Análisis de sentimientos de opiniones en redes sociales usando técnicas de procesamiento del lenguaje natural

Universidad de Belgrano -2021

Alumna: Julia Scotto

Matricula: 50211626

Director: Dr. Alejandro Mitaritonna

Indice

Indice de Tablas

Indice de Figuras

Resumen

Hoy en día al adquirir cualquier producto o servicio nos dirigimos a internet para ver opiniones previas de usuarios que ya tuvieron experiencias con estos. Todas estas opiniones subjetivas representan **datos** de mucha importancia para las empresas, ya que con este feedback pueden mejorar o redireccionar su empresa.

En este contexto, existen muchos sistemas o software pago para hacer estos análisis de opiniones de los consumidores. En el presente trabajo se realizará un análisis de las librerías más populares en el lenguaje más popular para trabajar en el campo de la Inteligencia Artificial.

Se evalúa resultados y eficacia para el desarrollo de un prototipo que utiliza la API de Twitter [45] y Reddit [46]. En la línea de comandos se podrá elegir qué API usar. En el caso de Reddit la API buscará comentarios (limitando a 10) en el posteo más controversial del mes del subreddit /r/politics. Para Twitter se tendrá que brindar de input una palabra clave a buscar y se listará los primeros 10 tweets. Con el listado devuelto por la API elegida se resolverá por medio de un modelo, desarrollado y entrenado, un análisis de sentimientos, finalmente clasificando las opiniones en positivas o negativas Para esto se analizará las diferentes arquitecturas y algoritmos de Machine Learning/Deep Learning que se utilizan actualmente [2] para resolver problemas de procesamiento de lenguaje natural aplicado al análisis de sentimientos, demostrando de forma práctica la precisión de cada una. Una vez analizado y comparado, se arribará a conclusiones finales con el objetivo de dejar asentado cuáles técnicas son las apropiadas para futuros desarrollos de similares características a las planteadas en la presente tesina.

Abstract

Nowadays when acquiring any product or service we go to the internet to see previous opinions of users who have already had experiences with them. All these subjective opinions represent data of great importance for companies, since with this feedback they can improve or redirect their company.

In this context, there are many paid systems or software to do these consumer opinion analyzes. In the present work, an analysis of the most popular libraries in the most popular language to work in the field of Artificial Intelligence will be carried out.

Results and effectiveness are evaluated for the development of a prototype that uses the Twitter API [45] and Reddit [46]. On the command line you can choose which API to use. In the case of Reddit, the API will look for comments (limited to 10) on the most controversial post of the month on the / r / politics subreddit. For Twitter, you will have to input a specific word to search and the first 10 tweets will be listed. With the list returned by the chosen API, it will be resolved by means of a model, previously developed and trained, a sentiment analysis, finally classifying the opinions as positive or negative.For this, the different Machine Learning / Deep Learning architectures and algorithms that are currently used [2] to solve natural language processing problems applied to sentiment analysis will be analyzed, demonstrating in a practical way the precision of each one. Once analyzed and compared, final conclusions will be reached in order to establish which techniques are appropriate for future developments with similar characteristics to those proposed in this thesis.

Introduccion

1. Contexto del problema

El análisis de sentimientos o la minería de opiniones es el estudio computacional de las opiniones, sentimientos, emociones, evaluaciones y actitudes de las personas hacia entidades como productos, servicios, organizaciones, individuos, problemas, eventos, temas y sus atributos.

El interés en este campo coincide con el crecimiento masivo de datos provistos por las redes sociales e internet en general. ¿Pero qué son estos datos? Son opiniones, críticas, puntos de vista o perspectivas de algún producto, servicio, marca, política o cualquier otro tema que la gente pueda expresar alguna opinión.

Las opiniones son fundamentales para casi todas las actividades humanas y son influyentes clave de nuestros comportamientos. Nuestras creencias y percepciones de la realidad, y las elecciones que hacemos, están, en gran medida, condicionadas a cómo otros ven y evalúan el mundo [15] . Por esta razón, cada vez que necesitamos tomar una decisión, a menudo buscamos las opiniones de los demás. Esto no solo es cierto para las personas sino también para las organizaciones.

Actualmente, cuando uno quiere adquirir un producto o utilizar un servicio, ya no solo pide opiniones a amigos u otros cercanos. En lugar de eso, uno consulta entre todas las reseñas sobre usuarios que ya tuvieron experiencias con dicho producto o servicio. Y de la misma forma, una organización ya no consulta las opiniones por medio de encuestas, sino que directamente recopila toda esa información pública del internet [12].

De esta forma, surgieron también nuevas empresas que ofrecen sistemas automatizados de análisis de sentimientos para las organizaciones. El estado del arte actual proporciona numerosas técnicas para diversas tareas de análisis de sentimientos. Por lo tanto, actualmente, existen muchos métodos de Machine y Deep Learning para evaluar y analizar las opiniones, siendo algunas más eficientes que otras [13][4] .

2. Idea Directriz de la Tesina

Internet brinda a los usuarios un espacio para expresar sus opiniones sobre algo, esto al final representa información divulgada que las empresas no pueden controlar. Una encuesta hecha en 2019 por BrightLocal [1] revela que "*El 82% de los consumidores lee reseñas online de empresas locales, y el 52% de los jóvenes de entre 18 y 54 años dice que* ***siempre*** *lee reseñas*". Además la misma encuesta afirma que 'Las críticas positivas aumentan la probabilidad de que el 91% de los consumidores utilicen una empresa, mientras que el 82% se desanima por las críticas negativas'.

Por esa razón, si se desea comenzar un emprendimiento exitoso en el mercado actual, se necesita saber cómo estas opiniones pueden afectar.

La idea directriz de la tesina es no solo proporcionar un prototipo de análisis de sentimientos de opiniones que servirá para un emprendimiento sino también para un usuario que tal vez sea futuro cliente de ese emprendimiento o desea utilizar el prototipo con otro fin, por ejemplo, analizar opiniones sobre un partido político.

3. Hipotesis

El análisis de sentimientos basados en machine learning y deep learning cuentan con diferentes técnicas y dependiendo de su implementación y finalidad se alteran para obtener un mejor resultado. En este contexto, se plantea como hipótesis de trabajo principal que, mediante técnicas de minería de textos y análisis de sentimientos se puede construir un sistema automático que clasifique correctamente en negativas y no negativas las opiniones escritas en redes sociales.

4. Objetivos

a. Objetivo General

Realizar un análisis sobre las diferentes técnicas de procesamiento de lenguaje natural y análisis de las diferentes arquitecturas y algoritmos de Machine Learning/Deep Learning para finalmente realizar un

prototipo usando una de las técnicas analizadas para que el usuario final pueda recopilar y analizar sentimentalmente opiniones en las redes sociales.

b. Objetivos específicos

Los objetivos que se proponen cumplir en el presente trabajo son:

- Elegir las librerías para NLP para analizar y justificar su elección.
- Realizar un ejemplo práctico para cada una de las librerías.
- Analizar las arquitecturas y algoritmos de deep/machine learning y aplicar un ejemplo práctico.
- Elegir el método más apropiado para implementar en nuestro prototipo.
- Conclusión final sobre cada método y prototipo.

5. Metodologia

Se realiza una investigación bibliográfica sobre el estado del arte y las librerías utilizadas. Se investigará también, todas las documentación oficiales y sus aplicaciones. Luego de recopilar toda la información, se plasmará lo investigado y se hará el análisis de cada librería de NLP (NLTK, Gensim y Textblob), de cada librería exclusivas para el análisis de sentimiento (CoreNLP, Flair y VADER) y de cada red neuronal (CNN y LSTM). Luego, se demostrará su precisión con un ejemplo práctico de cada uno. Para finalizar se realizará un prototipo utilizando el método Kanban para gestionar el proyecto y por último se evaluará el método más adecuado eligiendo el que mejor se adapte al objetivo de la tesis.

6. Justificacion

Existen muchos métodos, librerías, frameworks para el análisis de sentimientos. Python es uno de los lenguajes más populares para trabajar en el campo de la Inteligencia Artificial y para abordar los problemas relacionados con el Procesamiento del Lenguaje Natural, Python nos proporciona diferentes librerías con diferentes métodos y resultados que pueden ser aplicados a diferentes problemas.

La idea principal es brindar un prototipo open-source para que cualquier usuario pueda realizar un análisis de sentimientos de un conjunto de opiniones en redes sociales por medio de un input (una marca, producto, servicio, tema, etc).

El objetivo principal es brindar la posibilidad a un usuario de acceder a este tipo de tecnología sin necesidad de pagar [4] o tener que instalar algún software. Para esto se realizará una investigación del tipo de arquitecturas y algoritmos dentro del campo de Machine/Deep Learning y se explicará cuál será la más apta para nuestro prototipo final.

7. Alcance

El proyecto se limita a analizar las arquitecturas y algoritmos actuales [5] y que hayan arrojado una alta precisión (mayor al 80%) en el desarrollo de modelos similares al propuesto en la presente tesina. El proyecto además se limita a encontrar una solución que se pueda adaptar fácilmente al prototipo de línea de comandos.

8. Organización del Documento

La elaboración del presente documento está dividido en 5 capítulos:

En la *Introducción* se presenta el contexto del problema, la hipótesis de la tesina y objetivos. Además explica la metodología de trabajo y su alcance

En el *Marco Teórico* se explica el estado del arte actual y los conceptos fundamentales para poder entender la tesina.

En el *Análisis/Comparación* se realiza la experimentación práctica con todas las librerías y redes neuronales. Se muestran los resultados de cada experimentación y luego lo comparamos. Finalmente se elige un modelo a utilizar para el prototipo

En el *Diseño del prototipo* se presenta el conjunto de datos a utilizar y se desarrolla en modelo de redes neuronales. Entrenamos el modelo y desarrollamos el front-end del prototipo. Además se presenta el diagrama de casos de usos con sus secuencias escritas.

En las *Conclusiones* se contemplan las conclusiones finales y cuáles son las futuras líneas de investigación y desarrollo para mejorar el prototipo

Marco Teorico

1. Antecedentes

Actualmente el crecimiento de las redes sociales como Twitter, Instagram, Reddit, etc, permite el intercambio continuo de opiniones generando un valor (positivo o negativo) sobre productos, organizaciones, servicios, películas, individuos, eventos políticos, temas específicos, etc. Estas opiniones pueden ayudar a mejorar la calidad de dicho producto o servicio de una organización. En este contexto, el uso de **técnicas de procesamiento del lenguaje natural** (*Natural Language Processing, NLP*) es muy útil para analizar esta información masiva. En particular el **Análisis de Sentimiento** es una técnica cada vez más usada, cuyo objetivo es la clasificación de opiniones y sentimientos expresados en un texto, generado por una parte humana. Los primeros enfoques en el análisis de sentimientos se basaban en técnicas de ***Machine Learning***. En el trabajo realizado por Bo Pang y Lillian Lee en el 2002 [8] investigaron varias técnicas de machine learning (Naive Bayes (NB), Maximum Entropy (ME), y Support Vector Machines (SVM)) usando como datos opiniones de películas. En su trabajo pudieron lograr una precisión de 82.9% usando SVM y un unigram model. Sin embargo, en los métodos por machine learning tradicional, las características son definidas y extraídas de forma manual o por métodos de selección de características.

En los últimos años, varios estudios han propuesto el análisis de sentimientos basado en el aprendizaje profundo, que tienen diferentes características y rendimiento. En los modelos de aprendizaje profundo, las variables se aprenden y extraen automáticamente, logrando una mayor precisión y rendimiento.

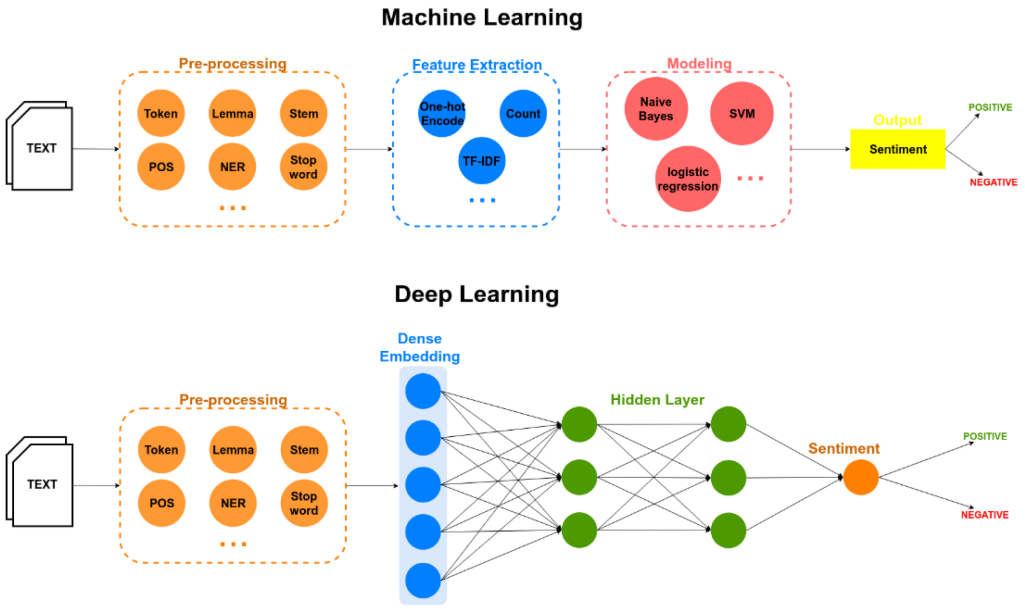

Figura 1. Diferencias entre dos enfoques de clasificación de polaridad de sentimiento, machine learning (arriba) y deep learning (abajo). Part of Speech (POS); Named Entity Recognition (NER); Term Frequency-Inverse Document Frequency (TF-IDF). [53]

Las redes neuronales artificiales y el Deep Learning actualmente brindan las mejores soluciones a muchos problemas en los campos del reconocimiento de imágenes y voz, así como en el procesamiento del lenguaje natural. Deep learning incluye varias redes como CNN (Convolutional Neural Networks), RNN (Recurrent Neural Networks), Recursive Neural Tensor Network (RNTN) , entre otras. Nadia Nedjah, Igor Santos y Luiza de Macedo Mourelle presentaron una investigación [9] de cómo diferentes configuración de hiper-parámetros en redes CNN aumentan el rendimiento y logra mejor desempeño sobre otras redes. Los resultados de la comparación lograron demostrar que una CNN utilizada para el análisis de sentimientos puede superar significativamente a los modelos más utilizados y que también puede ser competitiva y eventualmente superar a los modelos más modernos. Para la experimentación usaron datasets de reseñas de películas MR (con 2 posibles clasificaciones: Positivo y Negativo), Stanford Sentiment Treebank SST-1 que es una extensión de MR (con 5 posibles clasificaciones: muy positivo, positivo, neutro, negativo y muy negativo) y Light Stanford Sentiment Treebank SST-2 que es un derivado de SST-1 que elimina las reseñas neutrales y combina las muy positivas con positivas y lo mismo para muy negativas con negativas. Se logró por medios de varias experimentaciones de hyper-parámetros una precisión de 80.8% con MR, 47.9% con SST-1 y 86.6% con SST-2. En una investigación realizada por Kiran Baktha y B K Tripathy[10] se analizó el rendimiento de varias estructuras de Recurrent Neural Network en

tres datasets: reseñas de productos de Amazon, SST-1 y SST-2. En el trabajo realizado pudieron lograr una precisión usando una variante de RNN, Gated Recurrent Unit, de 83.90% sobre las reseñas de productos de Amazon, 44.61% con SST-1 y 84.40% con SST-2. En el trabajo pudieron concluir que estructuras RNN de tres capas modelo se vuelven demasiado complejas y conducen a una menor precisión. Para grandes conjuntos de datos, más capas podrían mejorar el rendimiento. Un trabajo presentado en la Universidad de Stanford en el 2010 [11] propone un modelo de Recursive Neural Tensor Network que toma como entrada frases de cualquier longitud. El modelo representa una frase a través de vectores de palabras y un árbol de análisis sintáctico y luego calculan vectores para los nodos superiores del árbol utilizando la misma función de composición basada en tensor. El trabajo compara su modelo con varios de composición supervisados, como RNN, las RNN de matriz-vector y redes neuronales que ignoran el orden de las palabras, Naive Bayes (NB), bi-gram NB y SVM. El modelo RNTN obtiene el rendimiento más alto con una precisión del 80,7% al predecir el sentimiento detallado (fine- grained sentiment) y para calificación positiva/negativa de una sola oración lograron una precisión de 85,4%.

2. Conceptos Fundamentales

a. Análisis de Sentimientos

Según Basant Agarwal, Richi Nayak, Namita Mittal y Srikanta Patnaik [16], el análisis de sentimientos es un proceso automatizado para extraer la polaridad del texto con opiniones. Es el campo de estudio que analiza las opiniones, sentimientos, valoraciones, actitudes y emociones de las personas hacia entidades como productos, organizaciones de servicios, individuos, problemas, temas de eventos y sus atributos. También hay muchos nombres y problemas ligeramente diferentes: análisis de sentimientos, minería de opiniones, extracción de opiniones, minería de sentimientos, análisis de subjetividad, análisis de afectos, análisis de emociones, minería de reseñas, etc.

