LA and drove toward␣
↪Barstow. They planned to get to Las Vegas early enough in the evening to␣
↪have a nice dinner and go see a show."
```

[18]:
```
pregunta = 'Where is Pioneer Boulevard ?'
respuesta = nlp_qa(context=secuencia, question='Where is Pioneer Boulevard ?')
respuesta = respuesta['answer']

print(f'Pregunta: {pregunta}')
print(f'Respuesta: {respuesta}')
```

```
The traffic began to slow down on Pioneer Boulevard in Los Angeles, making it
difficult to get out of the city. However, WBGO was playing some cool jazz, and
the weather was cool, making it rather pleasant to be making it out of the city
on this Friday afternoon. Nat King Cole was singing as Jo and Maria slowly made
their way out of LA and drove toward Barstow. They planned to get to Las Vegas
early enough in the evening to have a nice dinner and go see a show.
Pregunta: Where is Pioneer Boulevard ?
Respuesta: Los Angeles
```

Ejemplo 2:
NER (Named Entity Recognition)

[20]:
```
nlp_ner = pipeline("ner")
```

[21]:
```
print(nlp_ner(secuencia))
```

```
[{'word': 'Pioneer', 'score': 0.9735257625579834, 'entity': 'I-LOC', 'index': 8,
'start': 34, 'end': 41}, {'word': 'Boulevard', 'score': 0.9944824576377869,
'entity': 'I-LOC', 'index': 9, 'start': 42, 'end': 51}, {'word': 'Los', 'score':
0.9995775818824768, 'entity': 'I-LOC', 'index': 11, 'start': 55, 'end': 58},
{'word': 'Angeles', 'score': 0.9995693564414978, 'entity': 'I-LOC', 'index': 12,
'start': 59, 'end': 66}, {'word': 'W', 'score': 0.991984486579895, 'entity':
'I-ORG', 'index': 26, 'start': 121, 'end': 122}, {'word': '##B', 'score':
0.990750253200531, 'entity': 'I-ORG', 'index': 27, 'start': 122, 'end': 123},
{'word': '##G', 'score': 0.9884582161903381, 'entity': 'I-ORG', 'index': 28,
'start': 123, 'end': 124}, {'word': '##O', 'score': 0.972268283367157, 'entity':
'I-ORG', 'index': 29, 'start': 124, 'end': 125}, {'word': 'Nat', 'score':
0.9966881275177002, 'entity': 'I-PER', 'index': 59, 'start': 264, 'end': 267},
{'word': 'King', 'score': 0.997648298740387, 'entity': 'I-PER', 'index': 60,
'start': 268, 'end': 272}, {'word': 'Cole', 'score': 0.9986170530319214,
'entity': 'I-PER', 'index': 61, 'start': 273, 'end': 277}, {'word': 'Jo',
'score': 0.9978788495063782, 'entity': 'I-PER', 'index': 65, 'start': 293,
'end': 295}, {'word': 'Maria', 'score': 0.9988165497779846, 'entity': 'I-PER',
'index': 67, 'start': 300, 'end': 305}, {'word': 'LA', 'score':
```

```
0.998134434223175, 'entity': 'I-LOC', 'index': 74, 'start': 335, 'end': 337},
{'word': 'Bar', 'score': 0.9970266819000244, 'entity': 'I-LOC', 'index': 78,
'start': 355, 'end': 358}, {'word': '##sto', 'score': 0.8573917746543884,
'entity': 'I-LOC', 'index': 79, 'start': 358, 'end': 361}, {'word': '##w',
'score': 0.9920249581336975, 'entity': 'I-LOC', 'index': 80, 'start': 361,
'end': 362}, {'word': 'Las', 'score': 0.9993551969528198, 'entity': 'I-LOC',
'index': 87, 'start': 387, 'end': 390}, {'word': 'Vegas', 'score':
0.9989539384841919, 'entity': 'I-LOC', 'index': 88, 'start': 391, 'end': 396}]
```

[23]:
```
nlp_qa = pipeline('question-answering')

print("Pregunta 1.", nlp_qa(context=secuencia,
                            question='Where is Pioneer Boulevard ?'))
print("Pregunta 2.", nlp_qa(context=secuencia,
                            question='Where is Los Angeles located?'))
print("Pregunta 3.", nlp_qa(context=secuencia,
                            question='Where is LA ?'))
print("Pregunta 4.", nlp_qa(context=secuencia,
                            question='Where is Barstow ?'))
print("Pregunta 5.", nlp_qa(context=secuencia,
                            question='Where is Las Vegas located ?'))
```

