Modelo de lenguaje para reconocer si una opinión es buena en twitter sobre un tema específico

Aquino Santiago Rogelio Gerardo, Moreno Madrid María Guadalupe, Ramírez Ancona Simón Eduardo, Ruiz Pérez Ariel

1. Introducción

1.1. Justificación

El modelado estadístico del lenguaje se ocupa de la construcción de modelos probabilísticos de secuencias de palabras. Dichos modelos se pueden utilizar para discriminar secuencias probables de las improbables, una tarea importante para realizar el reconocimiento de voz, la recuperación de información y la traducción automática (Mnih, A. & Hinton, G. E. 2009). Sin embargo, esto es intrínsecamente difícil debido a la maldición de la dimensionalidad: es probable que una secuencia de palabras en la que se probará el modelo sea diferente de todas las secuencias de palabras vistas durante el entrenamiento (Bengio i.e 2003).
Keras tiene dos modos de construir redes neuronales. El más simple es el modelo secuencial que solo permite que las capas se agreguen en secuencia, de la cual, se mostrará en las siguientes páginas su uso y prácticas en entrenamientos de tweets.

1.2. Planteamiento del problema

La plataforma de Twitter es una de las redes sociales utilizadas dia a dia en todo el mundo. Es muy concurrida por los usuarios para expresar sus sentimientos y pensamientos con los demás, ya sea de su dia a dia o situaciones que son del interés público, es justo este último punto en donde podemos ver que es lo que opina un sector de la población en ciertos acontecimientos, dicho de otra forma, cuál fue el impacto de dicha situación.

Dichos tweets pueden ser clasificados de forma sencilla si es que son positivos o negativos. La forma en que sabremos que algo puede ser negativo o positivo es porque de antemano se conocen palabras que se suelen utilizar cuando se da una opinión negativa o positiva. Esta será nuestra base para poder clasificar los tweets, un conjunto de datos ya clasificados.

Pero analizar palabra a palabra puede ser complicado, para ello es mejor:

1. Asociar con cada palabra del vocabulario un vector de características de palabra distribuida (un vector de valor real) utilizando el método secuencial de Keras
2. Dicho vector será la entrada a una red neuronal secuencial, para entrenarla y validar sus salidas.

3. Comprobar su funcionamiento con datos no conocidos por la red.

Como se puede ver, es un proceso sencillo, sin embargo, la aplicación de la misma es complicada y, en el transcurso de este escrito, se detalla el cómo se aplican dichos pasos.

Podemos pensar entonces: ¿Es posible clasificar los tweets de distintos usuarios en positivos y negativos de una forma correcta?

1.3. Hipótesis

La pregunta a realizar en este trabajo es ¿Cómo aplicar el método secuencial de Keras para realizar el modelado estadístico utilizando los tweets populares, en el caso de nuestro proyecto, tweets respecto al capitolio como entradas de la misma?.

Las redes neuronales se definen en Keras como una secuencia de capas. El contenedor para estas capas es la clase 'Sequential', siendo este método el por cual se realizarán las iteraciones necesarias a través de las épocas.

Se plantea que el programa sea analizado por capas, utilizando el método secuencial de Keras, dividiendo las opiniones a través de entrenamiento previo con reseñas de películas, utilizando la información obtenida en el entrenamiento en diferentes épocas usando el tema del capitolio en twitter, con ayuda de la herramienta twin.

En caso de realizarse satisfactoriamente las pruebas, se mostrarán las reseñas positivas y negativas de la aplicación, sin importar si de por medio hayan palabras de doble sentido o irónicas.

En caso de realizarse y fuese con un resultado aceptable, se mostrarán los tweets positivos y negativos, con palabras usadas literalmente(es decir, ignorara si hay palabras con doble sentido o irónicas).

1.4. Descripción del resto del artículo

Se explica el contenido del resto de los apartados de la investigación.

En este escrito:

1. Revisar la parte teórica del método secuencial de Keras, así mismo como explicarla usando como base el artículo que publicó en el 2003.
2. Se realizará el experimento del mismo, este experimento se realizará de forma que pueda utilizarse tweets como entradas para entrenar, en este caso, utilizando los eventos recientes del Capitolio en Estados Unidos de América
3. Comprobar los resultados con diferentes datasets.
4. Comparar los datos que el mismo método dio de resultados.
5. Analizar los resultados para llegar a una conclusión y verificar si el experimento fue o no un éxito.

2. Marco teórico

2.1. Antecedentes

Un modelo del lenguaje es un mecanismo para definir la estructura del lenguaje, es decir, para restringir adecuadamente las secuencias de unidades lingüísticas más probables, y suelen ser útiles en aplicaciones que necesiten de una sintaxis y semántica compleja.

Uno de los modelos que se suelen utilizar es el conteo de n-gramas para hacer un modelado del lenguaje, es decir, un conteo de n caracteres o palabras acompañado de una técnica de suavizado que permitiera considerar secuencias desconocidas durante el entrenamiento de dicho modelo.

Dicho modelo se puede entender como:

$$P(W) = \Pi_{i=1..n}^{n} P(wi \mid w1 \ldots wi-1)$$

Con N=2 tendremos Bigramas

$$P(W) = P(w1) \,.\, P(w2 \mid w1) \,.\, P(w3 \mid w2) \ldots . P(wn \mid wn-1)$$

Con N= 3 tendremos Trigramas

$$P(W) = P(w1) \,.\, P(w2 \mid w1) \,.\, P(w3 \mid w2w1) \ldots . P(wn \mid wn-2 \,.wn-1)$$

Otro de los modelos clásicos de lenguaje son los basados en gramáticas, los cuales representan las restricciones del lenguaje de una manera natural, así como modelar dependencias tan largas como se quiera. Una de sus aplicaciones distintas es la de gramáticas libres de contextos estocásticas en FNC, en donde se han desarrollado algoritmos como los de: Probabilidad de una cadena(Inside y Outside), Probabilidad de la mejor derivación(Conteo de Viterbi), Estimación de probabilidades de cada regla.

Para obtener un rendimiento mayor respecto a los antes mencionados, se crearon técnicas basadas en redes neuronales. Entre otros trabajos se encuentra uno de los tipos de redes neuronales más empleados para el modelado del lenguaje que son las redes neuronales recurrentes, en donde cada una de las neuronas, además de tener información acerca de su entrada actual en un instante de tiempo, también tienen información sobre su operación con su entrada anterior en un instante anterior de tiempo. Esta propiedad convierte a este tipo de redes en una herramienta muy adecuada para el modelado de series temporales y de secuencias de unidades de texto en particular, donde dada una secuencia de datos de entrada se obtenga una predicción para el dato siguiente en la secuencia.

Estos modelos de lenguaje han tenido un gran auge en la incorporación del aprendizaje profundo, con la ventaja de que tratan de aprender las probabilidades partiendo de los datos que se encontraran representados en un espacio vectorial, es decir, una representación geométrica, en donde se toma en cuenta la posición de los puntos en el plano y si tiene vecinos se calcula su probabilidad.

2.2. Teoría utilizada (Estado de la técnica)

Redes Neuronales

Las redes neuronales son redes interconectadas masivamente en paralelo de elementos simples(usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico. La unidad básica de la red es la neurona y se trata de una unidad de procesamiento de información, es un dispositivo simple de cálculo que ante un vector de entradas proporciona una única salida.

Dichas redes se pueden clasificar por su arquitectura o aprendizaje. La arquitectura de una red consiste en la disposición y conexionado de las neuronas, en donde se distinguen el número de capas y su tipo(ocultas,visibles,entrada y salida) y el sentido de la interconección entre neuronas; pudiendo clasificarlas en monocapa y multicapa.

El aprendizaje se basa en el entrenamiento de la red con patrones hasta encontrar una respuesta esperada. El aprendizaje se puede clasificar en supervisado(se proponen patrones de salida), no supervisado(no se proponen patrones de salida) e híbrido.

Las ventajas que pueden brindar estas redes son, entre otras, el aprendizaje adaptativo, auto-organización, tolerancia a fallos, operación en tiempo real.

Arquitectura de Bengio(2003)

Bengio propone un modelo basado en una red neuronal profunda para estimar las probabilidades de transición a partir de los n-gramas. Una de sus principales ventajas referentes a modelos anteriores es que mientras aprende las probabilidades, paralelamente aprenda una representación vectorial de las palabras, la cual permite computar dichas probabilidades.

Dicho modelo se basa en tres puntos principales, los cuales son:

1. Representar cada palabra en un vocabulario a partir de un vector distribuido de rasgos (un vector con entradas en R).
2. Expresar la función de probabilidad conjunta de secuencia de palabras en términos de los vectores de rasgos de las palabras en la secuencia.
3. Aprender simultáneamente los vectores de rasgos y los parámetros de la función de probabilidad.

Para estimar la distribución de probabilidad con las redes neuronales se utiliza la función Softmax, la cual está dada por:

$$p(wj|wi) = Softmax[aj(i)] = \frac{e^{aj\,(i)}}{\sum_k e^{aj\,(i)}}$$

La arquitectura de la red neuronal se compone de las capas:

- **Entrada: Dada una palabra wi, con i = 1,...,N ,se obtiene el vector one-hot x(i) a partir del ındice i.**

La entrada de la red neuronal debe ser un valor fijo. En concreto la entrada de la red neuronal es un índice de la palabra. Sea x(i) el vector que representa a la palabra wi, este vector está definido indexicalmente por:

$$x(i)_k = \begin{cases} 1 & \text{si } k = i \\ 0 & \text{si } k \neq i \end{cases}$$

Esta forma de representar los índices en forma vectorial se conoce como representación one-hot, donde x(i) será un vector de dimensión N, donde cada entrada representa el índice a un sımbolo del vocabulario.

- **Capa de embedding: Determinada por: C(i) = Cx(i).**

Esta capa oculta es muy sencilla, pues se define como una función lineal dada por C(i) = Cx(i). En este caso C ∈ ×N es una matriz de N × d (d es un hiperparametro). Esto permite que el acceso a las representaciones vectoriales distribuidas sea más sencilla pues cada índice i apunta a la i-ésima columna de esta matriz.

- **Capa oculta: Determinada por:h(i) = tanh(WC(i) +b)**

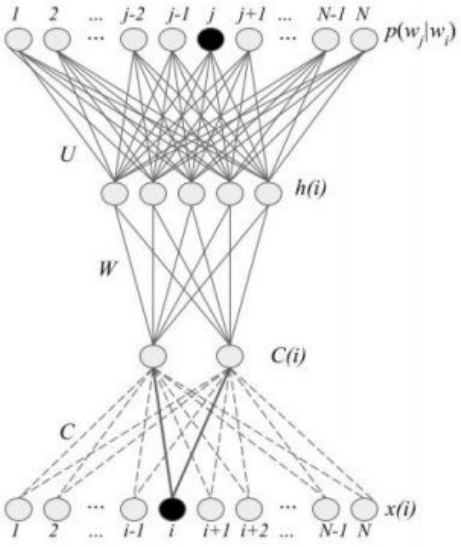
El objetivo de esta capa oculta es que se puedan estimar las probabilidades de manera adecuada para la salida. Se tiene una pre-activación WC(i) +b y la función de activación es la tangente hiperbólica.

- **Salida: Determinada por la pre-activación: a(i) = Uh(i) +c**

Esta capa está definida por las pre-activaciones aj(i) = Uh(i) +c y la función Softmax. Cada entrada $a_j(i)$ representa una salida asociada a un símbolo w_j del vocabulario. Entonces la activación queda como:

$$p(w_j|w_i) := \frac{e^{a_j(i)}}{\sum_{k=1}^{K} e^{a_k(i)}}$$

De forma gráfica se puede ver como:



Para poder entrar a esta red, el corpus de entrenamiento será indispensable y tendrá una influencia considerable en la estimación que haga el modelo del lenguaje neuronal. A partir de dicho corpus se debe de obtener los n-gramas. Estos n-gramas deben incluir a los símbolos < BOS > de inicio de una cadena y < EOS > que indica el final de la cadena y estos símbolos deben insertarse a las cadenas del corpus de entrenamiento.

Se debe de incluir una función de riesgo siendo la entropía cruzada, tomando la forma de:

$$R(\theta) = -\sum_i \sum_k y_k \ln p(w_k|w_i)$$

Es decir, que en el entrenamiento se observa cada bigrama (w_i,w_j) se obtendrá el vector x(i) y a partir de esto se podrán determinar las N salidas de la red $p(w_k|w_i)$.

Para poder compensar el error que se genera al tener salidas cuando se observan los distintos bigramas es necesario hacer uso del backpropagation y actualizar los pesos de la red. Por lo que se tiene que para las distintas capas se tiene:

Capa de salida : $d_{out}(k) = p(w_k|w_i) - y_k$

:

Capa oculta:

Capa de embedding: $d_h(k) = [1 - h(i)_k^2] \sum_q U_{k,q} d_{out}(q)$

$$d_C(k) = \sum W_{k,q} d_h(q)$$

3. Configuración del experimento

3.1. Datos

Lo que se pretende en este proyecto hacer es utilizar Tweets en análisis positivos o negativos, con el fin de clasificarlos si el contenido de este tiene una postura negativa o positiva.

Para obtener los datos se hará uso de la api Twin, el cual recopila tweets desde Twitter para ser usado en nuestro programa. La cantidad máxima de datos que se pueden obtener es de 3200 y extrae toda la información asociada al tweet como al usuario que lo publicó. Para efectos prácticos solamente requerimos del contenido del tweet únicamente y que esté escrito en inglés, no nos interesan los usuarios, países, horario, retweets, likes, etc. Estos datos están alojados(y se extraen) directamente en la aplicación de Twitter, twint únicamente hace el scrapping de los tweets sin necesitar la API de esta plataforma.
Además se hará uso de datos de opiniones de películas de la plataforma IMDB clasificadas en positivas y negativas, con el fin de entrenar a nuestro modelo, ya que estos datos se encuentran etiquetados en forma binaria(neg y pos o 0 y 1). Estos datos están comprimidos, dentro se encuentra, siendo lo más importante, dos carpetas: train y test. La primera de ella contiene dos subcarpetas con las opiniones positivas en una y las negativas en otra, lo mismo pasa con la carpeta de test. Las opiniones se encuentran en archivos de texto, con un total de 12500 opiniones por subcarpeta(neg y pos), además de contar con una carpeta de opiniones sin etiquetar, pero que en el desarrollo de este modelo no serán necesarias. Dicho dataset está destinado para aplicaciones de clasificación de sentimientos de forma binaria y está contenido en: https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gzdonde se puede descargar directamente.

Posteriormente, los datos recopilados con twint serán utilizados para probar el modelo, con diferentes temáticas y fechas, sin importar el usuario y el contenido mismo del tweet.

3.2. Herramientas

Colaboratory, también llamado "Colab", te permite ejecutar y programar en Python en tu navegador con las siguientes ventajas:

- No requiere configuración
- Da acceso gratuito a GPUs
- Permite compartir contenido fácilmente

“Colab” puede facilitar tu trabajo, ya seas estudiante, científico de datos o investigador de IA.

Twint.

Es una herramienta escrita en Python de scraping de Twitter, que permite obtener tweets sin la necesidad de recurrir a la API de twitter. Twint utiliza los operadores de búsqueda de Twitter para poder hacer el scraping de Tweets de usuarios específicos, Tweets relacionados con ciertos temas, hashtags y tendencias.

En este proyecto se utilizó para poder armar un dataset de diferentes tweets sin importar el usuario, solamente la fecha y el tema que tuviera que ver con palabras que representaran algo negativo o positivo y únicamente en inglés, ya que como más adelante se explicara, otros datasets fuera de twint están en inglés. Haremos tantos sean necesarios, ya que estos serán utilizados principalmente para hacer las pruebas de nuestro modelo, por lo que los cambios de dataset no representan un cambio al sistema.

Para poder hacer uso de dicha herramienta(después de instalarla) hay que configurar un objeto que es el que recibirá los tweets.

Módulo os

Permite acceder a funcionalidades dependientes del Sistema Operativo, en este caso fue ocupado para el manejo de directorios a través de sus diversos métodos. Fue ocupado con el fin de manejar un dataset descargado acerca de opiniones de películas IMDB. Gracias a este módulo el paquete fue descomprimido, revisar contenido y eliminar carpetas, para poder guardar su contenido en distintas variables.

TensorFlow

Es una plataforma de código abierto de extremo a extremo para el aprendizaje automático, desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. Maneja arrays multidimensionales que son llamados "tensores". Permite implementar el cálculo a una o más CPU o GPU en equipos de escritorio, servidores o dispositivos móviles con una sola API.

Numpy

Es una biblioteca de Python que da soporte para crear vectores y matrices de grandes datos multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas.

Keras

Se trata de una biblioteca de Redes Neuronales de código abierto escrita en Python y que es capaz de ejecutarse sobre TensorFlow. Funciona a nivel de modelo: proporciona bloques modulares sobre los que se pueden desarrollar modelos complejos de aprendizaje profundo. A diferencia de los frameworks, este software de código abierto no se utiliza para operaciones sencillas de bajo nivel, sino que utiliza las bibliotecas de los frameworks de aprendizaje automático vinculadas, que en cierto modo actúan como un motor de backend para Keras. Las capas de la red neuronal que se quieren configurar se relacionan entre sí de acuerdo con el principio modular, sin que el usuario de Keras tenga que comprender o controlar directamente el propio backend del framework elegido.

tf.keras.preprocessing.text_dataset_from_directory: Genera un tf.data.Dataset a partir de los archivos de texto contenidos en un directorio.

tf.strings.regex_replace: Reemplaza elementos de entrada por medio de expresiones regulares coincidentes entre pattern con rewrite.

TextVectorization: Esta capa tiene opciones básicas para administrar texto en un modelo de Keras. Transforma un lote de cadenas (una muestra = una cadena) en una lista de

índices de tokens (una muestra = tensor 1D de índices de tokens enteros) o una representación densa (una muestra = tensor 1D de valores flotantes que representan datos sobre los tokens de la muestra).

tf.data.AUTOTUNE: La API tf.data proporciona la transformación tf.data.Dataset.prefetch . Se puede utilizar para desacoplar el momento en que se producen los datos del momento en que se consumen. En particular, la transformación utiliza un subproceso en segundo plano y un búfer interno para obtener elementos del conjunto de datos de entrada antes de que se soliciten. El número de elementos para captar previamente debe ser igual (o posiblemente mayor que) el número de lotes consumidos por un solo paso de entrenamiento, dicho valor puede ser cambiado por este método.

tf.keras.Sequential: Es un modelo Sequential, que es apropiado para una pila simple de capas donde cada capa tiene exactamente un tensor de entrada y un tensor de salida, en donde sus capas son accesibles a través del atributo de layers, dichos layers, oor lo general, necesitan conocer la forma de sus entradas para poder crear sus pesos.

3.3. Descripción del experimento

Los pasos a desarrollar para realizar el modelo de lenguaje neuronal serán los siguientes:

Es necesario importar todas las bibliotecas que se van a ocupar a lo largo del experimento(contenidas en el archivo ipynb)

Se necesita configurar el objeto twint con el que vamos a hacer el scrapping de los tweets de acuerdo a una fecha y a un tema en específico. Como la herramienta regresa columnas que no necesitamos(ya que solo nos interesan los tweets) como usuario, ubicación, nombre, etc., es necesario eliminar dichas columnas e incluso los tweets que puedan venir repetidos y que solo estén escritos en inglés. La configuración es la siguiente:

```
def create_twint_object(target, date="2021-01-21", limit=100):
  c = twint.Config()
  c.Store_object = True
  c.Pandas =True
  c.Search = target
  c.Since = date
  c.Limit = limit
  c.Lang = 'en'
  twint.run.Search(c)
  tweets = twint.storage.panda.Tweets_df.drop_duplicates(subset=['id'])
  tweets.index = range(len(tweets))
  tweets['lang'] = tweets['tweet'].map(lambda t: detect_lang(t))
  tweets = tweets[tweets.lang == 'en']
```

.

Por ejemplo, si queremos buscar tweets con el tópico de “capitolio”, se haría uso de la función anterior función:

```
corpus_capitol = create_twint_object("#Capitol", "2021-01-21", 100)
```

En donde se observa que se tiene un tópico, una fecha y una cantidad de tweets a buscar(3200 máximo por cada búsqueda).

Estos tweets serán nuestro dataset para probar en nuestro modelo. A continuación se mencionan los pasos para elaborar el modelo.

Necesitamos hacer uso de otro dataset para entrenar, validar y probar nuestro modelo. Para ello se hace uso de un dataset ya creado, este se encuentra en la dirección https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz y se trata de opiniones de películas en IMDB divididos en dos carpetas: test y train. La primera de ella nos trae dos carpetas mas: pos y neg. La segunda de ella contiene tres: pos, neg y unsup. Cada una contiene archivos de texto con las opiniones.

Debido a que estamos desarrollando el modelo en google collab, necesitamos hacer uso del módulo os para poder manipular dicho archivo como si estuviéramos directamente en el sistema operativo, además es necesario eliminar la carpeta de unsup ya que no es necesaria para el modelo, aunado a que si la dejamos el modelo podría tomar sus valores y tener una exactitud más baja. Para hacer todo esto y almacenarlo en el caché, se necesitan las líneas siguientes:

```
dataset = tf.keras.utils.get_file("aclImdb_v1.tar.gz", url,
                                  untar=True, cache_dir='.',
                                  cache_subdir='')
```

```
train_dir = os.path.join(dataset_dir, 'train')
os.listdir(train_dir)
```

Se dividirán los documentos en tres conjuntos, el primero para entrenamiento, el segundo para validación y el tercero para test. También se agrega una semilla para el generador de números aleatorios y así evitar que se utilice la misma reseña para ambos conjuntos, otra opción sería agregar el parámetro shuffle como False para evitar el mismo problema, sin embargo en este caso la generación de los conjuntos será secuencial.

Se hará 20% para validación y el 80% para entrenamiento de los datos contenidos en la carpeta train(cada carpeta contiene 12500 archivos). Para separar los documentos con keras se necesita del siguiente módulo:

```
batch_size = 32
seed = 42

raw_train_ds = tf.keras.preprocessing.text_dataset_from_directory(
    'aclImdb/train',
    batch_size=batch_size,
    validation_split=0.2,
    subset='training',
    seed=seed)
```

Es importante observar que el parámetro subset nos da el tipo de clasificación que se hará para el dataset, en este caso para el entrenamiento, en donde estos serán divididos en dos clases: 0 y 1 o neg y pos. Estas clases se definen gracias a que el módulo clasifica de acuerdo a las carpetas que encuentre dentro del directorio. Se define una semilla para que

los datos no se repitan y se dividen en lotes, para este caso genera 625 lotes, es decir 625 x 32 batch = 20,000 que es el 80% de los datos.

Si quisiéramos ver cómo es que hasta este momento ya les ha asignado una etiqueta de acuerdo a la clase en donde fueron encontrados, obtenemos:

```
for text_batch, label_batch in raw_train_ds.take(1):
  for i in range(5):
    print("Reseña", text_batch.numpy()[i])
    print("Etiqueta", label_batch.numpy()[i])
```

```
Reseña b'"Pandemonium" is a horror movie spoof that comes off more stupid than funny. Beli
Etiqueta 0
Reseña b"David Mamet is a very interesting and a very un-equal director. His first movie '
Etiqueta 0
Reseña b'Great documentary about the lives of NY firefighters during the worst terrorist a
Etiqueta 1
Reseña b"It's boggles the mind how this movie was nominated for seven Oscars and won one.
Etiqueta 0
```

El 20% de los datos restantes aún hay que asignarlos a otro grupo, para eso ocupamos la misma función pero cambiando el parámetro subset por validación.

```
raw_val_ds = tf.keras.preprocessing.text_dataset_from_directory(
    'aclImdb/train',
    batch_size=batch_size,
    validation_split=0.2,
    subset='validation',
    seed=seed)
```

Como se mencionó anteriormente, haremos uso de tres grupos. Hasta este momento hemos hecho dos con los datos de la carpeta train, la carpeta test se ocupará enteramente para el último grupo.

```
raw_test_ds = tf.keras.preprocessing.text_dataset_from_directory(
    'aclImdb/test',
    batch_size=batch_size)
```

Se agrega una función para el preprocesamiento de los datos, esta realizará la conversión a minúsculas, removerá las etiquetas del html y finalmente hará la tokenización usando expresiones regulares. Esto con el fin de que podamos vectorizar en números enteros los datasets.

```
def custom_standardization(input_data):
  lowercase = tf.strings.lower(input_data) #convierte todo a minisculas
  stripped_html = tf.strings.regex_replace(lowercase, '<br />', ' ') #reemplaza los html por un espacio
  return tf.strings.regex_replace(stripped_html,
                                  '[%s]' % re.escape(string.punctuation),
                                  '') #los cadenas ya sin html ahora se le quitan los simbolos de puntuacion,
```

El texto se deberá vectorizar, esto se realiza utilizando la herramienta de TextVectorization de keras.

Dentro de los argumentos en standarize se pasa como parámetro la función creada anteriormente para que al vectorizar los conjuntos que se han creado tengan el mismo preprocesamiento.

Ahora que tenemos nuestra estandarización personalizada, podemos crear una instancia de nuestra capa de vectorización de texto. Estamos usando esta capa para normalizar, dividir y mapear cadenas a números enteros, por lo que configuramos nuestro 'output_mode' en 'int'. Tenga en cuenta que estamos usando la función de división predeterminada y la estandarización personalizada definida anteriormente. También establecemos una longitud de secuencia máxima explícita, ya que las CNN más adelante en nuestro modelo no admitirán secuencias irregulares.

```
vectorize_layer = TextVectorization(
    standardize=custom_standardization,
    max_tokens=max_features, #cantidad maxima del vocabulario en la capa
    output_mode='int', #cada token recibirá un numero entero
    output_sequence_length=sequence_length)
```

Lo siguiente será realizar un diccionario indexado, esto quitará las etiquetas a los datos y después se utilizará el método adapt para ajustar el estado de la capa de preprocesamiento al nuevo conjunto de datos, esto ya estarán convertidos a números enteros.

```
train_text = raw_train_ds.map(lambda x, y: x) #quita las etiquetas de 0 y 1 a los datos
vectorize_layer.adapt(train_text) #primer ajuste a nuevos datos,
```

Se define una función para la vectorización del texto, esta función devuelve una capa de vectorización. Esto nos servirá para poder visualizar los datos convertidos a números enteros.

```
def vectorize_text(text, label):
  text = tf.expand_dims(text, -1)
  return vectorize_layer(text), label
```

```
text_batch, label_batch = next(iter(raw_train_ds))
first_review, first_label = text_batch[0], label_batch[0]
print("Review", first_review)
print("Label", raw_train_ds.class_names[first_label])
print("Vectorized review", vectorize_text(first_review, first_label)) #regresa la oracion ya vectorizada
```

```
Review tf.Tensor(b'"A young woman suffers from the delusion that she is a werewolf, based upon a family le
Label neg
Vectorized review (<tf.Tensor: shape=(1, 250), dtype=int64, numpy=
array([[   4,  181,  246, 2320,   35,    2,    1,   12,   55,    7,    4,
        1823,  443,  718,    4,  215, 1725,    5,   33,    1, 3420,    5,
           3,  546,   15, 8957,  108,   28,  684,    6,   39,  491, 2217,
          32,  348,   55, 4096,    2, 4169,    1,    3,  846,    2,  348,
          55,  879, 1387,    8,  115,   16,    4,  236,  130,   39,  116,
         723,    6,  190,    4,  459,   15,    2,  122,   51,   55,    7,
        3377,    3,   39, 1522,    7,  546,   32,    4, 1090,    5, 3616,
        8981,  169,   32,  129, 2379,  650,    2,  246, 1734,    6,   39,
        1097,  760,    3, 4972, 1087,   20,    2, 3616, 1668,    6,    2,
         287,    1, 3658,    1,    1,    1,  990,    1,    1,  762,   16,
         374, 7652,    1,    1,  833,   32,    1, 1941,    1,    1,   36,
          14,    1,    1,    1, 2101,   55,    7,    4, 1823,    2,    1,
         170,    7,   51,    2,  379, 3898,  985,  973,  188,   39, 2988,
           2,  634, 2721, 2684,   39, 8463,    1,   13,  235,    4,    1,
        1612, 1474,    1,    7,  460,   21,    4, 1823,   55,    7, 1516,
           4,   52,  556, 6830,   14,    4,   19, 1823,  246,    8,  648,
          59,   25,   74,  122,   45,    1,  106,   62,   13,    4,  638,
        1823,   16,   39,  845,    1,    4,   86,  221,    5, 3414,   18,
          14,  231,   14, 2311, 2595,    3,  528, 1427,  137,   11,   28,
           7,  264,    6, 1638,    1,    1,    7,  338, 2613,   41,    1,
        8134,  458,    2,   19,   60,  915,    4, 1234, 1137,    5, 2135,
         594, 5842,    1,    1,   14,    1,    3, 8456, 2274, 5645,   14,
           1,   23,   49,  679, 1787,    0,    0,    0]])>, <tf.Tensor: shape=(), dtype=int32, numpy=0>)
```

Observando el resultado de esta capa podemos observar que a cada palabra se le asigna un número entero. Por ejemplo, el número 2 le corresponde a la palabra the, y a lo largo del array se observa que se repite el número, esto significa que volvió a encontrar la misma palabra en el documento.

Es necesario aplicar esta capa a los tres conjuntos, y no a solo un documento como se hizo anteriormente.

```
train_ds = raw_train_ds.map(vectorize_text)
val_ds = raw_val_ds.map(vectorize_text)
test_ds = raw_test_ds.map(vectorize_text)
```

Se optimizará el modelo agregando las opciones .cache para que los datos se mantengan en memoria y la opción .prefetch que permite que mientras se ejecuten las operaciones de un dato se esté leyendo el dato siguiente.

```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

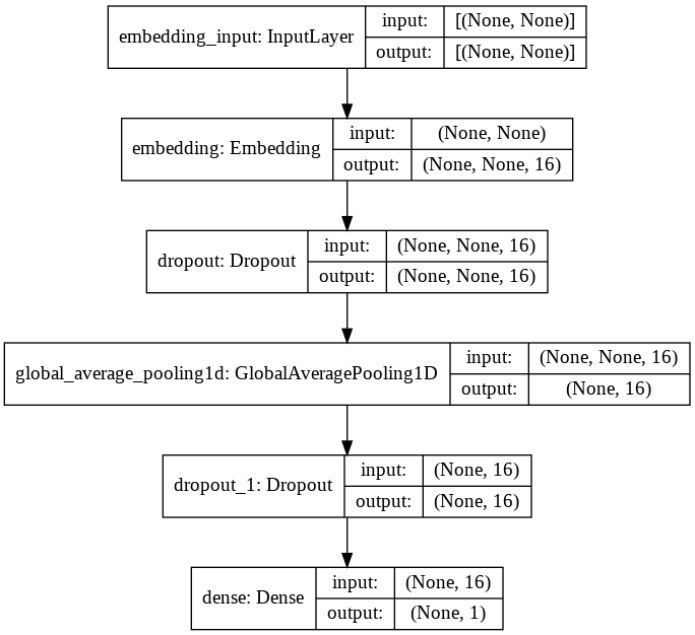
Se procede a crear el modelo, este será un modelo secuencial, con las capas de embedding, que toma las entradas de texto ya convertidas a enteros, es decir, convierte números enteros positivos (índices) en vectores densos de tamaño fijo y con salida (batch_size, input_length, output_dim). Una capa de pooling para trabajar con vectores de tamaño constante con salida (batch_size, features), las capas de dropout automáticamente se consideran para el entrenamiento nada más, a menos que se especifique explícitamente lo contrario, toma un flotante entre 0 y 1, ayuda a evitar el sobreajuste lo que significa que una de cada 5 entradas se excluirá aleatoriamente de cada ciclo de actualización , finalmente la capa Dense del modelo es agregada al final para obtener una salida.

```
embedding_dim = 16

model = tf.keras.Sequential([
  layers.Embedding(max_features + 1, embedding_dim),
  layers.Dropout(0.2),
  layers.GlobalAveragePooling1D(),
  layers.Dropout(0.2),
  layers.Dense(1)])

model.summary()
```

Viendo en diagrama de bloque se tiene el modelo secuencial de tres capas de la forma:



Se compila el modelo, se agrega un optimizador ADAM (Para optimización del learning rate), con las métricas para clasificación binaria ya que solo contamos con dos etiquetas(0 y 1), dichas métricas son las que el modelo evaluará durante el entrenamiento y las pruebas, como función de pérdida se usará la entropía cruzada binaria de keras que calcula la entropía cruzada entre etiquetas verdaderas y predichas. Esto corresponde a una configuración del modelo.

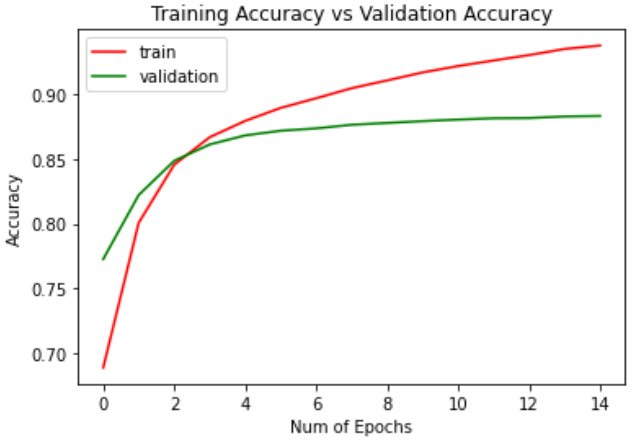
```
model.compile(loss=losses.BinaryCrossentropy(from_logits=True),
              optimizer='adam',
              metrics=tf.metrics.BinaryAccuracy(threshold=0.0))
```

Una vez configurado el modelo es momento de entrenarlo con los datos de training y con 15 épocas, estas pueden variar, pero este número nos ha dado un buen resultado.

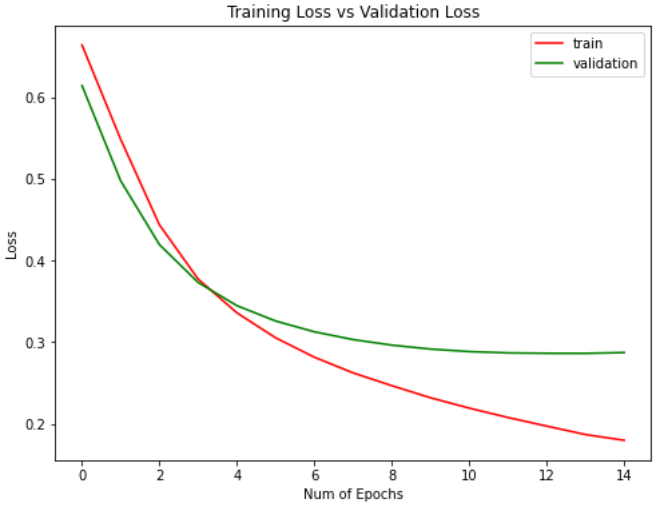
```
epochs = 15 #iteracion de los datos
history = model.fit( #entrena el modelo
    train_ds, #datos de entramiento
    validation_data=val_ds,#datos para evaluar la metrica de la compilacion
    epochs=epochs)
```

Si graficamos el valor retornado que nos da obtenemos las métricas de la función de pérdida y de precisión. Para la exactitud de los datos de entrenamiento con los de

validación observamos que hay un ligero sobreajuste debido a que el entrenamiento tienen valores mayores que los de validación. Además observamos que el modelo es bueno, ya que conforme transcurren las épocas adquiere una mayor exactitud en ambos datasets.



Para la pérdida, de los datos de entrenamiento con los de validación, observamos que conforme avanzan las épocas hay menos pérdida.



Ahora vamos a aplicar este modelo a nuestra capa de TextVectorization, ya que ahora tenemos ya pesos que acabamos de entrenar.

```
export_model = tf.keras.Sequential([
  vectorize_layer,
  model,
  layers.Activation('sigmoid')
])

export_model.compile(
    loss=losses.BinaryCrossentropy(from_logits=False), optimizer="adam", metrics=['accuracy']
)

loss, accuracy = export_model.evaluate(raw_test_ds)
print(accuracy)
```

Si ahora vemos la precisión de nuestro modelo.

```
782/782 [==============================] - 11s 14ms/step - loss: 0.3157 - accuracy: 0.8713
0.8736400008201599
```

Observamos que tenemos un 87% de precisión conforme a los datos de test que hasta el momento no habíamos ocupado, lo que lo hace un buen modelo.

Finalmente se crea una pequeña función para presentar la etiqueta de cada

```
def print_predicted(predicted, corpus):
  for t in range(len(predicted)):
    print("Sample number {} of {} (Twitter Set)\n".format(t, len(corpus)-1))
    if t == 0 :print("{:10}".format("Etiqueta"))
    print("{:3}:\t{}\n".format(predicted[t], corpus[t]))
```

```
def resume_opinions(predicted_values):
  count_ones, count_zeros = 0, 0
  labels = []
  for tweet in predicted_values:
    #print(tweet)
    if tweet > 0.4999:
      labels.append(1)
      count_ones += 1
    else:
      labels.append(0)
      count_zeros += 1

  if count_ones > count_zeros:
    print("Hay más reseñas positvas obre el capitoilio que negativas")
  elif count_ones == count_zeros:
    print("Las opiniones son iguales")
  else:
    print("Las opiniones son malas")
  return labels
```

3.4. Presentación de Resultados

Ya con nuestro modelo entrenado, procedemos a ocupar los datasets generados por twint como entrada a nuestro modelo.

```
labels_capitol = resume_opinions(tweets_predicted)
print_predicted(labels_capitol, corpus_capitol)
```

Obteniendo como resultado(solo una parte):

```
Hay más reseñas positvas obre el capitoilio que negativas
Sample number 0 of 187 (Twitter Set)

Etiqueta
  0:    Anyone else uncomfortable with @GOP nutjobs bringing guns into the #Capitol? I wouldn't trust someone like @laurenboebert or @mtgreenee with a gun ar

Sample number 1 of 187 (Twitter Set)

  0:    @TheView @MeghanMcCain #RepublicanTerrorists attacked OUR #Capitol   Tag who the #terrorist #Republicans are if they don't want to all be labeled #t

Sample number 2 of 187 (Twitter Set)

  1:    @Newsweek "#WhiteHouse Cleanup After Donald #Trump's Departure Cost at Least $127,000. With 132 rooms, 35 bathrooms, 412 doors and 28 fireplaces the

Sample number 3 of 187 (Twitter Set)

  0:    The question re #CNN isn't why do the retain viewership,  but the more clinical,  why do they retain any advertisers?  @Adweek #Trump #Biden @GOP #dr

Sample number 4 of 187 (Twitter Set)

  1:    You can steal a laptop from the #CapitolRiots and try to sell it to the Russians and get sent home arrest to your mom if you have #WhitePrivilege  Ye

Sample number 5 of 187 (Twitter Set)

  1:    Florida man accused of being in #Capitol riots was arrested at the inauguration, Justice Department says   https://t.co/inzjcVVbr0
```

Podemos observar que la etiqueta se refiere a si es una reseña positiva o negativa(0 y 1 respectivamente). Si leemos el tweets, por ejemplo el Sample 1, podemos notar que habla sobre terroristas y ataques en una postura molesta, por lo que la etiqueta es correcta.

Elaboremos una pequeña tabla con las oraciones en concreto de prueba con su etiquetado obtenido.

Tweet	**Etiqueta**	**¿Es correcto?**
You can steal a laptop from the #CapitolRiots and try to sell it to the Russians and get sent home arrest to your mom if you have #WhitePrivilege Yep 👍🟫 we are not the same.	1 (positivo)	No, utiliza el sarcasmo por lo que hace un mal etiquetado.
Also concerned if our soldiers can't handle a heated garage in the #capitol, how on earth will they handle going to #war, sleeping in the dirt?	0(negativo)	Sí, maneja una postura en desacuerdo con el capitolio.
Wixom man charged with attacking officers during failed #SeditiousGOP #insurrection attempt on US #Capitol	0(negativo)	Sí, a pesar de que no maneja ninguna postura y sólo informa un acontecimiento, este es de contexto negativo.
OU LOUSY #SEDITIOUS SNAKE. We will NEVER FORGET what you did. We will NEVER FORGET the part YOU played in the #SIEGE on our nation's #CAPITOL, in the	1(positivo)	No, es completamente una postura negativa y lo clasificó incorrectamente.

#MURDER of a police officer. Run & Hide, you #TRAITOR. Better yet, #RESIGN.		

Vamos a hacer otro dataset distinto al capitolio, en este caso buscaremos los tweets que contengan la palabra excelente.

```
corpus_congrats = create_twint_object("excellent", "2021-01-20", 100)
```

```
Sample number 0 of 92 (Twitter Set)

Etiqueta
  1:    @therecount They always have expectations so High for anyone but the dotard. He shouldn’t be the barometer of where we aim however it is unreasonable

Sample number 1 of 92 (Twitter Set)

  1:    The Biden Administration is off an excellent start destroying over 50,000 jobs in less than 48 hours. Who is having buyers remorse, be honest?   #Not

Sample number 2 of 92 (Twitter Set)

  1:    @SDuncovered Excellent!

Sample number 3 of 92 (Twitter Set)

  1:    This is excellent!

Sample number 4 of 92 (Twitter Set)

  1:    @ReesusP @SenTedCruz My union needs to consider that as our work slows and it’s slowing enough that people are retiring at 10 years. The welding skil

Sample number 5 of 92 (Twitter Set)

  1:    @BethWinsor1 @ldacompton @nslev116 @Martinez_LCPS @LCPSOfficial This is awesome!  Well-deserved for such a selfless and excellent principal. 👏

Sample number 6 of 92 (Twitter Set)

  1:    langesshop with a set of our Stainless Steel Stone shields! These are an excellent way to achieve that nostalgic look on your old or new Squarebody.
```

Tomemos como ejemplo al Sample 1 y 3. En este último vemos un etiquetado correcto al tratarse de una oración corta que menciona que algo es excelente. En cambio, el Sample 1 hace un etiquetado incorrecto al no ser capaz de distinguir el sarcasmo, ya que si bien tiene la palabra excelente, la oración nos habla de algo negativo.

Elaboremos una tabla similar.

Tweet	Etiqueta	¿Es correcto?
The Biden Administration is off an excellent start destroying over 50,000 jobs in less than 48 hours. Who is having buyers remorse, be honest? #NotMyPresident #ImpeachBidenNow #BidenHarrisInauguration #BidenCrimeFamily #HunterBiden #GreatReset #NewWorldOrder #NewAccount #Trump	1(positivo)	No, la postura es negativa conforme al gobierno de Biden.
Excellent observation. It's	1(positiva)	Sí, tiene una postura de

so critical for this kind of duplicity to be seen and shared. Thank you!		agradecimiento sin sarcasmo,
This is a great and excellent project and also the reward is great. Thanks for shared this wonderful opportunity. Best wishes for all team members. Keep going and get to the success #YearnyfiNetwork #yearnyfi #defi #ERC20 #eth #presale #ico #ieo #sale	1(positiva)	Sí, nuevamente es una postura de agradecimiento y elogio.
all excellent points - and no, none of that has been considered ... ZERO educators were included. The Director of Schools in Maryville expressed opposition, but the legislature won't hear it	0(negativa)	Sí, ya que se muestra una postura de inconformidad de alguna situación.

3.5. Evaluación del modelo

Una vez teniendo nuestro modelo entrenado y a la evaluación propia de keras podemos obtener el porcentaje de precisión de nuestro modelo, que nos muestra que tiene una precisión del 87%, lo cual significa que es un modelo bueno, con una pérdida baja para los datos con los que fue entrenado.

```
loss, accuracy = export_model.evaluate(raw_test_ds)
print(accuracy)
```

```
782/782 [==============================] - 11s 14ms/step - loss: 0.3157 - accuracy: 0.8713
0.8736400008201599
```

Este resultado es obtenido gracias a nuestro modelo secuencial de keras, ya que, a pesar de las distintas capas de la red, tenemos una configuración previa. Dicha configuración se da por una entropía cruzada binaria dada por la función:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Aunque si bien se ocupa este tipo de entropía, esta no es calculada por alguna función que hayamos hecho, es directamente implementada por keras.
También se necesita que los pesos de la matriz de la red neuronales sean optimizados, para ello ocupamos el optimizador adam, que de igual forma ya se encuentra implementado en keras y que se describe como:

$$m = \beta_1 * m + (1 - \beta_1) * \Delta W$$
$$v = \beta_2 * v + (1 - \beta_2) * \Delta W^2$$
$$W = W - \frac{\alpha * m}{\sqrt{v + \varepsilon}}$$

Por último se necesita una métrica en la precisión del modelo, para esto se utilizó una metrica BynaryAcurracy. Esta métrica crea dos variables locales, totaly countque se utilizan para calcular la frecuencia con la que y_pred coincide con y_true. Esta frecuencia se devuelve finalmente como binary accuracy, que es una operación idempotente que simplemente divide total por count.

4. Conclusiones

Los textos clasificados o etiquetados son poco comunes, así que en este caso solo se utilizó el dataset clasificado en 2 clases para entrenar el modelo y poder clasificar tweets.

Al crear un modelo que su primera capa limpia el texto y lo vectoriza permite un uso muy flexible del desarrollo, también es muy sencillo modificar los hiperparámetros, lo que hace que usar el modelo sea tan sencillo como pasarle una lista como argumento al método .predic el modelo.

Elaborar un modelo con keras resulta bastante sencillo, en comparación a elaborar un modelo con la arquitectura de Bengio, ya que este no contiene algún soporte o desarrollo como lo tiene keras y tensorflow, por lo que el desarrollo de capas de la red neuronal queda a total implementacion del usuario, que si no es realizada correctamente puede ser un modelo erróneo que nos arroje resultados poco eficientes.

Se observa que el modelo es capaz de clasificar los tweets en positivo y negativo, como se vio en la presentación de los resultados, pero tiene un gran problema, no es capaz de diferenciar el sarcasmo, ya que si el tweet contiene palabras positivas usadas en un contexto negativo, lo clasificara como positivo. Esto es un problema, ya que en temas de controversia muchas personas suelen comentar sarcásticamente, lo que hará que el modelo sea bastante ineficiente, de forma que hará malos etiquetados.

En otros tweets que no tienen sarcasmo pueden llegar a haber malas clasificaciones, esto se podría mejorar moviendo algunos parámetros de la red, o mejor aún teniendo más datos para entrenamiento porque tal vez las reseñas que teníamos contenían muchas palabras repetidas, entonces en el momento de entrenar no eran suficientes datos. En tweets que contengan groserías no es capaz de clasificar correctamente, ya que el conjunto de entrenamiento no las contiene, pasa lo mismo que con el sarcasmo.

Aunque el modelo nos indique un 87% de precisión y parezca un buen modelo, esto no es del todo correcto por los resultados obtenidos, por lo que dicho valor parece un poco engañoso conforme al lenguaje que se suele utilizar en twitter. Si este modelo se aplicará a un conjunto de datos en donde las personas no hagan uso(o muy poco) del sarcasmo, entonces podemos ver el porcentaje como correcto.

Sin embargo, tomando en cuenta los datos que poseemos, las reseñas usadas como entrenamiento y el objetivos de la clasificación, fue un muy buen resultado, un poco más de aceptable, esto es debido a que los críticos serios de cine suelen no usar sarcasmo, o en caso de usarlo, utilizan más palabras para evitar que se mal entiendan sus palabras. En el caso de los objetivos de la clasificación, al ser personas comunes y, en muchos casos, no muy hábiles con el uso de las palabras para diferenciar el sarcasmo de una opinión sincera, hubo una gran discrepancia entre el resultado deseado y el obtenido.

Para un trabajo a futuro, como primera parte, será tener un conjunto de datos mucho mayor para poder entrenar de mejor manera el modelo, siempre y cuando estos datos ya se encuentren clasificados de forma binaria, ya que de otra manera será erróneo para nuestro modelo. Esto nos permitirá tener una mejor clasificación en los tweets, aunque seguirá presente el error de que si un tweet contiene una gran cantidad de palabras clasificadas como positivas utilizadas de forma sarcástica, el modelo lo clasificará incorrectamente. Este punto probablemente sea uno de los más difíciles de elaborar en este trabajo a futuro.

Bibliografía

1. Mnih, A. & Hinton, G. E. (2009). A scalable hierarchical distributed language model. *Advances in neural information processing systems* (p./pp. 1081--1088).
2. Mnihamnih, A., & Whye Teh, Y. (2012). A fast and simple algorithm for training neural probabilisticlanguage models [Conference].In Proceedings of the International Conference on Machine Learning. https://arxiv.org/ftp/arxiv/papers/1206/1206.6426.pdf
3. Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C., & Kandola, J., Hofmann, T., Poggio, T., & Shawe-Taylor, J. (Eds.). (2003). A neural probabilistic language model. *Journal of Machine Learning Research, 3*(6), 1137–1155. https://doi.org/10.1162/153244303322533223
4. Hugo Galán Asensio. Inteligencia artificial.Redes neuronales y aplicaciones. Universidad Carlos III de Madrid. http://www.it.uc3m.es/jvillena/irc/practicas/10-11/06mem.pdf
5. Carlos Alberto Olarte Vega. Modelos de lenguaje. http://atlas.puj.edu.co/~caolarte/puj/cursos/semI/slides/Modelos_del_Lenguaje.pdf

6. Carlos Gomez-Rodriguez, Jesús Vilares. Segmentación de palabras en español mediante modelos del lenguaje basados en redes neuronales. Procesamiento del Lenguaje Natural, Revista nº 57, septiembre de 2016, págs. 75-82. https://core.ac.uk/download/pdf/85003514.pdf
7. Google Colab: https://colab.research.google.com/
8. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text_dataset_from_directory
9. https://www.tensorflow.org/api_docs/python/tf/strings/regex_replace
10. https://towardsdatascience.com/you-should-try-the-new-tensorflows-textvectorization-layer-a80b3c6b00ee
11. https://www.tensorflow.org/guide/data_performance
12. https://www.tensorflow.org/guide/keras/sequential_model
13. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a