Se han presentado numerosas técnicas hasta la fecha para realizar análisis de sentimientos basados en el aprendizaje supervisado y no supervisado. En el aprendizaje supervisado, las primeras investigaciones se centraron en la aplicación de técnicas de aprendizaje automático supervisado como naïves Bayes, máquinas de vectores de soporte (Support Vector Machines) y algoritmos de aprendizaje de características (feature learning algorithms). Los métodos de aprendizaje no supervisados incluyen el uso de métodos que incluyen el uso de léxicos de sentimientos, análisis gramatical, etc.

b. Niveles de Análisis de Sentimientos

El análisis de sentimientos se realiza en varios niveles de granularidad, como documentos, frases y aspectos. Estos niveles serán discutidos en esta subsección.

i. Nivel Documento

Este nivel determina el sentimiento de un párrafo completo o un documento. El modelo de análisis de sentimiento asume que el documento contiene texto con opiniones sobre una única entidad. Este nivel no admite documentos que comparen múltiples entidades. El problema de determinar si el documento tiene polaridad positiva o negativa se presenta como un problema de clasificación binaria. También se puede manejar como un problema de regresión, por ejemplo, asignando la puntuación de calificación para reseñas de películas en el rango de 1-5 estrellas. Esta tarea también se puede modelar como un problema de clasificación de cinco clases.

ii. Nivel Oración

Este nivel de clasificación de sentimientos tiene como objetivo determinar el sentimiento a partir de una sola oración. La clasificación de subjetividad y la clasificación de polaridad se pueden utilizar para inferir el sentimiento de una oración. La clasificación de subjetividad se enfoca en encontrar si una oración es subjetiva u objetiva. Por otro lado, la clasificación de polaridad determina si una oración subjetiva dada es positiva o negativa. Las técnicas de Deep Learning existentes se centran en predecir la polaridad de una oración como positiva, negativa y neutral. Similar a la suposición a nivel de documento, la clasificación de sentimiento a nivel de oración supone que cada oración contiene un sentimiento sobre una sola entidad.

iii. Análisis de Sentimientos basado en aspectos

En este nivel, se extraen los sentimientos que los usuarios expresan hacia aspectos (características) de las entidades (objetos) como películas y restaurantes. Su objetivo es encontrar los pares de aspecto y polaridad de un texto dado. Este nivel asume que una sola entidad está presente por documento. Aspect-based sentiment analysis se puede dividir en cuatro fases: extracción de términos de aspecto, polaridad de término de aspecto, detección de categoría de aspecto y polaridad de categoría de aspecto. La extracción de términos de aspecto implica identificar los términos de aspecto de un conjunto de oraciones con entidades predefinidas (p.ej., computadoras portátiles) y devolver la lista de distintos términos de aspecto. La segunda subtarea, la polaridad del término de aspecto se centra en determinar la polaridad del término de aspecto detectado en la primera subtarea. La detección de categorías de

aspectos identifica las categorías de aspectos en cada oración en función de un conjunto predefinido de categorías de aspectos (por ejemplo, general, precio). La cuarta subtarea polaridad de la categoría de aspecto se centra en determinar la polaridad de cada categoría de aspecto a partir de un conjunto dado de oraciones.

El análisis de sentimiento basado en aspectos dirigidos es una extensión del análisis de sentimiento basado en aspectos. El análisis de sentimiento basado en aspectos asume la ocurrencia de una sola entidad por documento, mientras que el análisis de sentimiento basado en aspectos dirigidos asume un solo sentimiento hacia cada aspecto de una o más entidades. El análisis de sentimientos basado en aspectos dirigidos extrae las entidades específicas, los diferentes aspectos y sus sentimientos correspondientes. Por ejemplo, "El ambiente es bueno en Viceroy pero el servicio es malo, por el contrario, el personal de Novotel es muy rápido y la comida es rica como siempre". Este ejemplo habla de aspectos de dos hoteles diferentes. El análisis de sentimientos basado en aspectos dirigidos reconoce "Viceroy" y "Novotel" como dos entidades específicas y genera el resultado como {Viceroy, ambiente, positivo}, {Viceroy, servicio, positivo}, {Novotel, servicio, positivo,} {Novotel, comida, positivo}

c. Procesamiento del lenguaje natural (NLP)

El procesamiento del lenguaje natural (NLP) se refiere a las teorías y técnicas que abordan el problema de la comunicación del lenguaje natural con las computadoras. Otra definición más completa de Ela Kumar [17] explica que el procesamiento del lenguaje natural es un campo de la informática y la lingüística que se ocupa de las interacciones entre las computadoras y el lenguaje humano (natural). Kumar explica que los sistemas de generación de lenguaje natural convierten la información de las bases de datos informáticas en un lenguaje humano legible. Los sistemas de comprensión del lenguaje natural convierten muestras de lenguaje humano en representaciones más formales, como árboles de análisis sintáctico, que son más fáciles de manipular para los programas informáticos.

Cuando se habla de niveles de análisis lingüístico se refiere al hecho de que existen múltiples tipos de procesamiento del lenguaje que actúan cuando los seres humanos producen o comprenden el lenguaje. Se cree que los seres humanos normalmente utilizan todos estos niveles, ya que cada nivel transmite diferentes tipos de significado. Pero varios sistemas de NLP utilizan diferentes niveles, o combinaciones de niveles de análisis lingüístico, y esto se ve en las diferentes aplicaciones de PNL. Esto también genera mucha confusión por parte de los no especialistas en cuanto a qué es realmente PNL, porque se puede decir que un sistema que utiliza cualquier subconjunto de estos niveles de análisis es un sistema basado

en PNL. La diferencia entre ellos, por lo tanto, puede ser en realidad si el sistema usa PNL "débil" o PNL "fuerte".

d. Deep Learning

John D. Kelleher [18] define Deep Learning como el subcampo de la inteligencia artificial que se centra en la creación de grandes modelos de redes neuronales que son capaces de tomar decisiones precisas basadas en datos.

Deep learning descubre una estructura intrincada en grandes conjuntos de datos mediante el uso del algoritmo de retropropagación para indicar cómo una máquina debe cambiar sus parámetros internos que se utilizan para calcular la representación en cada capa a partir de la representación en la capa anterior.

e. Redes Neuronales

Las redes neuronales artificiales representan una clase de modelos de aprendizaje automático (Machine Learning), inspirados en estudios sobre los sistemas nerviosos centrales de los mamíferos. Cada red está formada por varias neuronas interconectadas, organizadas en capas, que intercambian mensajes cuando ocurren ciertas condiciones.

Los estudios iniciales se iniciaron a finales de la década de 1950 con la introducción del perceptrón [19], una red de dos capas utilizada para operaciones simples, y que se expandió aún más a finales de 1960 con la introducción del algoritmo de retropropagación, utilizado para el entrenamiento eficiente de redes multicapa [20].

i. RNN

La red neuronal recurrente es una clase de redes neuronales que explotan la naturaleza secuencial de su entrada. Dichas entradas pueden ser texto, voz, series de tiempo y cualquier otra cosa en la que la aparición de un elemento en la secuencia dependa de los elementos que aparecieron antes.

Un RNN se puede pensar en un gráfico de celdas RNN, donde cada celda realiza la misma operación en todos los elementos de la secuencia.

Los RNN procesan una secuencia de entradas de a un elemento a la vez, manteniendo en sus unidades ocultas un "vector de estado" que contiene implícitamente información histórica de todos los elementos pasados de la secuencia. Cuando consideramos las salidas de las unidades ocultas en diferentes pasos de tiempo como si fueran las salidas de diferentes neuronas en una red profunda de múltiples capas (Fig.2,

derecha), queda claro cómo se puede aplicar la retropropagación para entrenar RNN. En la figura 2, en el tiempo ***t***, la celda tiene una entrada Xt y una salida Ot. Parte de la salida (el estado oculto St) se retroalimenta a la celda para su uso en un paso de tiempo t + 1 posterior. Los parámetros están contenidos en su matriz de peso, los parámetros de la RNN están definidos por tres matrices de peso U, V y W, correspondientes a la entrada, salida y estado oculto respectivamente.

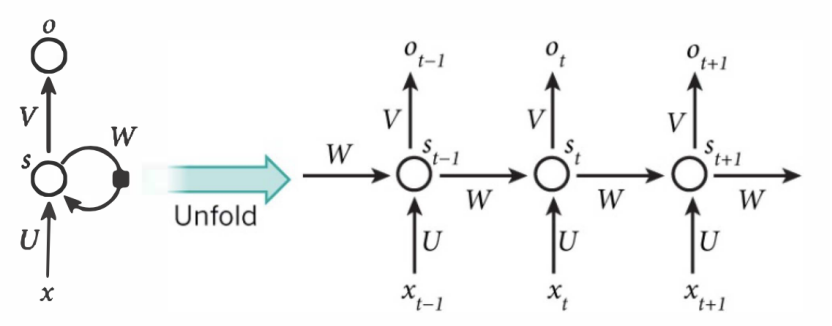

Figura 2. Salidas de diferentes neuronas en una red profunda de múltiples capas RNN [21]

1. LSTM

Las redes de memoria a largo y corto plazo o long short-term memory (LSTM) utilizan unidades ocultas especiales, cuyo comportamiento natural es recordar entradas durante mucho tiempo. Una unidad especial llamada celda de memoria actúa como un acumulador o una neurona con fugas cerrada: tiene una conexión consigo misma en el siguiente paso de tiempo que tiene un peso de uno, por lo que copia su propio estado de valor real y acumula la señal externa, pero esta auto-conexión es multiplicada por otra unidad que aprende a decidir cuándo borrar el contenido de la memoria.

Las redes LSTM han demostrado ser más efectivas que las RNN convencionales, especialmente cuando tienen varias capas para cada paso de tiempo. Las redes LSTM o formas relacionadas de unidades cerradas también se utilizan actualmente para las redes de codificadores y decodificadores que funcionan bien en el área de traducción automática (machine translation).

2. RNTN

Las redes de tensores neuronales recursivas (RNTN) son redes neuronales útiles para el procesamiento del lenguaje natural. Tienen una estructura de árbol con una red neuronal en cada nodo. Las redes de tensores neuronales recursivas se pueden utilizar para la segmentación de límites, a fin de determinar qué grupos de palabras son positivos y cuáles son negativos. Lo mismo se aplica a las oraciones en su conjunto. Los vectores de palabras (en inglés, word vectors) se utilizan como

características (features) y sirven como base para la clasificación secuencial. Luego se agrupan en sub-frases, y las sub-frases se combinan en una oración que se puede clasificar por sentimiento y otras métricas. Figura 3 muestra una sola capa tensorial

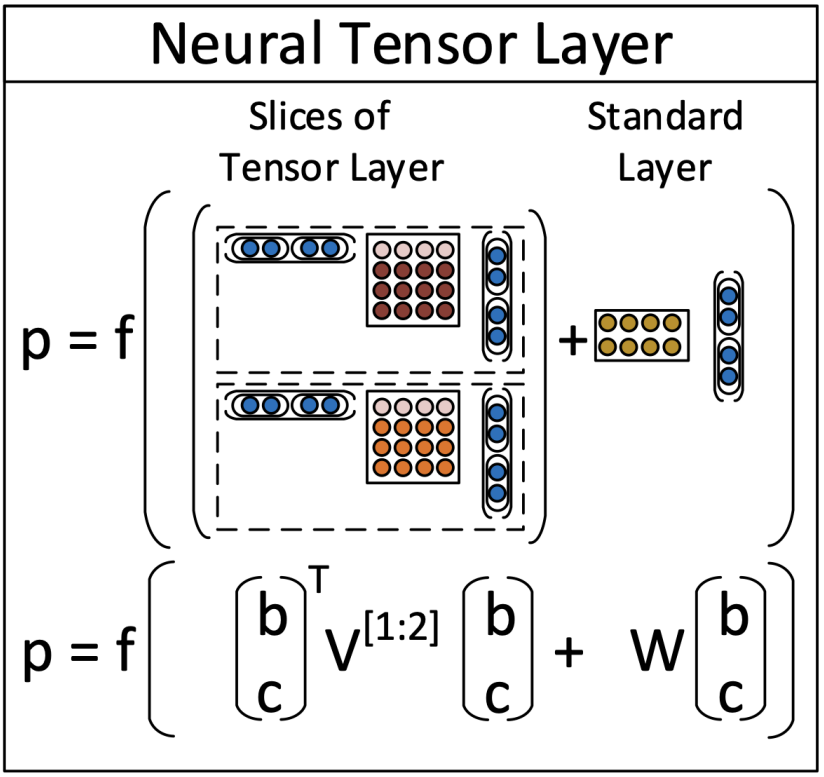

Figura 3. Una sola capa de la red de tensor neuronal recursivo. Cada cuadro punteado representa uno de varios sectores. Fuente de imagen [22]

ii. CNN

Las redes neuronales convolucionales son redes neuronales que se utilizan principalmente para clasificar imágenes (es decir, nombrar lo que ven), agrupar imágenes por similitud (búsqueda de fotos) y realizar el reconocimiento de objetos dentro de las escenas. Sin embargo, las CNN no se limitan al reconocimiento de imágenes. Se han aplicado directamente a la analítica de texto. Y se aplican al sonido cuando se representa visualmente como un espectrograma, y gráfica datos con redes convolucionales de gráficos.

Las redes neuronales convolucionales ingieren y procesan imágenes como tensores, y los tensores son matrices de números con dimensiones adicionales. Las imágenes son solo algunos puntos en el espacio, al igual que las palabras vectores (word vectors). Al representar cada palabra con un vector de números

de una longitud específica y apilar un montón de palabras una encima de la otra, obtenemos una "imagen". Los filtros de visión por computadora generalmente tienen el mismo ancho y alto y se deslizan sobre partes locales de una imagen. En NLP, normalmente se usan filtros que se deslizan sobre incrustaciones de palabras (word embeddings): filas de matriz. Por lo tanto, los filtros suelen tener el mismo ancho que la longitud de las incrustaciones de palabras. La altura varía, pero generalmente es de 1 a 5, lo que corresponde a diferentes n-grams. Los N-grams son solo un montón de palabras posteriores. Al analizar secuencias, podemos comprender mejor el significado de una oración. Por ejemplo, la palabra "me gusta" por sí sola tiene un significado opuesto en comparación con el bi-gram "no me gusta"; este último nos da una mejor comprensión del significado real. En cierto modo, al analizar n-grams, capturamos las relaciones espaciales en los textos, lo que facilita que el modelo comprenda el sentimiento. La visualización a continuación resume los conceptos que acabamos de cubrir.

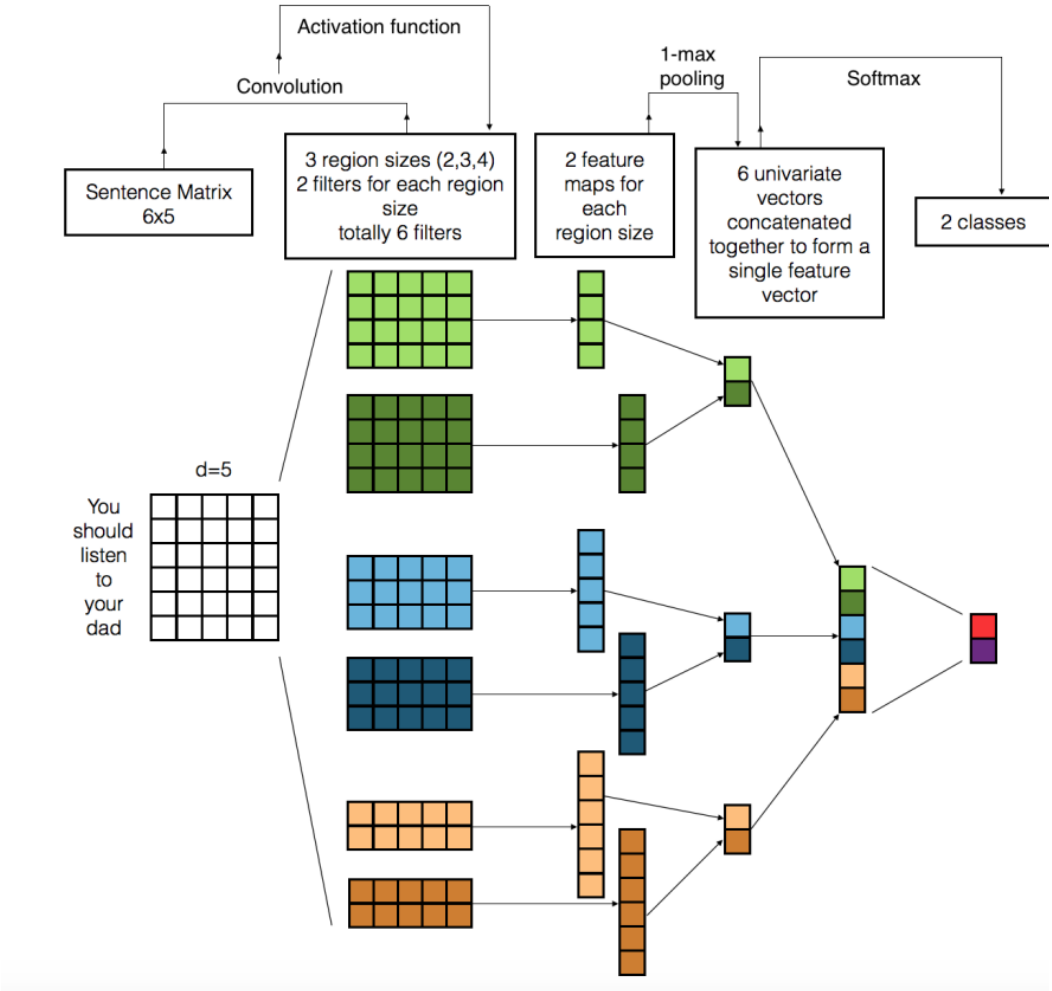

Figura 4. CNNs aplicado a NLP [23]

f. Librerías exclusivas para el análisis de sentimientos

i. VADER

VADER [36] (Valence Aware Dictionary and sEntiment Reasoner) es una herramienta de análisis de sentimientos basada en reglas y léxico que está específicamente en sintonía con los sentimientos expresados en las redes sociales.