```
Pregunta 1. {'score': 0.9879737496376038, 'start': 55, 'end': 66, 'answer': 'Los
Angeles'}
Pregunta 2. {'score': 0.9875388741493225, 'start': 34, 'end': 51, 'answer':
'Pioneer Boulevard'}
Pregunta 3. {'score': 0.5090540647506714, 'start': 55, 'end': 66, 'answer': 'Los
Angeles'}
Pregunta 4. {'score': 0.3695431649684906, 'start': 387, 'end': 396, 'answer':
'Las Vegas'}
Pregunta 5. {'score': 0.21839778125286102, 'start': 355, 'end': 362, 'answer':
'Barstow'}
```

0.5 Análisis de sentimientos

0.5.1 LSTM

Cómo funciona una LSTM

La mejor manera de entender una LSTM es imaginarla como una estructura en cadena con dos módulos no repetitivos y un módulo repetitivo. Dentro de esta estructura, el módulo de repetición tiene tres puertas. Son la puerta de entrada, la puerta de salida y la puerta de olvido. El estado de la célula conecta toda la cadena, con interacciones lineales mínimas.

Las LSTM utilizan una estructura llamada puerta para regular el flujo de datos que entran y salen de una célula. Funcionan de forma similar a las puertas condicionales (and, or, xor, etc.). Basándose en un conjunto de condiciones, deciden qué información debe pasar por la puerta. Las condiciones las decide la LSTM y no es necesario programarlas explícitamente.

- **Puerta del olvido (Forget gate)**: La salida de la puerta del olvido indica a la célula de la célula qué información debe olvidarse multiplicando una posición de la matriz por cero.

posición en la matriz por cero. Si la salida de la puerta del olvido es 1, la información se mantiene en el estado de la célula.

- **Puerta de entrada (Input gate)**: Esta puerta determina qué información debe entrar en el estado de celda. El componente importante es la función de activación. La puerta de entrada tiene una función sigmoidea que tiene un rango de [0,1]. La ecuación ecuación del estado de la célula es una suma entre los estados anteriores de la célula, por lo que la función Sigmoid solo sólo añadirá memoria y no podrá eliminar memoria. Si sólo puede añadir un número flotante entre [0,1], ese número nunca será cero. Esto hace que sea incapaz de olvidar. Por eso la puerta de modulación de entrada tiene una función de activación tanh. función de activación tanh. tanh tiene un rango de [-1, 1] y permite que el estado de la célula olvide la memoria.
- **Puerta de salida (Output gate)**: Esta puerta tiene en cuenta todos los posibles valores y utiliza una función de activación sigmoidea Sigmoide para decidir qué información debe ir al siguiente estado oculto.

La entrada para un LSTM tiene la forma de una matriz tridimensional. Donde la primera dimensión representa el tamaño del lote, la segunda dimensión representa el número de pasos de tiempo que estamos alimentando una secuencia y la tercera dimensión representa el número de unidades en una secuencia de entrada. - S(t-1)es el estado de la célula - f(gt) es la puerta de olvido - i(gt) es la puerta de entrada - o(gt) es la puerta de salida

Etapa 1

El LSTM decide qué información descartar del estado de la célula estado de la célula. Esta decisión es tomada por la capa de la puerta del olvido, que utiliza una función de activación Sigmoide Sigmoide. Esto significa que se asignará un valor entre 0 y 1. El valor 1 representa "almacenar" y el 0 representa "descartar".

Supongamos que el peso es W_{fg} y el sesgo es b_{fg}. Ahora, según al diagrama, tenemos dos $entradas - n_{(t-1)}$ y a. Ambos valores se multiplicarán por el peso (W_{fg}) y luego el producto resultante se sumará al sesgo (b_{fg}). Por último, se aplicará al resultado la función de activación sigmoidea.

Se puede derivar matemáticamente como sigue:

$$f_{gt} = Sigmoid(W_{fg}[n_{t-1}, a] + b_{fg})$$

Etapa 2

A partir de la información restante, la LSTM decide qué nueva información almacenar en el estado de la célula. Esto se hace en un proceso de dos pasos:

1. La capa de entrada utiliza una función sigmoidea para decidir qué valores se actualizarán. Supongamos que el peso es W_{ig} y el sesgo es b_{ig}.

$$i_{gt} = Sigmoid(W_{ig}[n_{t-1}, a] + b_{ig})$$

2. Una capa tanh crea un vector de los nuevos valores. Supongamos que el peso es W_c y el sesgo es b_c.

$$S_{t1} = tanh(W_c[n_{t-1}, a] + b_c)$$

Etapa 3

Ahora combinamos los resultados de estos dos pasos para crear una actualización del estado de la celda. El antiguo estado de la célula $S_{(t-1)}$ se multiplica por f_{gt}. Luego añadimos el resultado al producto de (i_{gt} * S_{t1})

$$S_{t2} = (f_{gt} * S_{t-1}) + (i_{gt} * S_{t1})$$

Etapa 4
Finalmente, el LSTM decide la salida. Esto se hace en un proceso de dos pasos:

1. La capa de salida utiliza una capa sigmoidea para decidir qué partes del estado de la célula serán la salida. Supongamos que el peso es W_c y el sesgo es b_c:

$$o_{gt} = Sigmoid(W_{og}[n_{t-1}, a] + b_{og})$$

2. El resultado se pasa por una función de activación tanh, para obtener valores entre -1 y 1. Luego el resultado se multiplica por la salida de la puerta Sigmoide.

$$n_t = o_{gt} * S_t$$

0.5.2 Ejemplo: Análisis de sentimientos

Importar librerías

[24]:
```
import tensorflow as tf
import numpy as np
import pandas as pd
import re
import keras
from keras import Model
from tensorflow.keras.layers import Flatten,LSTM, Dense, Flatten, Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from keras_preprocessing.text import Tokenizer
from keras.initializers import glorot_uniform
from sklearn import model_selection
```

Cargar los datos

[26]:
```
with open('/content/drive/MyDrive/Colab Notebooks/CIC/1 semestre/Matematicas␣
 ↪para las ciencias de la computacion/datasets/amazon_review_polarity_csv/
 ↪train.csv', 'r') as file:
  text = file.readlines()
```

[27]:
```
# creamos el dataframe
X_train = pd.DataFrame()
```

[30]:
```
# llenar en el dataframe
palabra = []
etiqueta = []
```

```
for i in text:
  i = i.split()
  etiqueta.append(1) if i[0] == '__label__2' else etiqueta.append(0)
  palabra.append(' '.join(i[1:]))

X_train['consumer_review'] = palabra
X_train['polarity_label'] = etiqueta
```

[37]:
```
# mostrar como queda el dataframe
X_train.head()
```

[37]:

	consumer_review	polarity_label
0	even for the non-gamer","This sound track was ...	0
1	best soundtrack ever to anything.","I'm readin...	0
2	soundtrack is my favorite music of all time, h...	0
3	Soundtrack","I truly like this soundtrack and ...	0
4	Pull Your Jaw Off The Floor After Hearing it",...	0

Praparar los datos

Como se menciona en la descripción del proyecto, por ahora sólo utilizaremos el 20% del total de los datos. Así que se utiliza la función `model_selection.train_test_split` en sklearn para indicar que, de todo el conjunto de datos, se quiere utilizar sólo el 20%.

[38]:
```
_, X_set, _, y_set = model_selection.train_test_split(
    X_train['consumer_review'], X_train['polarity_label'],
    test_size=0.02)
```

Limpiar los datos

Los ordenadores no pueden leer el texto como lo hacen los humanos. Por lo tanto, hay que cambiar el formato de las reseñas en el conjunto de datos para que el sistema las entienda más fácilmente. Esto significa que debe eliminar las mayúsculas, la puntuación y los artículos (a, an y the). Para ello, definirá una función personalizada.

[40]:
```
def limpiar_datos(texto_entrada):
  # Remover los signos de puntuación
  texto_salida = re.sub('[^a-zA-Z]', ' ', texto_entrada)