VADER es sensible tanto a la polaridad (positiva / negativa) como a la intensidad de la emoción. Está disponible en el paquete NLTK y se puede aplicar directamente a datos de texto sin etiquetar.

El análisis sentimental de VADER se basa en un diccionario que asigna características léxicas a intensidades emocionales conocidas como puntuaciones de sentimiento. La puntuación de sentimiento de un texto se puede obtener sumando la intensidad de cada palabra en el texto.

Por ejemplo, palabras como "amor", "disfrutar", "feliz", "me gusta", todas transmiten un sentimiento positivo. Además, VADER es lo suficientemente inteligente como para comprender el contexto básico de estas palabras, como "no amó" como una declaración negativa. También comprende el énfasis de las mayúsculas y la puntuación, como "DISFRUTAR".

La puntuación compuesta se calcula sumando las puntuaciones de valencia de cada palabra en el léxico, se ajusta de acuerdo con las reglas y luego se normaliza para estar entre -1 (más extremo negativo) y +1 (más extremo positivo). Esta es la métrica más útil si desea una única medida unidimensional de sentimiento para una oración determinada. Llamarlo una 'puntuación compuesta ponderada normalizada' es exacto.

ii. CoreNLP

Stanford CoreNLP [29] proporciona un conjunto de herramientas de análisis de lenguaje natural escritas en Java. Puede tomar la entrada de texto en lenguaje humano sin procesar y dar las formas básicas de las palabras, sus partes del discurso, ya sean nombres de empresas, personas, etc., normalizar e interpretar fechas, horas y cantidades numéricas, marcar la estructura de las oraciones en términos de frases o dependencias de palabras, e indique qué frases nominales se refieren a las mismas entidades. Originalmente se desarrolló para inglés, pero ahora también ofrece distintos niveles de compatibilidad con árabe, chino, francés, alemán y español. Stanford CoreNLP es un framework integrado, lo que hace que sea muy fácil aplicar un montón de herramientas de análisis de lenguaje a un fragmento de texto. A partir de texto sin formato, se puede ejecutar todas las herramientas con solo dos líneas de código. Sus análisis proporcionan los bloques de construcción fundamentales para aplicaciones de comprensión de texto de nivel superior y de dominio específico. Stanford CoreNLP es un conjunto de herramientas de procesamiento de lenguaje natural estables y bien probadas, ampliamente utilizadas por varios grupos en la academia, la industria y el gobierno. Las herramientas utilizan de forma diversa componentes de aprendizaje automático probabilístico y de aprendizaje profundo basados en reglas.

iii. Flair

Flair [37] es una poderosa biblioteca de NLP. Flair permite aplicar modelos de procesamiento de lenguaje natural (NLP) de última generación al texto, como reconocimiento de entidad con nombre (NER),

part-of-speech tagging (PoS), soporte especial para datos biomédicos, detección desambiguación y clasificación, con soporte para un número cada vez mayor de idiomas.

La idea central del framework es presentar una interfaz simple y unificada para tipos conceptualmente muy diferentes de incrustaciones de palabras y documentos (word and document embeddings). Esto oculta efectivamente toda la complejidad de la ingeniería específica de la incrustación y permite a los investigadores mezclar y combinar varias incrustaciones con poco esfuerzo. El marco también implementa rutinas de selección de hiper-parámetros y entrenamiento de modelos estándar, así como un módulo de búsqueda de datos que puede descargar conjuntos de datos de PNL disponibles públicamente y convertirlos en estructuras de datos para la configuración rápida de experimentos. Además, FLAIR también cuenta con una lista creciente de modelos previamente entrenados que permiten a los usuarios aplicar modelos ya entrenados a su texto. Flair una interfaz simple y unificada para todas las incrustaciones de palabras, así como para combinaciones arbitrarias de incrustaciones. Esta interfaz permite a los investigadores construir una arquitectura de modelo único que luego puede hacer uso de cualquier tipo de incrustación de palabras sin esfuerzo de ingeniería adicional.

g. Librerías de pre-procesamiento de texto

i. Natural Language Toolkit (NLTK)

NLTK es una plataforma open-source para crear programas Python que funcionan con datos de lenguaje humano. Proporciona interfaces fáciles de usar para más de 50 corpus y recursos léxicos como WordNet, junto con un conjunto de bibliotecas de procesamiento de texto para clasificación, tokenización, derivación, etiquetado, análisis y razonamiento semántico, envoltorios para bibliotecas de PNL de nivel industrial, y un foro de discusión activo. [24]

ii. TextBlob

TextBlob [28] es una librería de Python para el procesamiento del lenguaje natural (NLP). TextBlob usa activamente Natural Language ToolKit (NLTK) para realizar sus tareas. Como se mencionó, NLTK es una librería que brinda un fácil acceso a una gran cantidad de recursos léxicos y permite a los usuarios trabajar con categorización, clasificación, etc. TextBlob es una librería simple que admite análisis y operaciones complejas sobre datos textuales.

Para los enfoques basados en el léxico, un sentimiento se define por su orientación semántica y la intensidad de cada palabra en la oración. Esto requiere un diccionario predefinido que clasifique palabras

negativas y positivas. Generalmente, un mensaje de texto estará representado por un bag of words. Después de asignar puntajes individuales a todas las palabras, el sentimiento final se calcula mediante alguna operación de agrupación, como tomar un promedio de todos los sentimientos.

iii. Gensim

Gensim es una biblioteca de código abierto para el modelado no supervisado y NLP, que utiliza aprendizaje automático. Se considera una herramienta robusta de modelado de espacios vectoriales implementada en Python. A diferencia de NLTK, Gensim es la mejor forma de procesar conjuntos de datos masivos. La biblioteca de Gensim se construyó principalmente para la estimación de similitud de documentos. Admite tres tareas principales de NLP: recuperar documentos semánticamente similares, semántica estadística escalable y analizar documentos de texto para la estructura semántica. Gensim incluye implementaciones de muchos algoritmos como fastText, word2vec y doc2vec que se utilizan mucho en el campo del análisis de sentimiento. Su implementación altamente optimizada y nativa de los modelos de aprendizaje automático de word2vec de Google lo convierte en un fuerte competidor para su inclusión en un proyecto de análisis de sentimientos, ya sea como un marco central o como un recurso de biblioteca.

Análisis/Comparación

1. Librerias NLP

En esta sección haremos una introducción y experimentación con seleccionadas herramientas, técnicas y algoritmos con el fin de decidir cuál será el más adecuado para el prototipo objetivo. Se hará una descripción de cada técnica y una demostración de su funcionamiento.

a. Natural Language Toolkit (NLTK)

Tokenización

La tokenización es el primer paso en el análisis de texto. El proceso de dividir un párrafo de texto en fragmentos más pequeños, como palabras u oraciones, se llama Tokenización. Token es una entidad única que construye bloques para una oración o un párrafo. El tokenizador de oraciones divide el párrafo de texto en oraciones como se puede observar en el siguiente ejemplo:

```
import nltk
from nltk.tokenize import sent_tokenize
```

```
text = "Hola me llamo Julia. Estudio ingenieria en informatica en la
Universidad de Belgrano. Tengo 24 años"
print(sent_tokenize(text))
```

Esto devuelve el siguiente array donde cada elemento corresponde a cada oración del texto ingresado

```
['Hola me llamo Julia.', 'Estudio ingenieria en informatica en la
Universidad de Belgrano.', 'Tengo 24 años']
```

El tokenizador de palabras divide el texto en palabras en lugar de oraciones. Devolviendo el siguiente resultado

```
['Hola', 'me', 'llamo', 'Julia', '.', 'Estudio', 'ingenieria', 'en',
'informatica', 'en', 'la', 'Universidad', 'de', 'Belgrano', '.',
'Tengo', '24', 'años']
```

Stop Words

Un texto puede contener palabras como "soy", "quién", "dónde". Estas palabras se las llaman stop words o palabras vacías y podemos eliminarlas. No existe una lista universal de stop words en nlp, sin embargo, la biblioteca nltk proporciona una lista de stop words

```
sample = "Stopwords code which contain a sample sentence, showing off
the stop words filtration."
stop_words_array = set(stopwords.words('english'))
words = word_tokenize(sample)
filtered_sentence =  []
for w in words:
    if w not in stop_words_array:
        filtered_sentence.append(w)
print(words)
print('After removing stopwords')
print(filtered_sentence)
```

El ejemplo anterior devuelve lo siguiente

```
['Stopwords', 'code', 'which', 'contain', 'a', 'sample', 'sentence', ',', 'showing', 'off', 'the', 'stop', 'words',
'filtration', '.']
After removing stopwords
['Stopwords', 'code', 'contain', 'sample', 'sentence', ',', 'showing', 'stop', 'words', 'filtration', '.']
```

Movie Reviews

Para esta técnica, utilizaremos el corpus de movie_reviews de NLTK como nuestros datos de entrenamiento etiquetados. El corpus de películas contiene 2.000 reseñas de películas con clasificación de polaridad de sentimiento. Está compilado por Pang, Lee.

Ahora importamos un conjunto de datos de reseñas de películas que ofrece nltk.corpus e intentaremos limpiar esos datos

```
from nltk.corpus import stopwords
from nltk.corpus import movie_reviews
import string

useless_words = stopwords.words('english') + list(string.punctuation)
filtered_words = [word for word in movie_reviews.words() if not word
in useless_words]
print(len(filtered_words)/1e6)
```

Además de agregar la lista de stopwords, sumamos la lista de puntuación para conformar useless_words o palabras innecesarias. Así filtramos las palabras innecesarias de las reseñas de películas y obtenemos **710579** palabras. Podemos listar las las palabras más comunes usando:

```
from collections import Counter
word_counter = Counter(filtered_words)
most_commond_words = word_counter.most_common()[:10]
print(most_commond_words)
```

Así obtenemos el siguiente listado de las 10 palabras más comunes y su contador

```
[('film', 9517), ('one', 5852), ('movie', 5771), ('like', 3690),
('even', 2565), ('good', 2411), ('time', 2411), ('story', 2169),
('would', 2109), ('much', 2049)]
```

Análisis de sentimientos sobre Movie Corpus

La clasificación mediante el aprendizaje automático (machine learning) es una de las técnicas que se utiliza para los modelos de opinión. Construiremos un clasificador de sentimientos usando el corpus de reseñas de películas. La clasificación es una técnica que requiere etiquetas o labels de datos. Aquí es donde aprovecharemos de bag of words y las críticas positivas y negativas seleccionadas que

descargamos. Implementaremos la función de bag of words para crear una etiqueta positiva o negativa para cada bag of words de reseña.

```
from nltk.corpus import movie_reviews
def build_bag_of_words_features(words):
    return {
        word:1 for word in words if not word in useless_words}
positive_reviews = movie_reviews.fileids('pos')
negative_reviews = movie_reviews.fileids('neg')
negative_features                              =                              [
(build_bag_of_words_features(movie_reviews.words(fileids    =    [f])),
'neg')
                  for f in negative_reviews]
positive_features                              =                              [
(build_bag_of_words_features(movie_reviews.words(fileids    =    [f])),
'pos')
                  for f in positive_reviews]
print(len(negative_features))
print(len(positive_features))
```

Así se obtienen 1000 features o características de cada reseña positiva y 1000 features de cada reseña negativa.

Usaremos Naive Bayes Classifier para esta tarea de clasificación. Naive Bayes Classifier es un clasificador muy simple con un enfoque probabilístico de clasificación. Lo que esto significa es que las relaciones entre las features de entrada y las etiquetas de clase se expresan como probabilidades. Entonces, dadas las features o características de entrada, por ejemplo, se estima la probabilidad para cada clase. La clase con mayor probabilidad determina la etiqueta de la muestra.

Uno de los clasificadores de aprendizaje automático supervisados más simples es el Clasificador Naive Bayes, entrenaremos con el 80% de los datos qué palabras se asocian generalmente con reseñas positivas o negativas.

Se tiene 1,000 registros tanto de features positivas como negativas. Se puede utilizar el 80% de los datos para la clasificación en Naive Bayes. Cuando proporcionamos las primeras 800 filas de cada feature, es el 80%. Así que almacenaremos ese número, 800, en una variable llamada split.

```
from nltk.classify import NaiveBayesClassifier

split = 800
```

```
sentiment_classifier                                                    =
NaiveBayesClassifier.train(positive_features[:split]                    +
negative_features[:split] )
```

Se clasifican las primeras 800 features positivas y las primeras 800 features negativas. Hay que tener en cuenta que tenían etiquetas pos y neg.

```
print( nltk.classify.util.accuracy(sentiment_classifier,
positive_features[:split] + negative_features[:split] ))
print( nltk.classify.util.accuracy(sentiment_classifier,
positive_features[split:] + negative_features[split:] ))
```

Se puede ver que tiene aproximadamente un 98% de precisión, por lo que es bastante bueno. Y para el otro 20% ronda el 70%. La precisión estimada para un humano es de aproximadamente el 80%. Entonces, una precisión de alrededor del 70% es una precisión bastante buena para un modelo tan simple.

El listado del vocabulario era extenso y el Clasificador de sentimientos usó todas las palabras, pero ¿cuál de esas palabras nos dio esta precisión tan alta? El Clasificador de sentimientos que construimos tiene una función para mostrar las características más informativas.

```
print(sentiment_classifier.show_most_informative_features())
Most Informative Features
        outstanding = 1              pos : neg    =     13.9 : 1.0
        insulting = 1                neg : pos    =     13.7 : 1.0
        vulnerable = 1               pos : neg    =     13.0 : 1.0
        ludicrous = 1                neg : pos    =     12.6 : 1.0
        uninvolving = 1              neg : pos    =     12.3 : 1.0
        astounding = 1               pos : neg    =     11.7 : 1.0
        avoids = 1                   pos : neg    =     11.7 : 1.0
        fascination = 1              pos : neg    =     11.0 : 1.0
        affecting = 1                pos : neg    =     10.3 : 1.0
        animators = 1                pos : neg    =     10.3 : 1.0
```

b. Gensim

Word2Vec

Word2Vec [25] es un modelo no supervisado basado en word embedding. Su objetivo es detectar el significado y las relaciones semánticas entre palabras explotando la co-ocurrencia de palabras en

documentos pertenecientes a un corpus dado. El objetivo de Word2Vec es capturar el contexto de las palabras, utilizando enfoques de aprendizaje automático como Recurrent o Deep Neural Networks.
Involucra dos algoritmos de aprendizaje diferentes: (i) el algoritmo Continuous Bag-of-Words (CBOW), cuyo objetivo es predecir una palabra dada las palabras que las rodean, y (ii) el algoritmo Continuous Skip-Gram (Skip-gram): que predice un conjunto de palabras cuando se conoce una sola palabra. Según Mikolov [25], Skip-gram funciona bien con una pequeña cantidad de datos de entrenamiento, representando con precisión incluso palabras o frases raras, mientras que CBOW es mucho más rápido de entrenar (comparando con Skip-gram) y un poco más preciso para palabras frecuentes. Word2Vec opera en un corpus de oraciones, primero construye un vocabulario basado en las palabras que aparecen en el corpus más veces que un umbral definido por el usuario, y luego aplica el CBOW o el algoritmo Skip-gram en los documentos de entrada para aprender las representaciones vectoriales de las palabras en un espacio D-dimensional.

Doc2Vec

Doc2Vec es una extensión de Word2Vec que se aplica a un documento como un todo en lugar de palabras individuales. Este modelo fue desarrollado por Le y Mikolov [26], y tiene como objetivo crear una representación numérica de un documento en lugar de una palabra. Doc2Vec opera con la lógica de que el significado de una palabra también depende del documento en el que aparece. Los vectores generados por Doc2Vec se pueden utilizar para encontrar similitudes entre documentos.
El modelo PV-DM (Modelo Distributed Memory Model of Paragraph Vectors) en Doc2Vec mapea cada oración en un *vector de* independiente en la matriz D. Al mismo tiempo, cada palabra también se asigna en un vector separado. Al mismo tiempo, cada palabra también se mapea en un vector independiente - *vector palabra* como una matriz W de la columna. El vector de oración y el vector de palabra de un extremo a otro sirven para predecir la probabilidad de la siguiente palabra en el texto como se muestra en la Figura 5.

Word2Vec con Gensim

Se utiliza Word2Vec para el análisis de sentimientos al intentar clasificar el corpus de reseñas de películas de Cornell IMDB [27]. Usamos gensim, ya que proporciona una implementación de Word2Vec (y Doc2Vec) legible y sencilla de usar. También se utiliza numpy para la manipulación general de matrices y sklearn para el clasificador de regresión logística

Importamos los módulos necesarios:

```
# gensim modules
from gensim import utils
from gensim.models.doc2vec import LabeledSentence
from gensim.models import Doc2Vec

# numpy
import numpy

# classifier
from sklearn.linear_model import LogisticRegression

# random
import random
```

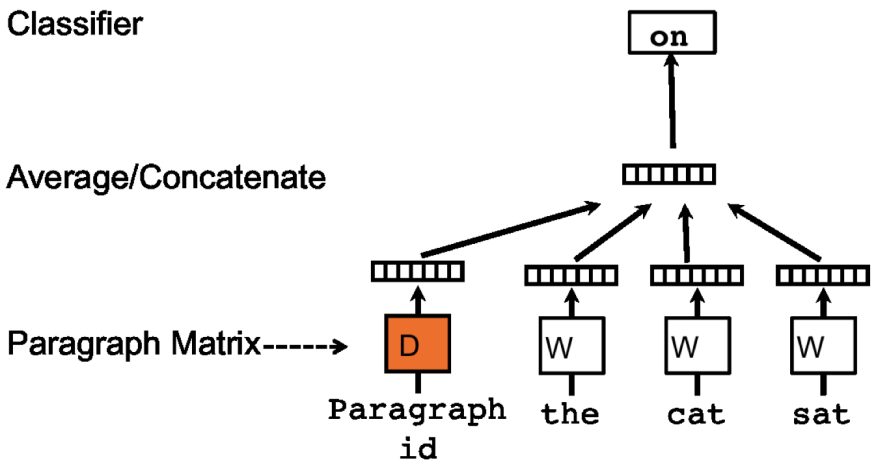

Figura 5.Modelo de cálculo de vectores de oraciones. [26]

Como se hizo anteriormente con NLTK, se ”limpiará” las reseñas convirtiéndo todo a minúsculas y eliminando la puntuación. Como resultado tenemos estos cinco documentos:

- test-neg.txt: 12500 reseñas negativas de películas a partir de los datos de prueba
- test-pos.txt: 12500 reseñas positivas de películas a partir de los datos de la prueba
- train-neg.txt: 12500 reseñas negativas de películas a partir de los datos de entrenamiento
- train-pos.txt: 12500 reseñas positivas de películas a partir de los datos de entrenamiento
- train-unsup.txt: 50000 reseñas de películas sin etiquetar

Cada documento debe estar en una línea, separado por nuevas líneas. Esto es extremadamente importante, porque nuestro analizador depende de esto para identificar oraciones.

Entrada de datos a Doc2Vec

Doc2Vec (la parte de gensim que implementa el algoritmo Doc2Vec) hace un gran trabajo en word embedding, pero no tiene facilidad para la lectura de archivos. Para eso se crea una clase que produce LabelSentence, una clase de gensim.models.doc2vec que representa una sola oración. ¿Por qué la palabra "labeled"? Con eso se diferencia Doc2Vec de Word2Vec. Word2Vec simplemente convierte una palabra en un vector. Doc2Vec no solo hace eso, sino que también agrega todas las palabras de una oración en un vector. Para hacer eso, simplemente trata la etiqueta, “label”, de una oración como una palabra especial. Por lo tanto, esa palabra especial es una etiqueta para una oración.
Así que tenemos que formatear las oraciones en

```
[['word1', 'word2', 'word3', 'lastword'], ['label1']]
```

LabeledSentence es simplemente una forma más ordenada de hacer eso. Contiene una lista de palabras y una etiqueta para la oración. Sin embargo, necesitamos una forma de convertir nuestro nuevo corpus separado por líneas en una colección de LabeledSentences. El constructor predeterminado para la clase La LabeledLineSentence predeterminada en Doc2Vec puede hacer eso para un solo archivo de texto, pero no puede hacerlo para varios archivos.

Así que utilizamos la siguiente clase de LabeledLineSentence. El constructor toma un diccionario que define los archivos a leer y la etiqueta prefija las oraciones de ese documento. Luego, Doc2Vec puede leer la colección directamente a través del iterador, o podemos acceder a la matriz directamente. También necesitamos una función que devuelva una versión permutada de la matriz de frases etiquetadas.

```
class LabeledLineSentence(object):
    def __init__(self, sources):
        self.sources = sources

        flipped = {}

        for key, value in sources.items():
            if value not in flipped:
                flipped[value] = [key]
            else:
                raise Exception('Non-unique prefix encountered')

    def __iter__(self):
        for source, prefix in self.sources.items():
            with utils.smart_open(source) as fin:
                for item_no, line in enumerate(fin):
                                                                yield
LabeledSentence(utils.to_unicode(line).split(),  [prefix  +  '_%s'  %
item_no])

    def to_array(self):
        self.sentences = []
        for source, prefix in self.sources.items():
            with utils.smart_open(source) as fin:
                for item_no, line in enumerate(fin):

self.sentences.append(LabeledSentence(utils.to_unicode(line).split(),
[prefix + '_%s' % item_no]))
        return self.sentences

    def sentences_perm(self):
```

```
        shuffled = list(self.sentences)
        random.shuffle(shuffled)
        return shuffled
```

```
sources = {'test-neg.txt':'TEST_NEG', 'test-pos.txt':'TEST_POS',
'train-neg.txt':'TRAIN_NEG', 'train-pos.txt':'TRAIN_POS',
'train-unsup.txt':'TRAIN_UNS'}

sentences = LabeledLineSentence(sources)
```

Construyendo la tabla de vocabulario

Doc2Vec requiere que construyamos una tabla de vocabulario (simplemente digiriendo todas las palabras y filtrando las palabras únicas, y haciendo algunos recuentos básicos de ellas). Entonces lo usamos de entrada con la matriz de oraciones. **model.build_vocab** toma una matriz de LabeledLineSentence, de ahí la función **to_array** en la clase LabeledLineSentences.

```
model = Doc2Vec(min_count=1, window=10, size=100, sample=1e-4,
negative=5, workers=7)

all_data = sentences.to_array();
model.build_vocab(all_data)
```

min_count: ignora todas las palabras con una frecuencia total inferior a esta. Lo dejamos en 1, ya que las etiquetas de las oraciones solo aparecen una vez. Si se establece en un valor superior a 1, se perderán las oraciones.

window: la distancia máxima entre la palabra actual y la predicha dentro de una oración. Word2Vec utiliza un modelo de skip-gram, y este es simplemente el tamaño de ventana del modelo de skip-gram..

size: dimensionalidad de los vectores de características en la salida.

sample: umbral para configurar qué palabras de mayor frecuencia se reducen aleatoriamente

workers: cantidad de subprocesos de trabajo para entrenar el modelo.

Entrenando Doc2Vec

Ahora entrenamos el modelo. El modelo está mejor entrenado si en cada **epoch** de entrenamiento, la secuencia de oraciones de entrada al modelo es aleatoria. Esto es importante: perderse estos pasos genera malos resultados. Esta es la razón por la que se utiliza el método **sentences_perm** de la clase LabeledLineSentences.

```
for epoch in range(10):
                                    model.train(sentences.sentences_perm(),
total_examples=len(all_data), epochs=epoch)
```

Clasificación de sentimientos

Ahora usemos estos vectores para entrenar un clasificador. Primero, debemos extraer los vectores de entrenamiento. Tenemos 25000 reseñas de entrenamiento con igual número de positivas y negativas (12500 positivas, 12500 negativas).

Por lo tanto, creamos una matriz numpy. Dado que el clasificador que usamos solo toma matrices numpy. Hay dos matrices paralelas, una que contiene los vectores (**train_arrays**) y la otra que contiene las etiquetas (**train_labels**).

Simplemente colocamos los positivos en la primera mitad de la matriz y los negativos en la segunda mitad.

```
train_arrays = numpy.zeros((25000, 100))
train_labels = numpy.zeros(25000)

for i in range(12500):
   prefix_train_pos = 'TRAIN_POS_' + str(i)
   prefix_train_neg = 'TRAIN_NEG_' + str(i)
   train_arrays[i] = model[prefix_train_pos]
   train_arrays[12500 + i] = model[prefix_train_neg]
   train_labels[i] = 1
   train_labels[12500 + i] = 0
```

La matriz de entrenamiento se ve así: filas y filas de vectores que representan cada oración.

```
print(train_arrays)
```

```
[[ 1.30512428  1.68576717  1.5165925  ...  0.29696208  2.78531575
  -0.20492822]
 [-0.11646625 -0.75204927  3.33039618 ... -1.0949223  -0.78319466
  -0.01701309]
 [ 0.6879009   0.80603647  3.42374945 ... -1.39232695 -0.60566896
  -3.47454762]
 ...
 [-1.7694217   2.83518744 -3.75648928 ... -0.73819071 -0.66753089
  -3.66449165]
 [-0.07310802  0.78843659 -3.29880404 ... -1.09200048 -2.49450827
  -0.4556644 ]
 [-0.85866153  0.36288846 -1.10579991 ...  2.43731594  2.01431465
  -1.47985995]]
```

Se hace lo mismo con los datos de prueba: datos que enviaremos al clasificador después de haberlo entrenado con los datos de entrenamiento. Esto nos permite evaluar nuestros resultados. El proceso es prácticamente el mismo que extraer los resultados de los datos de entrenamiento.

Clasificación

Ahora se entrena un clasificador de regresión logística usando los datos de entrenamiento.

```
classifier = LogisticRegression()
classifier.fit(train_arrays, train_labels)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Y se descubre que alcanza una precisión cercana al 87% para el análisis de sentimientos. Esto es bastante eficiente, dado que solo estamos usando una SVM lineal y una red neuronal muy poco profunda.

```
classifier.score(test_arrays, test_labels)
```

```
0.86968000000000001
```

c. TextBlob

TextBlob devuelve la polaridad y subjetividad de una oración. La polaridad se encuentra entre [-1,1], -1 define un sentimiento negativo y 1 define un sentimiento positivo. Las palabras de negación invierten la polaridad. TextBlob tiene etiquetas semánticas que ayudan con el análisis detallado. Por ejemplo: emoticones, signos de exclamación, emojis, etc. La subjetividad se encuentra entre [0,1]. La subjetividad cuantifica la cantidad de opiniones personales e información fáctica contenida en el texto. La subjetividad más alta significa que el texto contiene opiniones personales en lugar de información fáctica. TextBlob tiene un parámetro más: la intensidad. TextBlob calcula la subjetividad observando la "intensidad". La intensidad determina si una palabra modifica la siguiente. Para el inglés, los adverbios se utilizan como modificadores ("muy bueno").

Por ejemplo: Calculamos polaridad y subjetividad para “I do not like this movie, it is too awful”. Para este ejemplo en particular, polaridad es -1 y subjetividad es 1, lo cual es aceptable.
Sin embargo, para la oración "This is a well made movie but I would prefer another one". Devuelve 0.0 tanto para subjetividad como para polaridad, que no es la mejor respuesta que esperaríamos.
Se espera que si la biblioteca devuelve exactamente 0.0, ya sea si la oración no contiene ninguna palabra que tenga una polaridad en el conjunto de entrenamiento NLTK o porque TextBlob usa una puntuación de sentimiento promedio ponderada sobre todas las palabras en cada muestra. Esto difumina fácilmente el efecto de las oraciones con polaridades muy variables entre las palabras como en nuestro caso: "well made" y "but".

2. Librerías exclusivas para el análisis de sentimientos

a. CoreNLP

Aunque CoreNLP fue desarrollado originalmente para Java, el grupo de NLP de Stanford desarrolló una librería oficial para Python “Stanza” [30].
Stanza incluye una interfaz Python para el paquete CoreNLP Java y hereda funcionalidad adicional de allí, como análisis de constituyentes, resolución de correferencia y coincidencia de patrones lingüísticos.

Stanza está construido con componentes de redes neuronales de alta precisión que también permiten un entrenamiento y una evaluación eficientes con sus propios datos anotados. Los módulos están construidos sobre la biblioteca PyTorch.

Para el análisis de sentimientos Stanza usa un clasificador de CNN. El modelo de inglés está entrenado en las siguientes fuentes de datos:

- Stanford Sentiment Treebank, incluidas frases de entrenamiento adicionales [31]
- MELD, solo texto [32]
- SLSD [33]
- Arguana [34]
- Sentimiento de aerolínea en Twitter [35]

La puntuación de este modelo no es directamente comparable a los modelos de SST existentes, ya que utiliza una proyección de 3 clases de los datos de 5 clases e incluye varias fuentes de datos adicionales. Sin embargo, entrenar este modelo en datos de 2 clases usando vectores de palabras de mayor dimensión logra la precisión de 87% reportada en el documento clasificador original de CNN. En una proyección de tres clases de los datos de prueba de SST, el modelo entrenado en múltiples conjuntos de datos obtiene el 70.0%.

SentimentProcessor agrega una etiqueta de sentimiento a cada oración. Admite negativo, neutral y positivo, representados por 0, 1, 2 respectivamente. Los modelos personalizados pueden admitir cualquier conjunto de etiquetas siempre que tenga datos de entrenamiento. En las siguientes líneas de código podremos apreciar la simplicidad de uso que tiene esta librería:

```
nlp = stanza.Pipeline(lang='en', processors='tokenize,sentiment')
doc = nlp("Peter should close his snack bar as soon as possible. Pricing pressures are just too high, and his pool of customers is shrinking constantly. Of course he could do another big ad campaign, but that won't do the trick if the whole neighbourhood is going down the drain in the end.")
for i, sentence in enumerate(doc.sentences):
    print(i, sentence.sentiment)
```

```
0 1
1 0
```

```
20
```

i. Flair

Flair viene con un modelo previamente entrenado de análisis de sentimientos, entrenado con el conjunto de datos de IMDB.

El uso, la descarga y el almacenamiento del modelo se han incorporado en un único método que hace que todo el proceso de uso de modelos previamente entrenados sea sorprendentemente sencillo.

Para utilizar el modelo de análisis de sentimiento, simplemente podemos ejecutar:

```
from flair.models import TextClassifier
from flair.data import Sentence

classifier = TextClassifier.load('en-sentiment')

sentence = Sentence('Flair is pretty neat!')
classifier.predict(sentence)

# print sentence with predicted labels
print('Sentence above is: ', sentence.labels)
```

Flair descarga el modelo de análisis de opiniones y, por defecto, lo almacenará en la subcarpeta .flair del directorio de inicio. Puede tardar unos minutos.

El código anterior primero carga las bibliotecas requeridas, luego carga el modelo de análisis de opinión en la memoria (lo descarga primero si es necesario) y luego predice la puntuación de opinión de la oración "¡Flair es bastante bueno!" en la escala de 0 a 1. Se imprime el comando final:

```
Sentence above is:  [POSITIVE (0.9997)]
```

ii. VADER

```
sentences = ["VADER is smart, handsome, and funny.",
             "VADER is smart, handsome, and funny!",
             "VADER is very smart, handsome, and funny.",
             "VADER is VERY SMART, handsome, and FUNNY.",
             "VADER is VERY SMART, handsome, and FUNNY!!!",
                   "VADER is VERY SMART, uber handsome, and FRIGGIN
FUNNY!!!",
             "VADER is not smart, handsome, nor funny.",
             "The book was good.",
             "At least it isn't a horrible book.",
             "The book was only kind of good.",
              "The plot was good, but the characters are uncompelling
and the dialog is not great.",
             "Today SUX!",
             "Today only kinda sux! But I'll get by, lol",
             "Make sure you :) or :D today!",
             "Catch utf-8 emoji such as such as 💘 and 💋 and 😁",
             "Not bad at all"
             ]
```

```
analyzer = SentimentIntensityAnalyzer()
for sentence in sentences:
    vs = analyzer.polarity_scores(sentence)
    print("{:-<65} {}".format(sentence, str(vs)))
```

```
VADER is smart, handsome, and funny.---------------------------- {'neg': 0.0, 'neu': 0.254, 'pos': 0.746,
'compound': 0.8316}
VADER is smart, handsome, and funny!---------------------------- {'neg': 0.0, 'neu': 0.248, 'pos': 0.752,
'compound': 0.8439}
VADER is very smart, handsome, and funny.----------------------- {'neg': 0.0, 'neu': 0.299, 'pos': 0.701,
'compound': 0.8545}
VADER is VERY SMART, handsome, and FUNNY.----------------------- {'neg': 0.0, 'neu': 0.246, 'pos':
0.754, 'compound': 0.9227}
```

VADER is VERY SMART, handsome, and FUNNY!!!---------- {'neg': 0.0, 'neu': 0.233, 'pos': 0.767, 'compound': 0.9342}
VADER is VERY SMART, uber handsome, and FRIGGIN FUNNY!!!--------- {'neg': 0.0, 'neu': 0.294, 'pos': 0.706, 'compound': 0.9469}
VADER is not smart, handsome, nor funny.---------- {'neg': 0.646, 'neu': 0.354, 'pos': 0.0, 'compound': -0.7424}
The book was good.---------- {'neg': 0.0, 'neu': 0.508, 'pos': 0.492, 'compound': 0.4404}
At least it isn't a horrible book.---------- {'neg': 0.0, 'neu': 0.678, 'pos': 0.322, 'compound': 0.431}
The book was only kind of good.---------- {'neg': 0.0, 'neu': 0.697, 'pos': 0.303, 'compound': 0.3832}
The plot was good, but the characters are uncompelling and the dialog is not great. {'neg': 0.327, 'neu': 0.579, 'pos': 0.094, 'compound': -0.7042}
Today SUX!---------- {'neg': 0.779, 'neu': 0.221, 'pos': 0.0, 'compound': -0.5461}
Today only kinda sux! But I'll get by, lol---------- {'neg': 0.127, 'neu': 0.556, 'pos': 0.317, 'compound': 0.5249}
Make sure you :) or :D today!---------- {'neg': 0.0, 'neu': 0.294, 'pos': 0.706, 'compound': 0.8633}
Catch utf-8 emoji such as such as 💘 and 💋 and 😁---------- {'neg': 0.0, 'neu': 0.615, 'pos': 0.385, 'compound': 0.875}
Not bad at all---------- {'neg': 0.0, 'neu': 0.513, 'pos': 0.487, 'compound': 0.431}

3. Redes Neuronales

a. CNN

Para la siguiente experimentación se utilizará el conjunto de datos de IMDB [39] que contiene 50 mil reseñas.

```
movie_reviews = pd.read_csv("IMDB Dataset.csv")

movie_reviews.isnull().values.any()

movie_reviews.shape
```

```
(50000, 2)
```

Como el conjunto de datos contenía signos de puntuación y etiquetas HTML. En esta sección definiremos una función que toma una cadena de texto como parámetro y luego realiza un preprocesamiento en la cadena para eliminar caracteres especiales y etiquetas HTML de la cadena.

```
def preprocess_text(sen):
    # Removing html tags
    sentence = remove_tags(sen)

    # Remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sentence)

    # Single character removal
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)

    # Removing multiple spaces
    sentence = re.sub(r'\s+', ' ', sentence)

    return sentence

TAG_RE = re.compile(r'<[^>]+>')

def remove_tags(text):
    return TAG_RE.sub('', text)
```

```
X = []
sentences = list(movie_reviews['review'])
for sen in sentences:
    X.append(preprocess_text(sen))
for sen in X:
  print(sen)
```

A continuación, necesitamos convertir nuestras etiquetas en dígitos. Dado que solo tenemos dos etiquetas en la salida, es decir, "positivo" y "negativo". Podemos simplemente convertirlos en números

enteros reemplazando "positivo" con el dígito 1 y negativo con el dígito 0 como se muestra a continuación:

```
y = movie_reviews['sentiment']

y = np.array(list(map(lambda x: 1 if x=="positive" else 0, y)))
```

Finalmente, necesitamos dividir nuestro conjunto de datos en conjuntos de prueba y de entrenamiento. El conjunto de entrenamiento se utilizará para entrenar nuestros modelos de aprendizaje profundo, mientras que el conjunto de pruebas se utilizará para evaluar qué tan bien funciona nuestro modelo. Dividimos nuestros datos en un 80% para el conjunto de entrenamiento y un 20% para el conjunto de prueba.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=42
```

Para preparar la Capa de incrustación (Embedding Layer) utilizaremos la clase Tokenizer del módulo keras.preprocessing.text para crear un diccionario de palabra a índice. En el diccionario de palabra a índice, cada palabra del corpus se usa como clave, mientras que el índice único correspondiente se usa como valor para la clave.

```
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X_train)

X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)
```

Establecemos el tamaño máximo de cada lista en 100. Las listas con un tamaño superior a 100 se truncarán a 100. Para las listas que tengan una longitud inferior a 100, agregaremos 0 al final de la lista hasta que alcance la longitud máxima. Este proceso se llama relleno (padding)

```
# Adding 1 because of reserved 0 index
vocab_size = len(tokenizer.word_index) + 1

maxlen = 100

X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
```

Existen varios tipos de incrustaciones de palabras previamente entrenadas, sin embargo, usaremos las incrustaciones de palabras GloVe de Stanford NLP [40], ya que es la más famosa y de uso común.

Usaremos incrustaciones de GloVe para crear nuestra matriz de características. En el siguiente script cargamos las incrustaciones de palabras GloVe y creamos un diccionario que contendrá palabras como claves y su correspondiente lista incrustada como valores. En las incrustaciones de palabras, cada palabra se representa como un vector denso de n dimensiones. Las palabras que son similares tendrán un vector similar.

```
embeddings_dictionary = dict()
glove_file = open('glove.6B.100d.txt', encoding="utf8")

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dictionary [word] = vector_dimensions
glove_file.close()
```

Finalmente, crearemos una matriz de incrustación donde cada número de fila corresponderá al índice de la palabra en el corpus. La matriz tendrá 100 columnas donde cada columna contendrá las incrustaciones de palabras GloVe para las palabras de nuestro corpus.

```
embedding_matrix = zeros((vocab_size, 100))
for word, index in tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector
```

Se crea una red neuronal convolucional simple con 1 capa convolucional y 1 capa de agrupación.

```
model = Sequential()

Embedding_layer = Embedding(vocab_size, 100,
weights=[embedding_matrix], input_length=maxlen , trainable=False)
model.add(embedding_layer)

model.add(Conv1D(128, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['acc'])
```

En el script anterior creamos un modelo secuencial, seguido de una capa de incrustación. A continuación, creamos una capa convolucional unidimensional con 128 características o núcleos. El tamaño del kernel es 5 y la función de activación utilizada es sigmoidea. A continuación, agregamos una capa de agrupación

máxima global para reducir el tamaño de la entidad. Finalmente agregamos una capa densa con activación sigmoidea.

```
print(model.summary())
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 100, 100)          9254700
_________________________________________________________________
module_wrapper_8 (ModuleWrap (None, 96, 128)           64128
_________________________________________________________________
module_wrapper_9 (ModuleWrap (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 129
=================================================================
Total params: 9,318,957
Trainable params: 64,257
Non-trainable params: 9,254,700
_________________________________________________________________
None
```

Finalmente entrenamos el modelo. Se entrena solo en nuestro conjunto de entrenamiento. El validation_split de 0.2 significa que el 20% de los datos de entrenamiento se usa para encontrar la precisión de entrenamiento del algoritmo.

```
history = model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2)

score = model.evaluate(X_test, y_test, verbose=1)
```

```
Epoch 1/6
250/250 [==============================] - 24s 93ms/step - loss: 0.5002 - acc: 0.7529 - val_loss: 0.3950 - val_acc: 0.8206
Epoch 2/6
250/250 [==============================] - 23s 92ms/step - loss: 0.3698 - acc: 0.8371 - val_loss: 0.3801 - val_acc: 0.8288
Epoch 3/6
250/250 [==============================] - 23s 92ms/step - loss: 0.3192 - acc: 0.8643 - val_loss: 0.3710 - val_acc: 0.8305
```

```
Epoch 4/6
250/250  [==============================]  -  23s  93ms/step  -  loss:
0.2813 - acc: 0.8856 - val_loss: 0.3880 - val_acc: 0.8184
Epoch 5/6
250/250  [==============================]  -  23s  92ms/step  -  loss:
0.2523 - acc: 0.8998 - val_loss: 0.3440 - val_acc: 0.8447
Epoch 6/6
250/250  [==============================]  -  23s  93ms/step  -  loss:
0.2142 - acc: 0.9240 - val_loss: 0.3383 - val_acc: 0.8524
313/313  [==============================]  -  3s  10ms/step  -  loss:
0.3382 - acc: 0.8520
```

Al final del entrenamiento, verá que la precisión del entrenamiento es de alrededor del 85,20%.

```
score = model.evaluate(X_test, y_test, verbose=1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

```
Test Score: 0.33822059631347656
Test Accuracy: 0.8519999980926514
```

Si se compara la precisión del entrenamiento y la prueba, se verá que la precisión del entrenamiento para CNN será de alrededor del 92%. La precisión de la prueba es de alrededor del 85% para la CNN.

Tracemos la pérdida y la diferencia de precisión entre el entrenamiento y el conjunto de prueba.

```
import matplotlib.pyplot as plt

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])d

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()
```

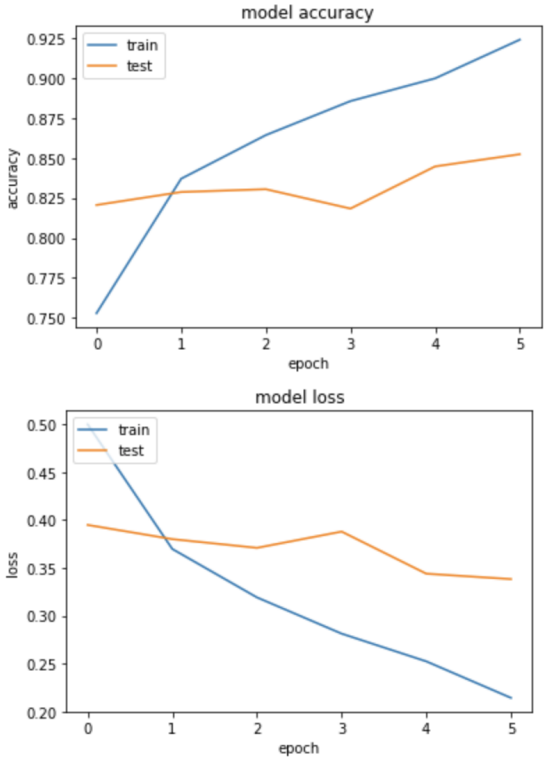

Figura 6.Gráfico de la pérdida y la diferencia de precisión entre el entrenamiento y el conjunto de prueba para CNN

Se puede ver claramente las diferencias de pérdida y precisión entre los conjuntos de entrenamiento y los de prueba.

b. RNN

i. LSTM

Usando el mismo conjunto de datos e incrustaciones de palabras de Glove definimos el el modelo

```
model = Sequential()
```

```
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=maxlen , trainable=False)
model.add(embedding_layer)
model.add(LSTM(128))

model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

Inicializamos un modelo secuencial seguido de la creación de la capa de incrustación. A continuación, creamos una capa LSTM con 128 neuronas.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 100)          9254700
_________________________________________________________________
module_wrapper (ModuleWrappe (None, 128)               117248
_________________________________________________________________
dense (Dense)                (None, 1)                 129
=================================================================
Total params: 9,372,077
Trainable params: 117,377
Non-trainable params: 9,254,700
_________________________________________________________________
None
```

Se entrena el modelo en el conjunto de entrenamiento y evaluamos el desempeño en el conjunto de prueba.

```
history = model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2)

score = model.evaluate(X_test, y_test, verbose=1)
```

El código anterior entrena el modelo en el conjunto de prueba. El tamaño del lote es 128, mientras que el número de epochs es 6

```
Epoch 1/6
250/250 [==============================] - 87s 341ms/step - loss: 0.5463 - acc: 0.7180 - val_loss: 0.4721 - val_acc: 0.7790
Epoch 2/6
```

```
250/250 [==============================] - 85s 341ms/step - loss:
0.4410 - acc: 0.7954 - val_loss: 0.4131 - val_acc: 0.8134
Epoch 3/6
250/250 [==============================] - 85s 341ms/step - loss:
0.3920 - acc: 0.8233 - val_loss: 0.3691 - val_acc: 0.8357
Epoch 4/6
250/250 [==============================] - 85s 342ms/step - loss:
0.3632 - acc: 0.8382 - val_loss: 0.3550 - val_acc: 0.8469
Epoch 5/6
250/250 [==============================] - 85s 341ms/step - loss:
0.3405 - acc: 0.8508 - val_loss: 0.3540 - val_acc: 0.8474
Epoch 6/6
250/250 [==============================] - 85s 341ms/step - loss:
0.3266 - acc: 0.8562 - val_loss: 0.3610 - val_acc: 0.8499
313/313 [==============================] - 10s 31ms/step - loss:
0.3589 - acc: 0.8488
```

Al final del entrenamiento, podemos ver que la precisión del entrenamiento es de alrededor del 85,00%.

Una vez que se entrena el modelo, podemos ver los resultados del modelo en el conjunto de prueba con el siguiente código:

```
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

```
Test Score: 0.3588893711566925
Test Accuracy: 0.848800003528595
```

En el resultado, vemos que la precisión de nuestra prueba es de alrededor del 85,00%.

A continuación se visualiza la pérdida y las diferencias de precisión entre los conjuntos de entrenamiento y de prueba.

```
import matplotlib.pyplot as plt

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()
```

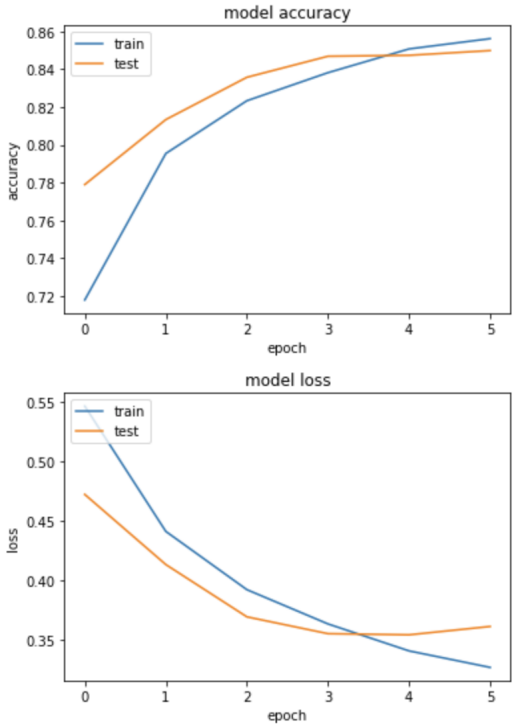

Figura 7. Gráfico de la pérdida y la diferencia de precisión entre el entrenamiento y el conjunto de prueba para LSTM

4. Comparación

Según la experimentación se puede concluir que de las librerías exclusivas para el análisis de sentimientos VADER sería la más apta para nuestra problemática, ya que fue desarrollada para detectar sentimientos expresados en redes sociales. VADER comprende el énfasis en mayúsculas y la puntuación.

La ventaja de CoreNLP y Flair es que admiten varios lenguajes en cambio VADER sólo admite soluciones en inglés. Otra ventaja de Flair es que permite entrenar un clasificador de texto personalizado y permite combinar diferentes incrustaciones de palabras (word embedding) y usarlas para incrustar documentos.

En comparación a VADER y coreNLP hacer clasificación de texto con Flair es una tarea de nivel relativamente bajo. Tenemos un control total de cómo se realiza la incrustación de texto y el entrenamiento al tener una opción para establecer parámetros como la tasa de aprendizaje, el tamaño del lote, el factor de recocido, la función de pérdida, la selección del optimizador. Pero la desventaja de Flair es que el análisis de sentimiento está entrenado en datos de IMDB y este modelo podría no generalizarse bien en datos de otros dominios como Twitter.

Aun así con las redes neuronales podemos customizar aún más el pre-procesamiento y así tener un mejor resultado. En la siguiente tabla compararemos algunas métricas para RNN y Flair

Para la comparación, se usa el mismo conjunto de datos que se usó en la demostración de CNN, el conjunto de datos de IMDB.

Modelo	Precision	Customizacion	Dificultad de implementación
Flair	~51%	Permite entrenar un clasificador de texto personalizado y usar diferentes word embedding y establecer diferentes parámetros	Solo requiere unas líneas de código y no precisa (si no se quiere) de pre-procesar el texto.
CNN	~82%	Podemos controlar cada aspecto del modelo y además customizar el preprocesamiento de texto	Se debe hacer un poco más de “trabajo” para implementarlo. Requiere pre-procesar el texto y saber cómo funcionan los parámetros del modelo para un resultado apropiado. Conv1D requiere más parámetros para instanciarlo como el tamaño del kernel, etc.
LSTM	~85%	Podemos controlar cada aspecto del modelo y además customizar el preprocesamiento de texto	Se debe hacer un poco más de “trabajo” para implementarlo. Requiere pre-procesar el texto y saber cómo funciona para parámetro del modelo

			para un resultado apropiado

Tabla 1. Comparación usando el conjunto de datos de IMDB entre Flair, CNN y LSTM

Con la experimentación el modelo de CNN está sobreajustado (en inglés, overfitting), ya que existe una gran diferencia entre la precisión del entrenamiento y la prueba. Podemos comprobarlo mirando la figura 6.

Un modelo está sobreajustado si durante el entrenamiento, las métricas del modelo fueron buenas, pero cuando usamos el modelo para predecir los datos de prueba, no clasifica con precisión los datos en el conjunto de prueba.

El concepto de sobreajuste se reduce al hecho de que el modelo no puede generalizar bien. Ha aprendido las características del conjunto de entrenamiento extremadamente bien, pero si le damos al modelo cualquier dato que se desvíe levemente de los datos exactos utilizados durante el entrenamiento, no podrá generalizar y predecir con precisión el resultado.

En cambio el modelo LSTM la precisión de la prueba es mejor que la CNN. Además, podemos ver en la figura 7 que hay una diferencia muy pequeña entre la precisión del entrenamiento y la precisión de la prueba, lo que significa que nuestro modelo no está sobreajustado.

Dado la experimentación y resultados vemos más apta la solución con LSTM y por lo tanto realizaremos el prototipo con el mencionado a continuación.

Diseño del prototipo

1. Conjunto de Datos

Se utilizó el conjunto de datos de Kaggle, “Sentiment140 dataset with 1.6 million tweets” [43]. Contiene 1,600,000 tweets extraídos usando la API de Twitter. Los tweets han sido anotados (0 = negativo, 4 = positivo) y se pueden usar para detectar sentimientos.

Pre-procesamiento

Como se realizó en la experimentación “limpiamos” el texto para que el modelo pueda entenderlo mejor.

```
def format_text(df,col):
  comp_df = df.copy()

  # eliminamos @ y puntuación
  comp_df[col] = comp_df[col].str.replace(r'(@\w*)','')

  # eliminamos links
  comp_df[col] = comp_df[col].str.replace(r"http\S+", "")

  # eliminamos # y las palabra que lo sigue
  comp_df[col] = comp_df[col].str.replace(r'#\w+',"")

  # eliminamos todos lo que no sea caracteres
  comp_df[col] = comp_df[col].str.replace(r"[^a-zA-Z ]","")

  # eliminamos espacios extras
  comp_df[col] = comp_df[col].str.replace(r'( +)'," ")
  comp_df[col] = comp_df[col].str.strip()

  # eliminamos mayúsculas
  comp_df[col] = comp_df[col].str.lower()

  return comp_df

formated_comp_df = format_text(data,'tweet')
tweets = np.array(formated_comp_df.iloc[:, 2].values)
sentiment = np.array(data.iloc[:, 0].values)
```

Tokenización

De la misma forma que se realizó en la experimentación con LSTM y CNN, para capa de incrustación (Embedding Layer) utilizaremos la clase Tokenizer del módulo keras.preprocessing.text para crear un diccionario de palabra a índice. Establecemos el tamaño máximo de cada lista en 30 y se realiza el padding (relleno)

```
maxlen=30
tk = Tokenizer()
#tw = tweets
tk.fit_on_texts(tweets)
t = tk.texts_to_sequences(tweets)
X = np.array(sequence.pad_sequences(t, maxlen=maxlen, padding='post'))
y = sentiment

print(X.shape, y.shape)
```

```
(1600000, 30) (1600000,)
```

Como en la experimentación con CNN y LSTM se utiliza incrustaciones de palabras previamente entrenada y también como en la experimentación se usa Glove. En este caso se utilizaron vectores de 100 dimensiones de palabras pre-entrenados con 2 billones de tweets por GloVe [44]. El siguiente paso es cargar las incrustaciones de palabras de GloVe y luego crear nuestra matriz de incrustación que contiene las palabras en nuestro corpus y sus valores correspondientes de las incrustaciones de GloVe.

```
embeddings_dictionary = dict()
glove_file = open('glove.twitter.27B.200d.txt', encoding="utf8")

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dictionary [word] = vector_dimensions
glove_file.close()
```

El diccionario **embeddings_dictionary** ahora contiene palabras y las correspondientes incrustaciones GloVe para todas las palabras. Se quiere incrustaciones de palabras sólo para aquellas palabras que están presentes en el corpus. Se crea, entonces, una matriz numérica bidimensional de 415225 (tamaño del vocabulario) filas y 200 columnas. La matriz inicialmente contendrá ceros. La matriz se denominará **embedding_matrix**

A continuación, se itera a través de cada palabra del corpus recorriendo el diccionario **tk.word_index** que contiene las palabras y su índice correspondiente.

Cada palabra se pasará como clave al **embeddings_dictionary** para recuperar el vector de 200 dimensiones correspondiente a la palabra. El vector de 200 dimensiones se almacenará en el índice correspondiente de la palabra en **embedding_matrix**.

```
word_index = tk.word_index
vocab_size = len(word_index)+1

embedding_matrix = zeros((vocab_size, 200))
for word, index in tk.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector
```

2. Modelo de las redes neuronales

Se desarrolló un modelo de redes neuronales LSTM bidireccional.

LSTM bidireccional

La memoria bidireccional a largo-corto plazo (bi-lstm) es el proceso de hacer que cualquier red neuronal tenga la información de secuencia en ambas direcciones hacia atrás (del futuro al pasado) o hacia adelante (del pasado al futuro) [41] .

En bidireccional, la entrada fluye en dos direcciones, lo que hace que un bi-lstm sea diferente del LSTM normal. Con el LSTM normal, la entrada puede fluir en una dirección, ya sea hacia atrás o hacia adelante. Sin embargo, en bidireccional, la entrada puede fluir en ambas direcciones para preservar la información pasada y futura.

En la figura 8 se observa el flujo de información de las capas hacia atrás y hacia adelante. BI-LSTM se emplea generalmente cuando se necesita la secuencia para secuenciar tareas. Este tipo de red se puede utilizar en modelos de clasificación de texto, como es el caso presente.

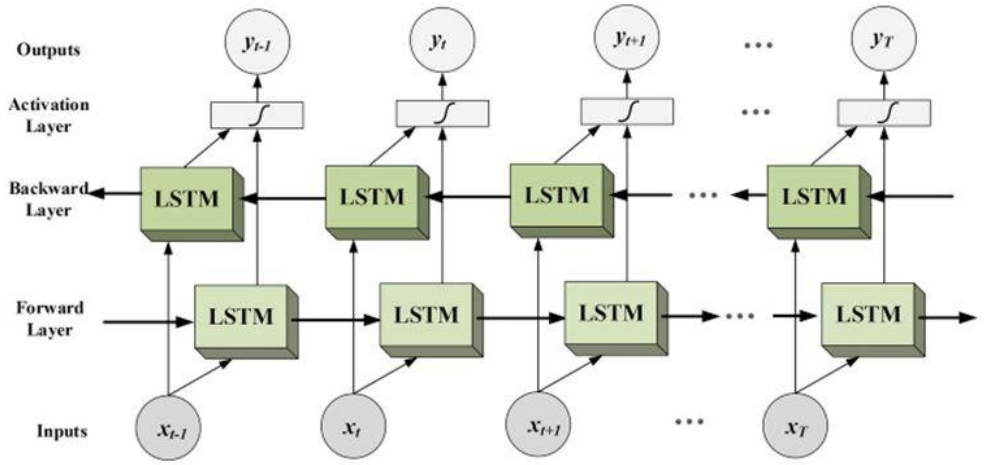

Figura 8. bi-LSTM [42]

El modelo propuesto:

```
# en el conjunto de datos los positivos se definen con 4, lo cambiamos a 1
 y[y == 4] = 1

 model = Sequential([
          Embedding(input_dim=vocab_size,
                    output_dim=200,
                    input_length=maxlen,
                   weights=[embedding_matrix],
                   trainable=False),
          Bidirectional(LSTM(64,dropout=0.3,return_sequences=True)),
          Dropout(0.3),
          Bidirectional(LSTM(64,dropout=0.3)),
          Dropout(0.3),
          Dense(1,activation='sigmoid')])

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['acc'])
```

En este modelo se usó LSTM bidireccional apilado. La primera capa bidireccional se define con 64 celdas y la segunda capa se define con, también, 64 unidades LSTM bidireccionales. La capa final es la capa de salida con función de activación sigmoidea, ya que nuestro problema es un problema de clasificación

binaria, se utiliza la función sigmoidea. Como en la experimentación se obtuvo sobreajuste (overfitting) también se utilizan capas de dropout para evitarlo.

El **dropout** es un método de regularización donde la entrada y las conexiones recurrentes a las unidades LSTM se excluyen probabilísticamente de la activación y las actualizaciones de peso mientras se entrena una red. Esto tiene el efecto de reducir el sobreajuste y mejorar el rendimiento del modelo.

Un **dropout** en la entrada del LSTM significa que, para una probabilidad determinada, los datos de la conexión de entrada a cada bloque LSTM se excluirán de la activación del nodo y las actualizaciones de peso.

En Keras, esto se especifica con un argumento de **dropout** al crear una capa LSTM. El valor de **dropout** es un porcentaje entre 0 (sin abandono) y 1 (sin conexión).

Se usa la función de optimización de Adam para la retropropagación y se usa la función de pérdida de entropía cruzada binaria para la pérdida y precisión para la métrica. La función de pérdida se usa para optimizar el modelo, mientras que la métrica se usa para la comparación.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 30, 200)           83044800
_________________________________________________________________
bidirectional (Bidirectional (None, 30, 128)           135680
_________________________________________________________________
dropout (Dropout)            (None, 30, 128)           0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 128)               98816
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 1)                 129
=================================================================
Total params: 83,279,425
Trainable params: 234,625
Non-trainable params: 83,044,800
_________________________________________________________________
None
```

3. Entrenamiento red neuronal

Como en se realizó en la experimentación, se entrena el modelo de la siguiente forma

```
history = model.fit(X,y,
epochs=10,validation_split=0.2,verbose=1,batch_size=120)
```

Para batch_size, el valor 120 dio los mejores resultados sin tener que esperar tanto tiempo. Si se reduce este valor el modelo tardará mucho en entrenarse. Para el `validation_split` usa el último 20% de los datos para su validación. Se ejecuta por 10 epochs

```
Epoch 1/10
10667/10667 [==============================] - 2695s 252ms/step -
loss: 0.4347 - acc: 0.7971 - val_loss: 0.3939 - val_acc: 0.8214
Epoch 2/10
10667/10667 [==============================] - 2755s 258ms/step -
loss: 0.4018 - acc: 0.8168 - val_loss: 0.3804 - val_acc: 0.8285
Epoch 3/10
10667/10667 [==============================] - 2645s 248ms/step -
loss: 0.3918 - acc: 0.8223 - val_loss: 0.3751 - val_acc: 0.8315
Epoch 4/10
10667/10667 [==============================] - 2664s 250ms/step -
loss: 0.3862 - acc: 0.8254 - val_loss: 0.3744 - val_acc: 0.8335
Epoch 5/10
10667/10667 [==============================] - 2715s 255ms/step -
loss: 0.3819 - acc: 0.8280 - val_loss: 0.3727 - val_acc: 0.8338
Epoch 6/10
10667/10667 [==============================] - 2698s 253ms/step -
loss: 0.3787 - acc: 0.8294 - val_loss: 0.3704 - val_acc: 0.8344
Epoch 7/10
10667/10667 [==============================] - 2628s 246ms/step -
loss: 0.3768 - acc: 0.8306 - val_loss: 0.3688 - val_acc: 0.8359
Epoch 8/10
10667/10667 [==============================] - 2625s 246ms/step -
loss: 0.3750 - acc: 0.8315 - val_loss: 0.3672 - val_acc: 0.8366
Epoch 9/10
10667/10667 [==============================] - 2608s 244ms/step -
loss: 0.3737 - acc: 0.8323 - val_loss: 0.3713 - val_acc: 0.8363
Epoch 10/10
10667/10667 [==============================] - 2615s 245ms/step -
loss: 0.3731 - acc: 0.8323 - val_loss: 0.3704 - val_acc: 0.8360
```

Evaluamos el desempeño en el conjunto de prueba.

```
score = model.evaluate(X, y, verbose=1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

```
Test Score: 0.35232028365135193
Test Accuracy: 0.8441399931907654
```

Se logró una precisión de 84.41%

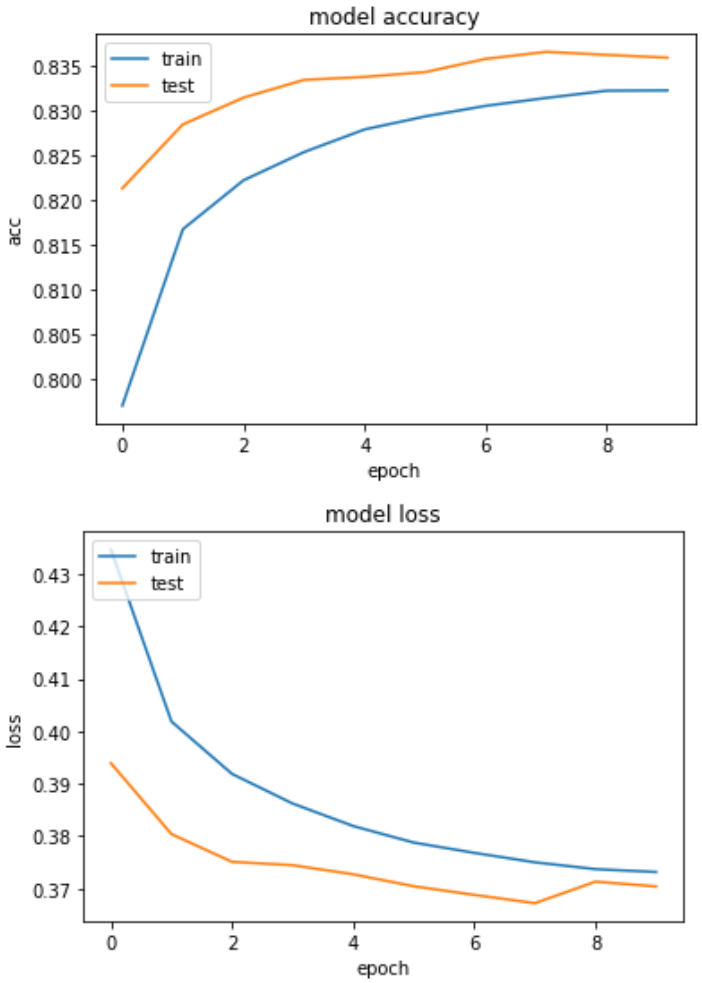

Figura 9. Gráfico de la pérdida y la diferencia de precisión entre el entrenamiento y el conjunto de prueba para el modelo LSTM bidireccional

4. Front End y APIs

El prototipo consiste en demostrar que el modelo que se desarrolló anteriormente puede clasificar en positivo o negativo, bajo un grado de precisión de más de 80% texto recopilado de Twitter y Reddit. El prototipo utiliza la API de Twitter, Tweepy [45] y Reddit, PRAW [46]. En la línea de comandos se podrá elegir qué API usar. En el caso de Reddit la API buscará comentarios (limitando a 10) en el primer posteo activo del subreddit /r/politics. Para Twitter se tendrá que brindar de input una palabra clave a buscar y se listará los primeros 10 tweets. Con el listado devuelto por la API elegida se predice con el modelo LSTM Bidireccional entrenado.

Se define la función para obtener el listado de comentarios del posteo más controversial del mes. La función retorna un array con los comentarios (limitamos a 10)

```
def reddit_fetch():
  reddit = praw.Reddit(
      client_id = client_id,
      client_secret = client_secret,
      user_agent = user_agent,
  )

  subreddit = reddit.subreddit("politics")
  reddit_comments_array = []
  for submission in
subreddit.controversial(limit=1,time_filter='month'):
      for top_level_comment in submission.comments:
        if isinstance(top_level_comment, MoreComments):
            breakpoint
        else:
            reddit_comments_array.append(top_level_comment.body)

  reddit_comments_array.pop(0)
  return reddit_comments_array[:11]
```

Para twitter se define una función que solicita al usuario una palabra clave, con el cual buscará tweets que lo contengan. La función retorna una array de los 10 primeros tweets con la palabra clave ingresada.

```
def twitter_fetch():
  consumer_key = consumer_key
  consumer_secret = consumer_secret
  access_token = access_token
  access_token_secret = access_token_secret

  auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
  auth.set_access_token(access_token, access_token_secret)
  api = tweepy.API(auth,wait_on_rate_limit=True)


  val = input("Ingresar palabra a buscar: ")
  print(val)

  twitter_comments_array = []
  for tweet in tweepy.Cursor(api.search,q=val,count=50,
                             lang="en").items(10):
      twitter_comments_array.append(tweet.text)
  return twitter_comments_array
```

Para inicializar el programa se corre el siguiente código:

```
api_val = input("Seleccionar API: \n1)Twitter \n2)Reddit \n")

fetched_array = []
if (api_val == "1"):
  fetched_array = twitter_fetch()
else:
  fetched_array = reddit_fetch()

clean_fetched_array = []
for comment in fetched_array:
  clean_fetched_array.append(format_text_predict(comment))
clean_fetched_array
```

Con el código anterior se solicita al usuario que API usar. Se llama a la función correspondiente y se realiza la “limpieza” o preprocesamiento del texto. Si lo ejecutamos se obtiene lo siguiente:

```
Seleccionar API:
1)Twitter
2)Reddit
```

```
1
Ingresar palabra a buscar: love
love

['love both of their trajectories mac has the upper hand for now',
 'the moons gonna be crazy cause i think im in love with you',
 'the ones actually winning an not fake are the ones who get the
least love',
 'women really wake up one day and say yea i dont like him anymore
its not gonna work out and i love that for us',
 'stevie ray vaughn amp double trouble aint gone n give up on love',
 'everytime you put ateez on one of these shows you can hear dozens
of crew absolutely losing it in the back no wonder all s',
 'no matter how much you love loves him more',
 'i love not being american cos im actually not white washed',
 'people will chop breakfast and turn to overnight motivational
speakersi love the kinikan im becomingjumoke if',
 'this is my little brother he been missing for weeks as of today now
please help me and my family find him one retwee']
```

Tal como se realizó en el desarrollo del modelo en la sección anterior, para que entienda el modelo el texto de entrada se transforma cada texto en una secuencia de enteros y se realiza el “padding”

```
predictNow = clean_fetched_array
tpredict = tk.texts_to_sequences(predictNow)
Xpredict = np.array(sequence.pad_sequences(tpredict, maxlen=30,
padding='post'))
```

Ahora se carga el modelo que entrenamos y predecimos:

```
new_model = keras.models.load_model('lstmbidirectional.h5')
analysis = new_model.predict(Xpredict)
```

Se asignó las salidas positivas a 1 y las salidas negativas a 0. Sin embargo, la función sigmoidea predice un valor flotante entre 0 y 1. Si el valor es menor que 0.5, el sentimiento se considera negativo, y si el valor fuera mayor que 0.5, el sentimiento se considera positivo.

```
array([[0.5811826 ],
       [0.88521236],
       [0.8431756 ],
       [0.3977375 ],
       [0.2775379 ],
```

```
       [0.16169491],
       [0.95347226],
       [0.42038202],
       [0.9128811 ],
       [0.11785111]], dtype=float32)
```

Se agregó un formato más legible y además se clasificó al texto con valor entre 0.45 a 0.59 como neutral.

text	sentiment
love both of their trajectories mac has the upper hand for now	neutral
the moons gonna be crazy cause i think im in love with you	positive
the ones actually winning an not fake are the ones who get the least love	positive
women really wake up one day and say yea i dont like him anymore its not gonna work out and i love t..	negative
stevie ray vaughn amp double trouble aint gone n give up on love	negative
everytime you put ateez on one of these shows you can hear dozens of crew absolutely losing it in th..	negative
no matter how much you love loves him more	positive
i love not being american cos im actually not white washed	negative
people will chop breakfast and turn to overnight motivational speakersi love the kinikan im becoming..	positive
this is my little brother he been missing for weeks as of today now please help me and my family fin..	negative

5. Casos de uso

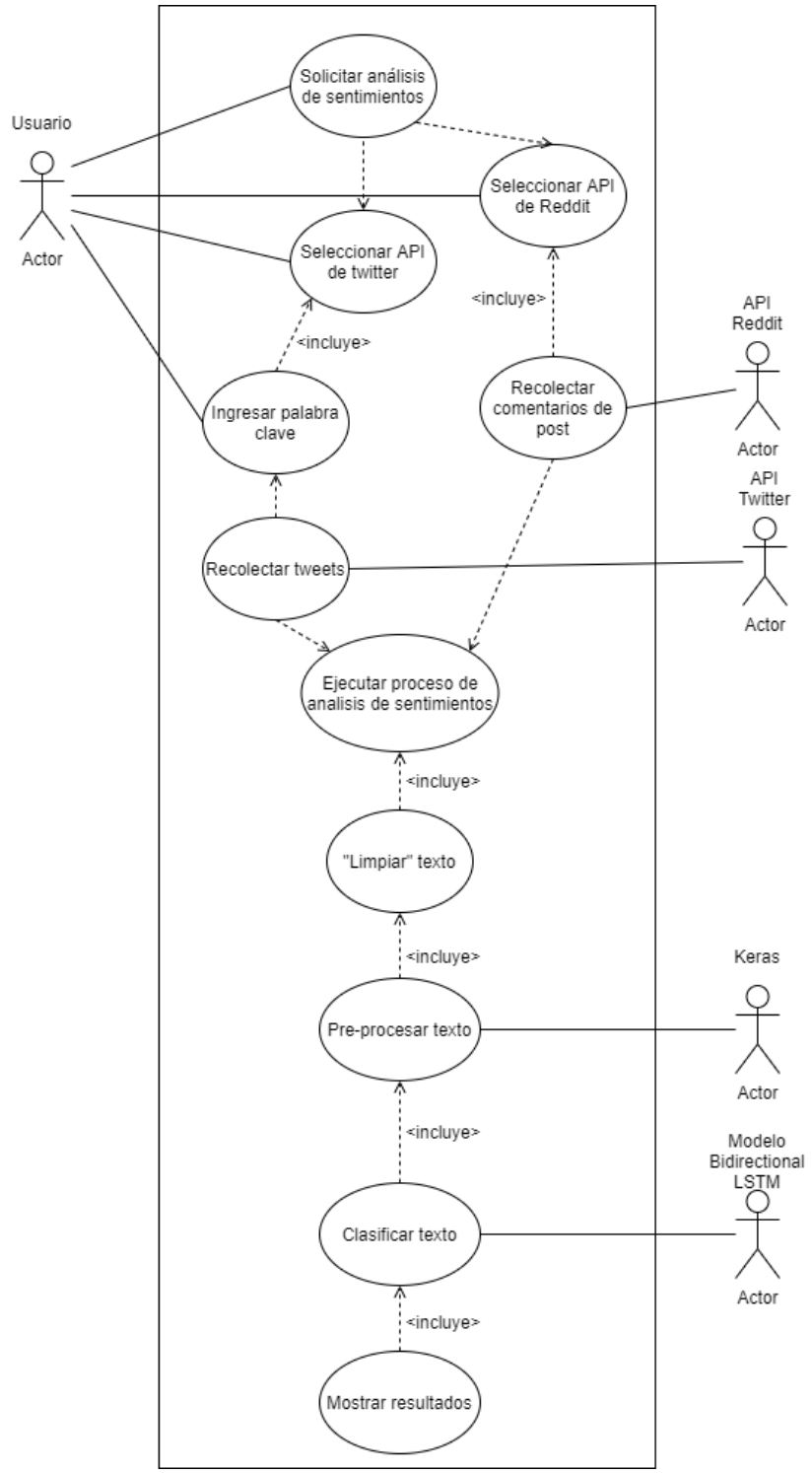

Figura 10. Diagrama de casos de uso del prototipo

En la sección Anexos se detalla cada caso de uso escrito.

Ambiente de entrenamiento

1. Ambiente de prueba

El modelo entrenado se ejecuta sobre la siguiente infraestructura

a. Software

- **i.** Python [47]: es un lenguaje de programación interpretado. Se eligió este lenguaje porque es uno de los lenguajes más populares para trabajar en el campo de la Inteligencia Artificial y para abordar los problemas relacionados con el Procesamiento del Lenguaje Natural
- **ii.** Pandas [48]: es una librería de código abierto que proporciona estructuras de datos y herramientas de análisis de datos de alto rendimiento y fáciles de usar para el lenguaje de programación Python. Se usó esta librería para leer el .zip del dataset
- **iii.** NumPy [49]: es una librería para la computación científica en Python. Proporciona soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ella.
- **iv.** Tweepy [45]: es una biblioteca de Python para acceder a la API de Twitter.
- **v.** PRAW [46]: es una librería de Python que permite un acceso simple a la API de Reddit. PRAW tiene como objetivo ser fácil de usar y sigue internamente todas las reglas de la API de Reddit.
- **vi.** Keras [50]: Keras es una biblioteca de Redes Neuronales de Código Abierto escrita en Python. Keras funciona como una interfaz de uso intuitivo (API) que permite acceder a varios frameworks de aprendizaje automático y desarrollarlos. Proporciona bloques modulares sobre los que se pueden desarrollar modelos complejos de aprendizaje profundo.

b. Hardware

Toda la experimentación, desarrollo y ejecución se realizó usando Google Colaboratory [51]. Colab es un entorno portátil Jupyter [52] gratuito que se ejecuta completamente en la nube. Colab permite escribir y

ejecutar código Python en la nube de Google. Colab nos brinda acceso a todas las librerías mencionadas en la sección anterior. Las especificaciones técnicas son las siguientes:

i. **Procesador:** Intel(R) Xeon(R) CPU @ 2.20GHz
ii. **RAM:** 12,69GB
iii. **Disco:** 107,72GB

Conclusiones

En el desarrollo de esta tesina se logró el análisis y comparación del estado de arte del análisis de sentimientos usando técnicas de procesamiento del lenguaje natural. Se comparó entre diferentes librerías de pre-procesamiento de texto, entre librerías que ofrecen una herramienta para el análisis de sentimientos y entre redes neuronales. Se concluyó que entre la herramienta cerrada y las redes neuronales, este último es la opción más adecuada para la problemática presente, ya que se puede manipular la entrada, entrenamiento, y los parámetros del modelo para adaptarlo a diferentes casos. Finalmente se desarrolló un prototipo con un modelo LSTM Bidireccional. Se realizó el entrenamiento y evaluación usando un conjunto de datos [46] de 1,600,000 tweets logrando una precisión de 84,41%. Luego, se demostró su eficacia prediciendo el sentimiento de un listado de texto provisto por la API de Twitter y Reddit.

Con los resultados obtenidos de la predicción, como era de esperar, se observó un mejor rendimiento con el texto provisto por Twitter. Esto se debe a ciertos factores. Para empezar, el conjunto de datos usado para el entrenamiento y evaluación era de tweets. Se puede asumir que el contexto de un tweet será muy diferente al de un comentario en un posteo específico. En el caso presente se recopila texto de comentarios de un foro de política, así que se pudo observar una temática y contexto a el texto de estos. En cambio, en los tweets se pudo observar texto sin contexto o de simple comprensión, de opinión fácil de deducir a simple vista. Se puede demostrar esto observando la figura 11 de un tweet y la figura 12 de un comentario extraido del foro de política.

Figura 11. Un tweet provisto por twitter.

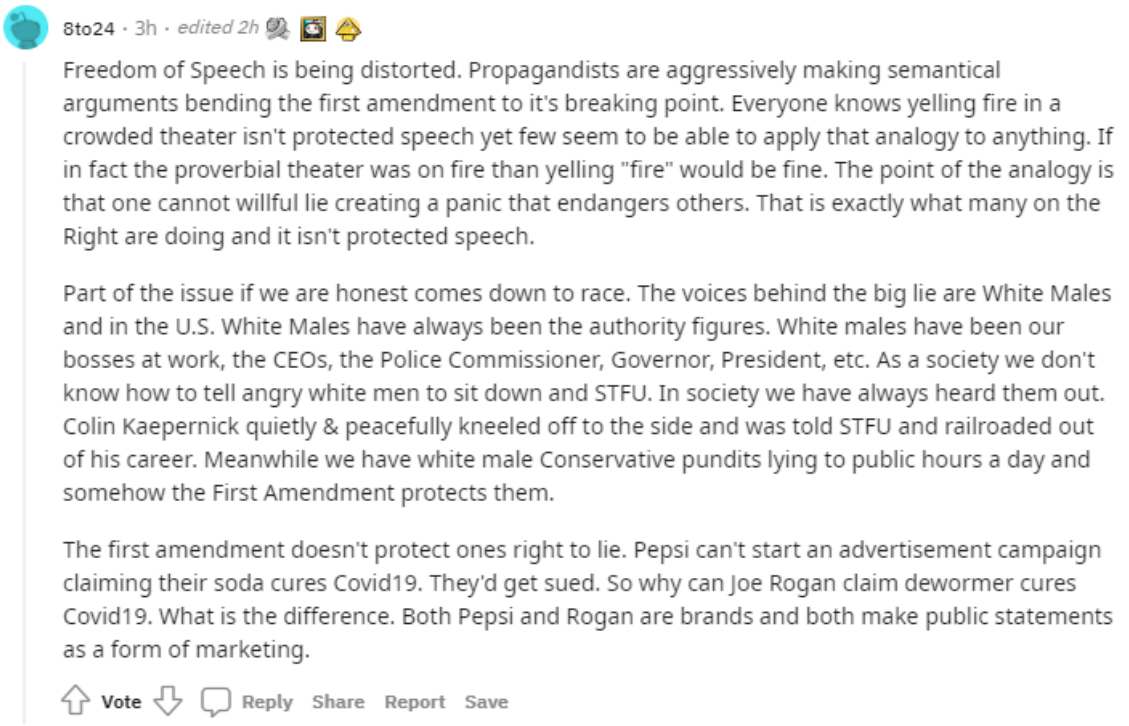

Figura 12. Un comentario en un posteo provisto por Reddit.

Se concluyó además que el que los tweets tuvieron predicciones más acertadas por la particularidad de que estos suelen ser más cortos, máximo de 280 caracteres, en cambio un comentario en Reddit tiene límite de 10.000 caracteres por comentario. Finalmente el factor a tener en cuenta para entender el rendimiento resultado es que la lista de cada texto aceptado por la capa de incrustación tiene un máximo de solo 30 caracteres. Se trabajó con ese valor debido a los tiempos de entrenamiento. Cuando se cambiaba el valor a uno mayor se multiplicaba el tiempo que tardaba entrenar el modelo y provocaba que el entorno de ejecución se desconectará.

Resumen conclusiones finales	
Conclusion #1	Entre las librerías exclusivas para análisis de sentimientos y los modelos de redes neuronales, este último tiene mejor rendimiento y mejor personalización para adaptarlo a diferentes problemáticas
Conclusion #2	Se logró una precisión mayor 80%. Respetando el alcance que se definió
Conclusion #3	Se obtuvo mejor rendimiento con el texto provisto de Twitter que

	Reddit
Conclusion #4	Los tweets, en general, no tienen contexto o son de fácil comprensión. En cambio, un posteo de Reddit tiene contexto y suelen ser mucho más extensos que un tweet
Conclusion #5	La lista de cada texto aceptado por la capa de incrustación tiene un máximo de solo 30 caracteres, aceptando por lo tanto texto de solo 30 caracteres. Si este valor se eleva, el tiempo de entrenamiento se multiplica.

Tabla 2. Resumen de conclusiones finales

Líneas futuras de investigación

En el trabajo presente se investigó y analizó dos redes neuronales que lograron buenos porcentajes de precisión pero el prototipo deja muchas posibilidades de crecimiento. En un futuro se puede usar otros modelos de redes neuronales o deep learning.

El resultado concluido demostró que tal vez una mejor opción es la de desarrollar y entrenar dos modelos distintos dependiendo la red social a analizar. Esto se debe a lo distinto en contexto que presenta cada uno de los textos provistos.

Además se puede variar la técnica de "limpieza" del texto en el pre-procesamiento. Una opción que recomienda GloVe [40] para el preprocesamiento es agregar un tag en lugar de eliminar las palabras o puntuaciones no deseadas. Por ejemplo, agregar <user> en lugar de eliminar el usuario mencionado en un tweet. En el prototipo no se incluyó este pre-procesamiento debido al tamaño máximo de nuestra lista (30).

Finalmente se puede desarrollar un front-end web donde en el back-end este alojado el modelo y se podría manipular y presentar los resultados. Además se podría desarrollar un diseño accesible a cualquier usuario.

Bibliografia

[1] R. Murphy, *Local Consumer Review Survey 2020*, BrightLocal, Dec. 2020. Ultimo acceso: Ago. 23, 2020. [En línea]. Disponible en: https://www.brightlocal.com/research/local-consumer-review-survey/#search-frequency

[2] S. Agrawal, *Sentiment Analysis using LSTM*, Feb. 2019. Ultimo Acceso: Ago. 23 , 2020. [En Linea]. Disponible en: https://towardsdatascience.com/sentiment-analysis-using-lstm-step-by-step-50d074f09948

[4] Talkwalker, *Talkwalker*. Ultimo Acceso: Ago. 24, 2020. [En Linea]. Disponible en: https://www.talkwalker.com/quick-search-form

[5]A. Oppermann, *Sentiment Analysis with Deep Learning of Netflix Reviews*, Feb. 2019. Ultimo Acceso: Ago. 17 , 2020. [En Linea]. Disponible en: https://towardsdatascience.com/sentiment-analysis-with-deep-learning-62d4d0166ef6

[6] N. Sinha, *Understanding lstm and it´s quick implementation in keras for sentiment analysis*, Feb. 2018. Último acceso: Ago. 23, 2020. [En línea]. Disponible en: https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47

[7] P. Rao, *Fine-grained Sentiment Analysis in Python*, Sept. 2019. Ultimo acceso: Ago. 23, 2020. [En línea]. Disponible en: https://towardsdatascience.com/fine-grained-sentiment-analysis-in-python-part-1-2697bb111ed4

[8] B.Pang, L. Lee, *Thumbs up? Sentiment Classification using Machine Learning Techniques*, Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Jul. 2002. Último acceso: Oct. 10, 2020. [En línea]. Disponible en: https://dl.acm.org/doi/pdf/10.3115/1118693.1118704

[9] N. Nedjah, I. Santos & L. Mourelle, *Sentiment analysis using convolutional neural network via word embeddings*, Evol. Intel, Abr. 2019. Ultimo acceso: Nov. 15, 2020. [En línea]. Disponible en: https://link.springer.com/article/10.1007/s12065-019-00227-4

[10] K. Baktha, B K Tripathy, *Investigation of Recurrent Neural Networks in the field of Sentiment Analysis*, Electrical Computer and Communication Engineering (ECCE) 2019 International Conference, Abr. 2017. Ultimo acceso: Nov. 23, 2020. [En línea]. Disponible en: https://ieeexplore.ieee.org/abstract/document/8286763

---

[11]R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng and C. Potts, *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank*, Stanford University, Oct. 2013. Último acceso: Dic. 30, 2020. [En línea]. Disponible en: https://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf

[12] Qingyuan Zhou, Zheng Xu, Neil Y. Yen, *User sentiment analysis based on social network information and its application in consumer reconstruction intention*, Jul. 2018. Último acceso: Dic. 30, 2020. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/abs/pii/S0747563218303285

[13] L. Ziora, *The sentiment analysis as a tool of business analytics in contemporary organizations*, 2016. Último acceso: Dic. 30, 2020. [En línea]. Disponible en: http://yadda.icm.edu.pl/yadda/element/bwmeta1.element.cejsh-bc989119-d00a-4bff-ac7a-77478fadaa14/c/19.pdf

[15] A. C. R. Martins, *Mobility and social network effects on extremist opinions*, Sep. 2008. Último acceso: Dic. 30, 2020. [En línea]. Disponible en: https://journals.aps.org/pre/abstract/10.1103/PhysRevE.78.036104

[16] B. Agarwal, R. Nayak, N. Mittal, S. Patnaik, *Deep Learning-Based Approaches for Sentiment Analysis*, 2020. Último acceso: Ene. 11, 2021. [En línea]. Disponible en: https://books.google.com.ar/books?id=tSTMDwAAQBAJ&pg=PR6&dq=sentiment+analysis&hl=es&sa=X&ved=2ahUKEwivhdf7wpLuAhWJHLkGHTpqDZgQ6AEwAXoECAQQAg#v=onepage&q=sentiment%20analysis&f=false

[17] E. Kumar, *Natural Language Processing*, Ene. 2011. Último acceso: Feb. 02, 2021. [En línea]. Disponible en: https://books.google.com.ar/books?hl=es&lr=&id=FpUBFNFuKWgC&oi=fnd&pg=PP2&dq=Natural+Language+Processing&ots=GFu40FgEQs&sig=VlWjw6ztgWzFFDPRzTR2rto0Kwk#v=onepage&q=Natural%20Language%20Processing&f=false

[18] J. D. Kelleher, Deep Learning, Ago. 2019. Último acceso: Mar. 09, 2021. [En línea]. Disponible en: https://books.google.com.ar/books?hl=es&lr=&id=b06qDwAAQBAJ&oi=fnd&pg=PP9&dq=+deep+learning&ots=_oAXRQhXYO&sig=iA9NO9pNBS9NhzMHO9zpj0jp2jI#v=onepage&q=deep%20learning&f=false

[19] F. Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain*, Nov. 1958. Último acceso: Mar. 10, 2021. [En línea]. Disponible en: https://pdfs.semanticscholar.org/5d11/aad09f65431b5d3cb1d85328743c9e53ba96.pdf

---

[20] P. J. Werbos, *Backpropagation through time: what it does and how to do it*, Proceedings of IECON '93 - 19th Annual Conference of IEEE Industrial Electronics, Oct. 1993. Último acceso: Mar. 10, 2021. [En línea]. Disponible en: https://ieeexplore.ieee.org/document/58337

[21] R. Ganda, A. Mahmood, *Deep Learning approach for sentiment analysis of short texts*, 2017 3rd International Conference on Control, Automation and Robotics (ICCAR), Abri. 2017. Último acceso: Mar. 11, 2020. [En línea]. Disponible en: https://www.researchgate.net/publication/317701706_Deep_Learning_approach_for_sentiment_analysis_of_short_texts/figures?lo=1

[22] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng and C. Potts, *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank*, Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, Oct. 2013. Último acceso: Mar. 13, 2021. [En línea]. Disponible en: https://www.aclweb.org/anthology/D13-1170.pdf

[23] M. M. Lopez, J. Kalita, *Deep Learning applied to NLP*, Mar. 2017. Último acceso: Mar. 14, 2020. [En línea]. Disponible en: https://arxiv.org/pdf/1703.03091.pdf

[24] Natural Language Toolkit. Último acceso: Mar. 18, 2020. [En línea]. Disponible en: https://www.nltk.org

[25] T. Mikolov, K. Chen, G. Corrado, & J. Dean, *Efficient estimation of word representations in vector space*, Oct. 2013. Último acceso: May. 24, 2021. [En línea]. Disponible en: https://arxiv.org/pdf/1310.4546.pdf

[26] Q. Le, T. Mikolov, *Distributed Representations of Sentences and Documents*, Jun. 2014. Último acceso: May. 24, 2021. [En línea]. Disponible en: https://arxiv.org/pdf/1405.4053.pdf

[27] Cornell, *Movie Review Data.* Último acceso: Jun. 14, 2021. [En línea]. Disponible en: http://www.cs.cornell.edu/people/pabo/movie-review-data/

[28] Textblob. Último acceso: Jun. 2, 2021. [En línea]. Disponible en: https://textblob.readthedocs.io/en/dev/quickstart.html

[29] CoreNLP. Último acceso: Jun. 28, 2021. [En línea]. Disponible en: https://stanfordnlp.github.io/CoreNLP/

[30] Stanza. Último acceso: Jul. 05, 2021. [En línea]. Disponible en: https://stanfordnlp.github.io/stanza/

---

[31] Stanford Sentiment Treebank. Último acceso: Jul. 05, 2020. [En línea]. Disponible en: https://github.com/stanfordnlp/sentiment-treebank

[32] MELD. Último acceso: Jul. 05, 2021. [En línea]. Disponible en: https://github.com/declare-lab/MELD/tree/master/data/MELD

[33] SLSD. Último acceso: Jul. 05, 2021. [En línea]. Disponible en: https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences

[34] Arguana. Último acceso: Jul. 05, 2021. [En línea]. Disponible en: http://argumentation.bplaced.net/arguana/data

[35] Airline Twitter Sentiment Dataset. Último acceso: Jul. 05, 2021. [En línea]. Disponible en: https://www.kaggle.com/crowdflower/twitter-airline-sentiment/data

[36] C.J. Hutto and E. Gilbert, *VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text*, Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, Jun. 2014. Último acceso: Jul. 05, 2021. [En línea]. Disponible en: https://github.com/cjhutto/vaderSentiment

[37] Flair, Humboldt University of Berlin. Último acceso: Ago. 28, 2021. [En línea]. Disponible en: https://github.com/flairNLP/flair

[38] Twitter Sentiment Data, Kaggle. Último acceso: Ago. 29, 2021. [En línea]. Disponible en: https://www.kaggle.com/seunowo/sentiment-analysis-twitter-dataset

[39] IMDB Dataset of 50K Movie Reviews, Kaggle. Último acceso: Ago. 30, 2021. [En línea]. Disponible en: https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

[40] J. Pennington, R. Socher, C. D. Manning, *GloVe: Global Vectors for Word Representation*. Último acceso: Ago. 30, 2021. [En línea]. Disponible en: https://nlp.stanford.edu/projects/glove/

[41] *Bidirectional layer*, Keras.Último acceso: Sep. 15, 2021. [En línea]. Disponible en: https://keras.io/api/layers/recurrent_layers/bidirectional/

[42] Y. Verma, *Complete Guide To Bidirectional LSTM,* Jul. 2021. Último acceso: Sept. 15, 2021. [En línea]. Disponible en: https://analyticsindiamag.com/complete-guide-to-bidirectional-lstm-with-python-codes/

[43] Go, A., Bhayani, R. and Huang, L.. Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford, 1(2009), 2009. Último acceso: Sept. 15, 2021. [En línea]. Disponible en: https://www.kaggle.com/kazanova/sentiment140?select=training.1600000.processed.noemoticon.csv

---

[44] Twitter Vestors, NLP Stanford (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download)Último acceso: Sept. 15, 2021. [En línea]. Disponible en: https://nlp.stanford.edu/data/glove.twitter.27B.zip

[45] Roesslein, J. *Tweepy: Twitter for Python.* Último acceso: Sept. 16, 2021. [En línea]. Disponible en: https://github.com/tweepy/tweepy

[46] Boe B, *PRAW: The Python Reddit API Wrapper,* 2012. Último acceso: Sept. 16, 2021. [En línea]. Disponible en: https://github.com/praw-dev/praw/

[47] V. Rossum, G. & D. F.L, *Python 3 Reference Manual, Scotts Valley*, CA: CreateSpace, 2009.Último acceso: Sept. 18, 2021. [En línea]. Disponible en: https://www.python.org

[48] W. McKinney & others, *Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference,* 2010. Último acceso: Sept. 18, 2021. [En línea]. Disponible en: https://pandas.pydata.org

[49] C.R. Harris, K.J. Millman, van der Walt, S.J, *Array programming with NumPy,* 2020. Último acceso: Sept. 18, 2021. [En línea]. Disponible en: https://numpy.org

[50] F. Chollet & others, *Keras*, 2015. Último acceso: Sept. 18, 2021. [En línea]. Disponible en: https://github.com/fchollet/keras

[51] Google, *Google Colab*, Último acceso: Sept. 18, 2021. [En línea]. Disponible en: https://colab.research.google.com

[52] T. Kluyver, *Jupyter Notebooks – a publishing format for reproducible computational workflows*, In F. Loizides & B. Schmidt, eds. Positioning and Power in Academic Publishing: Players, Agents and Agendas,2016, Último acceso: Sept. 18, 2021. [En línea]. Disponible en: https://jupyter.org

[53] C. Dang, N. M. Moreno García, F. De La Prieta, *Sentiment Analysis Based on Deep Learning: A Comparative Study,* Jun 2020. Último acceso: Sept. 18, 2021. [En línea]. Disponible en: https://www.researchgate.net/figure/Differences-between-two-classification-approaches-of-sentiment-polarity-machine-learning_fig1_341998176

Glosario

Natural language processing, NLP: en español, procesamiento del lenguaje natural (NLP, por sus siglas en inglés) es una rama de la inteligencia artificial que ayuda a las computadoras a entender, interpretar y manipular el lenguaje humano. NLP toma elementos prestados de muchas disciplinas, incluyendo la ciencia de la computación y la lingüística computacional, en su afán por cerrar la brecha entre la comunicación humana y el entendimiento de las computadoras.

Naive Bayes, NB: El algoritmo Naïve Bayes es un algoritmo de aprendizaje supervisado, que se basa en el teorema de Bayes y se utiliza para resolver problemas de clasificación. En teoría de la probabilidad y minería de datos, un clasificador Naive Bayes es un clasificador probabilístico fundamentado en el teorema de Bayes y algunas hipótesis simplificadoras adicionales.

Maximum Entropy, ME: El clasificador Max Entropy es un clasificador probabilístico que pertenece a la clase de modelos exponenciales. El MaxEnt se basa en el principio de máxima entropía y, de todos los modelos que se ajustan a los datos de entrenamiento, selecciona el que tiene la mayor entropía.

Support Vector Machines, SVM: Support Vector Machine o SVM es uno de los algoritmos de aprendizaje supervisado, que se utiliza para problemas de clasificación y regresión. Sin embargo, principalmente, se usa para problemas de clasificación en Machine Learning.El objetivo del algoritmo SVM es crear la mejor línea o límite de decisión que pueda segregar el espacio n-dimensional en clases para colocar fácilmente el nuevo punto de datos en la categoría correcta en el futuro. Este límite de mejor decisión se denomina hiperplano.

Unigram Model: En los campos de la lingüística computacional y la probabilidad, un n-grama es una secuencia contigua de n elementos de una muestra dada de texto o habla. Los elementos pueden ser fonemas, sílabas, letras, palabras o pares de bases según la aplicación.

Features, características: En el aprendizaje automático y el reconocimiento de patrones, una característica es una propiedad individual medible o una característica de un fenómeno. La elección de características informativas, discriminatorias e independientes es un elemento crucial de los algoritmos efectivos en el reconocimiento, clasificación y regresión de patrones. Las características suelen ser numéricas, pero las características estructurales, como cadenas y gráficos, se utilizan en el reconocimiento de patrones sintácticos.

Part of Speech, POS: Se refiere al proceso de clasificar palabras en sus partes del discurso (también conocido como clases de palabras o categorías léxicas). Este es un enfoque de aprendizaje supervisado.

Named Entity Recognition, NER: El reconocimiento de entidades con nombre (NER), a veces denominado fragmentación, extracción o identificación de entidades, es la tarea de identificar y categorizar información clave (entidades) en el texto. Una entidad puede ser cualquier palabra o serie de palabras que se refieran consistentemente a la misma cosa. Cada entidad detectada se clasifica en una categoría predeterminada.

Term Frequency-Inverse Document Frequency, TF-IDF: TF-IDF (término frecuencia inversa de frecuencia de documentos) es una medida estadística que evalúa la relevancia de una palabra para un documento en una colección de documentos.

Word Embeddings: en español, incrustaciones de palabras, son un tipo de representación de palabras que permite que las palabras con un significado similar tengan una representación similar. Son una representación distribuida de texto que es quizás uno de los avances clave para el impresionante desempeño de los métodos de aprendizaje profundo en los desafiantes problemas de procesamiento del lenguaje natural. Cada palabra está representada por un vector de valor real, a menudo decenas o cientos de dimensiones. Esto contrasta con los miles o millones de dimensiones requeridas para representaciones de palabras dispersas.

Hiperparámetros: Los hiperparámetros son parámetros ajustables que permiten controlar el proceso de entrenamiento de un modelo. Por ejemplo, con redes neuronales, puede decidir el número de capas ocultas y el número de nodos de cada capa. El rendimiento de un modelo depende en gran medida de los hiperparámetros

Epoch: Una epoch, o “época” en español, es un término utilizado en el machine learning e indica el número de pasadas de todo el conjunto de datos de entrenamiento que ha completado el algoritmo de machine learning. Los conjuntos de datos generalmente se agrupan en lotes (especialmente cuando la cantidad de datos es muy grande).

Anexo

1. **Anexo 1: Caso de Uso - Solicitar análisis de sentimientos**

Overview	Solicitación de una análisis de sentimientos. La inicialización del prototipo
Actores	Usuario
Precondiciones	-
Postcondiciones	-

Flujo de eventos

Actor	Sistema
1. El usuario ejecuta el programa	
	2. Preguntar al usuario que API usar

2. **Anexo 2: Caso de Uso - Seleccionar API de twitter**

Overview	Usuario elige la API de twitter para obtener el texto para realizar el análisis de sentimientos
Actores	Usuario
Precondiciones	Haber ejecutado el programa
Postcondiciones	Solicitar al usuario la palabra clave

Flujo de eventos

Actor	Sistema

	1. Preguntar al usuario que API usar. “1” para Twitter, “2” para Reddit
2. Ingresa “1”	3. Solicita al usuario la palabra clave a buscar

3. Anexo 3: Caso de Uso - Seleccionar API de Reddit

Overview	Usuario elige la API de Reddit para obtener el texto para realizar el análisis de sentimientos
Actores	Usuario
Precondiciones	Haber ejecutado el programa
Postcondiciones	Ejecutar la api de reddit para obtener los comentarios del posteo

Flujo de eventos

Actor	Sistema
	1. Preguntar al usuario que API usar. “1” para Twitter, “2” para Reddit
2. Ingresa “2”	3. Ejecuta la función para recolectar comentarios de posteo

4. Anexo 4: Caso de Uso - Ingresar palabra clave

Overview	Usuario eligio la API de twitter y debe ingresar la palabra clave para buscar los tweets
Actores	Usuario
Precondiciones	Haber seleccionado la API de Twitter

Postcondiciones	Ejecutar la api de Twitter para obtener los tweets con la palabra clave

Flujo de eventos

Actor	Sistema
	1. Solicitar al usuario la palabra clave
2. Ingresa palabra clave	3. Ejecuta la función para recolectar los tweets con la palabra clave

5. Anexo 5: Caso de Uso - Recolectar tweets

Overview	El sistema recolecta los tweets que contienen la palabra clave y lo adjunta a un array
Actores	API Twitter
Precondiciones	Haber seleccionado la API de Twitter e ingresado la palabra clave
Postcondiciones	Obtener el array de tweets con la palabra clave

Flujo de eventos

Actor	Sistema
	1. Se busca en tweets con la palabra clave, limitando a 10. 2. Se obtiene el listado de tweets 3. Se agrega los tweets al array 4. Se retorna el array.

6. Anexo 6: Caso de Uso - Recolectar comentarios de posteo

Overview	El sistema recolecta los comentarios del posteo mas controversial del mes
Actores	API Reddit
Precondiciones	Haber seleccionado la API de Reddit
Postcondiciones	Obtener el array de comentarios del posteo

Flujo de eventos

Actor	Sistema
	1. Se busca en el subreddit de “politics” el posteo mas controversial del mes 2. Se obtiene el posteo 3. Se itera en los comentarios 4. Se filtra los 10 primeros 5. Retorna el array de comentarios.

7. Anexo 7: Caso de Uso - “Limpiar” texto

Overview	El sistema elimina el texto no deseado.
Actores	Sistema
Precondiciones	Ingresar de entrada el array de tweets o comentarios.
Postcondiciones	Obtener el array de comentarios o tweets filtrados.

Flujo de eventos

Actor	Sistema

	1. Se itera entre el array de comentarios o tweets 2. Se elimina las "@" y puntuaciones 3. Se elimina links 4. Se elimina "#" y la palabras que la siguen 5. Se elimina todo lo que no sea caracteres 6. Se elimina espacios extra 7. Se convierte todo a minúsculas

8. Anexo 8: Caso de Uso - Pre-procesar texto

Overview	El sistema transforma cada texto en una secuencia de enteros y se realiza el "padding"
Actores	Keras
Precondiciones	Ingresar de entrada el array de texto "limpio"
Postcondiciones	Obtener el array de índices correspondiente.

Flujo de eventos

Actor	Sistema
	1. Toma cada palabra en el texto y la reemplaza con su valor entero correspondiente al diccionario 2. Se realiza el padding, rellenando con 0 al principio de cada secuencia hasta que cada secuencia tenga la misma longitud que la secuencia más larga. 3. Se devuelve el array de índices.

9. Anexo 9: Caso de Uso - Clasificar texto

Overview	El modelo predice el sentimiento devolviendo el valor entre 0 a 1
Actores	Modelo LSTM bidireccional
Precondiciones	Ingresar de entrada el array de indices correspondientes
Postcondiciones	Obtener el array de sentimiento

Flujo de eventos

Actor	Sistema
	1. Con el modelo cargado en el prototipo, se predice el sentimiento agregando como input el array de índices del diccionario. 2. El modelo devuelve el array con cada valor entre 0 a 1..

10. Anexo 10: Caso de Uso - Mostrar resultados

Overview	Se agrega formato, se tabula y muestra el resultado
Actores	Sistema
Precondiciones	Contar con el array de texto a analizar, contar con el array con los sentimientos
Postcondiciones	Mostrar el resultado con un formato más legible al usuario.

Flujo de eventos

Actor	Sistema
	1. Se ingresa el array de texto a analizar y el del texto con el análisis de sentimientos

	2. Se cambia el valor a texto: mayor a 0.59 es positivo, entre 0.45 y 0.59 es neutral, y menor a 0.45 es negativo. 3. Se agrega los valores a un array para mostrar con formato de tabla.