  # Convertir mayusculas a minusculas
  texto_salida = "".join(list(map(lambda x: x.lower(), texto_salida)))

  # Remover caracteres simples
  texto_salida= re.sub(r"\s+[a-zA-Z]\s+", ' ', texto_salida)
  return texto_salida
```

[41]:
```
# Guardamos los datos ya limpios
set_texto = []

for reviews in list(X_set):
  set_texto.append(limpiar_datos(reviews))
```

Ahora creamos un nuevo marco de datos para almacenar los datos limpios. Si utiliza el mismo

marco de datos que antes, los datos originales se sobrescribirán con los datos limpios.

[43]:
```
X_train= pd.DataFrame()
X_train['consumer_review'] = set_texto
X_train['polarity_label'] = list(y_set)
X_train.head()
```

[43]:

	consumer_review	polarity_label
0	basic not for me way over the basics win ...	0
1	great book to read to introduce the author t...	0
2	have been using pampers since my baby was born...	0
3	obsolescence not so fast mr coffee had t...	0
4	crawford big city have been listening to thi...	0

Estos datos están limpios, pero aún necesitan más preprocesamiento. Primero hay que dividirlos en un 70% para el conjunto de datos de entrenamiento y un 30% para el conjunto de datos de prueba. Utiliza la misma función de división de sklearn que antes.

[44]:
```
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    X_train['consumer_review'], X_train['polarity_label'], test_size=0.3)
```

Para aplicar el tokenizador, necesitamos convertir las listas en arreglos

[45]:
```
X_train = np.array(X_train.values.tolist())
X_test = np.array(X_test.values.tolist())
y_train = np.array(y_train.values.tolist())
y_test = np.array(y_test.values.tolist())
```

[47]:
```
# Aplicamos el tokenizador
tokenizador = Tokenizer()
tokenizador.fit_on_texts(X_train)
index_palabra = tokenizador.word_index
tamano_total = len(index_palabra) + 1
print(tamano_total)
```

```
76740
```

Convertir la entrada de texto en secuencias. Esta es la forma más fácil de procesar para el modelo.

[48]:
```
X_train = tokenizador.texts_to_sequences(X_train)
X_test = tokenizador.texts_to_sequences(X_test)
```

Para evadir la imcompatibilidad debido al tamaño del arreglo, necesitamos agregar padding a los datos

[49]:
```
long_maxima = 100
X_train = pad_sequences(X_train, padding='post', maxlen=long_maxima)
X_test = pad_sequences(X_test, padding='post', maxlen=long_maxima)
```

Estructurar el modelo

[50]:
```
modelo = Sequential()
modelo.add(Embedding(tamano_total, 20, input_length=long_maxima))
modelo.add(LSTM(32, dropout=0.2, recurrent_dropout=0.2))
modelo.add(Dense(1, activation='sigmoid'))
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernel since it doesn't meet
the cuDNN kernel criteria. It will use generic GPU kernel as fallback when
running on GPU
```

Compilar el modelo

[51]:
```
modelo.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
print(modelo.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 20)           1534800
_________________________________________________________________
lstm (LSTM)                  (None, 32)                6784
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 1,541,617
Trainable params: 1,541,617
Non-trainable params: 0
_________________________________________________________________
None
```

Entrenar el modelo

[52]:
```
modelo.fit(X_train, y_train, batch_size=128, epochs=5, verbose=1,
           validation_data=(X_test, y_test))
modelo.save('sentiment_analysis.h5')
```

```
Epoch 1/5
394/394 [==============================] - 147s 359ms/step - loss: 0.1325 - acc:
0.9904 - val_loss: 5.8259e-04 - val_acc: 1.0000
Epoch 2/5
394/394 [==============================] - 141s 359ms/step - loss: 4.7184e-04 -
acc: 1.0000 - val_loss: 2.2501e-04 - val_acc: 1.0000
Epoch 3/5
394/394 [==============================] - 143s 362ms/step - loss: 2.0018e-04 -
acc: 1.0000 - val_loss: 1.2226e-04 - val_acc: 1.0000
Epoch 4/5
394/394 [==============================] - 143s 362ms/step - loss: 1.1326e-04 -
acc: 1.0000 - val_loss: 7.6678e-05 - val_acc: 1.0000
Epoch 5/5
394/394 [==============================] - 142s 361ms/step - loss: 7.2325e-05 -
acc: 1.0000 - val_loss: 5.2117e-05 - val_acc: 1.0000
```

[ ]: