Aplicación de técnicas de Machine Learning para hacer análisis de polaridad de sentimientos en texto para detectar tendencias de opinión en plataformas online

Juan David Granados Figueroa

UNIVERSIDAD SANTO TOMÁS
FACULTAD DE INGENIERÍA ELECTRÓNICA
BOGOTA D.C.
2020

Aplicación de técnicas de Machine Learning para hacer análisis de polaridad de sentimientos en texto para detectar tendencias de opinión en plataformas online

Juan David Granados Figueroa

Proyecto de grado presentado como requisito para optar al título de INGENIERO ELECTRÓNICO

Director: Sindy Paola Amaya. M.Sc.
Codirector: Armando Mateus Rojas. M.Sc.

UNIVERSIDAD SANTO TOMÁS DE AQUINO
FACULTAD DE INGENIERÍA ELECTRÓNICA
BOGOTÁ D.C.
2020

Dedicatoria

Le dedico este trabajo a mi madre Claudia Figueroa quien siempre me ha apoyado, y quien es un ejemplo a seguir y símbolo de admiración y tenacidad en que los sueños se pueden cumplir. A mi padre Francisco Granados, quien siempre me ha cuidado, y apoyado con cariño y amor.

AGRADECIMIENTOS

Agradezco a mi directora Sindy Amaya quien me guio durante todo el proceso de realización de este trabajo de grado, a mi Codirector Armando Mateus quien también me guio para su culminación. A todos mis profesores durante mi carrera de Ingeniería Electrónica que me brindaron sus conocimientos, a todas las personas con las que compartí, compañeros y amigos, de mi Universidad Santo Tomás.

Contenido

Figuras

Tablas

GLOSARIO

Accuracy (AC): Exactitud por su traducción al español, es la relación entre predicciones correctas y el número de ejemplos, en un modelo de *Machine Learning*.

API*:* Interfaz de programación de aplicaciones por la traducción de sus siglas al español, es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Corpus*:* Similar al *Dataset* cuando se aplica a tareas de Procesamiento Natural de Lenguaje.

Dataset*:* Es el conjunto de datos con el cual se entrena un modelo de *Deep Learning.*

Deep Learning**:** Aprendizaje Profundo, es un campo dentro del *Machine Learning* que está inspirado en cómo funcionan las neuronas en el cerebro humano, con múltiples capas, y muchas conexiones entre estas.

Dropout: Es una técnica para disminuir el sobreajuste en los sistemas de *Deep Learning*.

F-Measure: Es una medida armónica que relaciona la precisión y el recall, y es bastante utilizada para problemas de clasificación.

GRU (Gate Recurrent Unit)**:** Unidad Recurrente Cerrada, es otra clase de unidad utilizada en las *RNN*, que permite manejar dependencias a largo plazo.

Hashtag: Etiqueta en español, es una palabra precedida por el símbolo #, que en redes sociales suele hacer alusión a un tema en específico, para señalar tendencias, y temas sobre el que gira cierta conversación.

Hiperparámetros: Son los parámetros con los cuales se configura un modelo de *Deep Learning.*

Link: Hace referencia a dominios de páginas web

LSTM (Long-Short Term Memory Unit): Unidad de Memoria de Corto-Largo Plazo, es una unidad especializada utilizada en las *RNN*, que permite manejar dependencias de más largo plazo.

***Machine Learning*:** Aprendizaje de máquina, es un campo dentro de la Inteligencia Artificial, que busca crear sistemas que son capaces de aprender, como la regresión lineal, *Support Vector Machine* (*SVM)*, *Naive Bayes*.

Naive Bayes: 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

***NLP* (*Natural Language Processing*):** Que en español significa Procesamiento de Lenguaje Natural, es la tarea de entendimiento o abstracción del lenguaje del ser humano, por parte de una máquina.

***RNN (Recurrent Neuronal Networks)*:** Redes Neuronales Recurrentes, son una clase de Redes Neuronales dentro del *Deep Learning*, que está basada en una arquitectura que genera conexiones recurrentes entre distintos pasos de tiempo dentro de una secuencia.

***Stack* de tecnología:** Se refiere al conjunto de tecnologías, librerías, entorno de entrenamiento, lenguaje de programación utilizados.

Sobreajuste: Es cuando el modelo obtiene una exactitud alta con los datos de entrenamiento, pero su resultado es inferior con los datos de prueba.

SVM: *Support Vector Machine*, máquinas de soportes vectoriales por la traducción al español, son un conjunto de algoritmos de aprendizaje supervisado en el dominio del *Machine Learning* especializados en problemas de clasificación.

TASS: Taller de Análisis de sentimientos en la SEPLN, es un trabajo realizado por varias Universidades de España, en que se generó un *Dataset* de análisis de sentimientos en lenguaje español.

***Test*:** Es el proceso de prueba de un modelo de *Machine Learning.*

***Train*:** Es el proceso de entrenamiento de un modelo de *Machine Learning*.

Tuit: Es una publicación en Twitter, de 140 caracteres máximo.

Username: En Twitter, son los nombres de usuario

Word2Vec: Es un arreglo de información, que contiene numéricamente un vocabulario de palabras, para que los modelos de *Machine Learning* puedan procesar estos datos.

XML (Extensible Markup Language): Lenguaje de marcado extensible por la traducción de sus siglas al español, es un metalenguaje el cual es utilizado para almacenar datos en forma legible.

RESUMEN

Internet ha permitido a millones de personas conectarse y generar interacciones, como en las redes sociales, lo que ha generado mucha información no estructurada, que es difícil de analizar por un grupo de seres humanos, debido a su gran cantidad. En este trabajo se aplican técnicas de *Machine Learning* para analizar la polaridad de sentimiento en lenguaje español, de los comentarios de usuarios de Twitter acerca de diversos temas. El análisis de polaridad de sentimiento permite analizar las tendencias de opinión de una forma rápida y automática, permitiendo a las empresas y organización tener información valiosa para la toma de decisiones. Se implementan Redes Neuronales Recurrentes, las cuales son uno de los métodos que mejores resultados muestran para el análisis de secuencias, mediante la aplicación de *Deep Learning*, el cual pertenece al campo del *Machine Learning* y que, además, evita la necesidad de realizar extracción de características, lo cual conllevaría una minuciosa selección por parte de expertos de lenguaje. Se utiliza Keras para programar el modelo con tensorflow, y se obtienen resultados de exactitud muy cercanos a los sistemas más avanzados en el estado del arte. El modelo es entrenado con un *Dataset* de 49.444 oraciones etiquetadas con positivo o negativo, en base al corpus de TASS.

INTRODUCCIÓN

El análisis de sentimientos es una poderosa técnica que permite a las empresas y organizaciones tener datos relevantes acerca de la opinión de usurarios, de forma rápida y automática, análisis que pueden ser utilizados para la toma de decisiones [1]. Muchos investigadores se han interesado en el tema desde el surgimiento del *Machine Learning*, con la implementación de métodos como *Naive Bayes*, y *SVM*, los cuales requerían realizar una minuciosa selección de características, como frecuencia de aparición de palabras dentro de un texto, entre otras [2]. Sin embargo, con el resurgimiento del *Deep Learning*, se facilitó la tarea de extracción de características, sin la necesidad de recurrir a un experto que las seleccione manualmente, esto permitió un salto muy importante en el estado del arte del *Machine Learning*, permitiendo alcanzar *records* en diversas áreas de aplicación, como en el campo del Procesamiento del Lenguaje Natural [3]. El Procesamiento del Lenguaje Natural [4], es un campo especializado en procesar y abstraer las palabras de los seres humanos en texto, para realizar diversas tareas, como análisis de polaridad de sentimiento. Las oraciones son secuencias de palabras, y el significado de una palabra no depende solo de su propio significado, sino del sentido que le dan las demás palabras dentro de la oración, es decir del contexto. Las Redes Neuronales Recurrentes, *RNN* por sus siglas en inglés [4], son una técnica de *Deep Learning* que se especializa en procesar secuencias, y son capaces de actualizar el estado de las neuronas según los estados anteriores, manteniendo dependencia entre las diferentes palabras de una oración.

A pesar de que las *RNN* teóricamente son muy buenas para manejar secuencias, no pueden mantener dependencias de largo plazo debido a limitaciones técnicas. Su arquitectura está deseñada de tal modo que presentan altísimas cantidades de conexiones entre sus neuronas, y suelen presentar problemas de desvanecimiento de gradiente, es decir que su gradiente tiende a cero y deja de actualizarse, lo que conlleva a un punto en el cual no puede aprender más [4]. Los investigadores propusieron unidades especializadas dentro de las *RNN* [5], como lo son las unidades de memoria a corto-largo plazo, *LSTM* por sus siglas en inglés, y más recientemente las unidades de compuerta recurrente, *GRU* por sus siglas en inglés, las cuales muestran resultados superiores a las *RNN* simples, y permiten que estas “recuerden” dependencias a más largo plazo. Además, aunque las *LSTM* y *GRU* muestran resultados similares, las *GRU* son más simple en su arquitectura.

En español son pocos los trabajos realizados en el campo de análisis de sentimientos en comparación con los trabajos realizados en inglés, donde hay muchos *Datasets* e investigaciones en el área. Uno de los pocos corpus en español de análisis de polaridad de sentimiento es TASS (Taller de Análisis de Sentimientos en la SEPLN) [6]. El foco del presente proyecto de grado está en implementar un algoritmo basado en *Machine Learning*, para detectar tendencias de opinión en plataformas online. Se entrena un sistema mediante los comentarios de usuarios de

Twitter extraídos de *TASS* acerca de diversos temas en lenguaje español, obteniendo resultados que se acercan al estado del arte.

Se implementa un modelo de *Deep Learning* con *RNN* y unidades *GRU*, utilizando 49.444 oraciones etiquetadas con positivo o negativo, que se logran extraer de *corpus* generados en diferentes países de habla hispana, gracias al trabajo de TASS. Se utiliza *Google Colaboratory* como entorno de entrenamiento, python como lenguaje de desarrollo, *tensorflow* y *keras*. El trabajo está distribuido de la siguiente manera: En el capítulo 2 se enuncian los trabajos de *Machine Learning* y *Deep Learning* en cuanto a análisis de polaridad de sentimientos, en el 3 está la justificación, en el 4 el objetivo general y específicos. En el capítulo 5 de marco teórico se profundiza acerca de las bases del *Machine Learning*, con especial énfasis en *Deep Learning*, una explicación acerca de las *RNN, LSTM, y GRU*, también se enuncian las métricas de medición del rendimiento del modelo, se expone el corpus de TASS, finalmente se expone las herramientas de desarrollo como *tensorflow*, *Keras* y *Google Colaboratory*. En el capítulo 6 se describe el Diseño Metodológico. En el 7 se enuncian los resultados. En el 8, se dan a conocer las conclusiones. Y finalmente se encuentra la Bibliografía.

1. PROBLEMA

Con el avance de la tecnología *web* y su crecimiento, se ha generado una gran cantidad de datos. Internet se ha convertido en una plataforma de aprendizaje online, intercambio de ideas y opiniones. Las redes sociales como Twitter, Facebook entre otras, han ganado mucha popularidad ya que permiten que las personas compartan y expresen sus opiniones sobre diferentes temas [1]. Además, existen plataformas como *TripAdvisor* y *Google Maps* que dan recomendaciones de locales comerciales.

Toda esta información representa una gran oportunidad para empresas, organizaciones gubernamentales entre otras, para ver cuáles son las opiniones de su público objetivo, para detectar las tendencias positivas y negativas, es decir la polaridad de opinión, pero la cantidad de información generada por los usuarios es demasiado grande para ser analizada por un grupo de personas, además, es información poco estructurada [2]. Las encuestas, talleres y entrevistas que se utilizan para recabar opiniones públicas consumen mucho tiempo, son costosas, demoradas y limitadas, con posibles sesgos, actualizaciones bajas, así como poca participación [7].

Las técnicas convencionales de procesamiento de texto se basan en análisis objetivos como resúmenes de texto, las cuales no pueden hacer análisis subjetivos para conocer las opiniones, valoraciones, actitudes, y emociones de los usuarios en su polaridad, es decir en la tendencia positiva y negativa de los usuarios [1], [2]. Además, mucha de la información publicada por usuarios en plataformas online contiene ruido, caracteres innecesarios como *hashtags*, etiquetas, emoticones, entre otros, lo que hace difícil procesar esta información de manera adecuada [8]. Además, las técnicas de *Machine Learning* generalmente necesitan una gran cantidad de información para realizar el entrenamiento de los algoritmos, y ajustar los datos para que los modelos sean precisos. Otro de los problemas es que generalmente estos modelos se entrenan en temas en específico (dominio) y no sirven para un dominio diferente para el cual fue entrenado [9], [10]. También en lenguaje español a comparación de inglés, son pocas las fuentes de información y herramientas para realizar tareas de procesamiento de lenguaje natural y análisis de sentimientos [6]. Dificultando la tarea de los investigadores en esta área, y el

aprovechamiento de estas tecnologías por parte de las organizaciones hispanohablantes.

Por eso se plantea en este proyecto de grado desarrollar un modelo de *Machine Learning* que sea capaz de hacer análisis de polaridad de sentimientos (positivos o negativos) en texto en lenguaje español para detectar tendencias de opinión en plataformas online.

Pregunta de Investigación

¿Cómo identificar la polaridad de sentimientos en lenguaje español en texto de manera automatizada para detectar tendencias de opinión en plataformas online mediante técnicas de *Machine Learning*?

2. ANTECEDENTES

En esta sección de antecedentes, se enuncian varios trabajos realizados en el campo de análisis de sentimientos. En los primeros párrafos se enuncian trabajos en los que se utilizan técnicas de *Machine Learning* como SVM y Naive Bayes, entre otras. Luego se habla de trabajos más actuales realizados con técnicas de *Deep Learning,* los cuales no requieren realizar un proceso de selección de características de forma manual, y alcanzan exactitudes similares o incluso superiores a los métodos basados en Machine Learning con *SVM* o *Naive Bayes*. Finalmente se enuncian algunos trabajos de *Deep Learning* que realizan otras tareas dentro del área de Análisis de Sentimientos.

En [1] hacen un resumen de los trabajos realizados de análisis de polaridad de sentimientos (positivo, negativo, neutro). También dicen cuáles son las fases requeridas para llevar a cabo el análisis de sentimientos de *Twitter*, como lo son: preprocesamiento, donde se elimina el ruido del texto como caracteres innecesarios entre otros, lo que ayuda a mejorar la exactitud en las siguientes fases. Extracción de características: donde se extraen las características del sistema, que luego serán utilizadas como vectores en los procesos de *Machine Learning*, entrenamiento: se necesita entrenar el modelo para que pueda llevar a cabo su tarea de predicción, clasificación: mediante métodos como *Naive Bayes*, Máxima entropía y *Support Vector Machine* (*SVM*). En [2] se catalogan varios trabajos acerca de análisis de sentimientos. Se discute acerca de los retos a resolver que requiere el análisis de sentimientos en la actualidad como lo son: reconocer el sarcasmo, palabras gramaticalmente incorrectas, manejar el dinamismo y ruido de las redes sociales entre otros. También encontraron que la mayoría de los últimos trabajos en análisis de sentimientos, están basados en técnicas de *Machine Learning* en vez de técnicas basadas en métodos léxicos. En [9] utilizan las *APIs* (*Application Programming Interface* por sus siglas en inglés) abiertas de Twitter para hacer análisis de sentimientos a tuits, que consultan según *Hashtags* públicos. Utilizan las librerías de *Stanford core NLP* en Java, las cuales proveen herramientas para realizar análisis de *NLP*, y análisis de sentimientos (en base a métodos de palabras clave) que asigna una calificación que va de 1 a 5 denotando que tan positivo o negativo es un fragmento de texto. Lo que hace su sistema es consumir la *API* de *Twitter*, con 25 tuits según un tema en específico mediante un Hashtag (puede buscar acerca de cualquier tema mediante Hashtags). Twitter utiliza un standard llamado *OAuth* para dar autorización de acceso a su *API*, el cual brinda seguridad al no revelar la contraseña de los usuarios. Una *API* es una interfaz que permite comunicar bases de datos o diferentes servicios web entre sí. En [11] utilizan análisis de sentimientos para clasificación de reseñas de negocios, utilizando un

conjunto grande de datos proporcionados por *Calc* (plataforma de valoración de locales comerciales a través de reseñas de usuarios) en el cual dicen que tan positivo o negativo son las opiniones de los usuarios en locales comerciales con respecto a la calificación de estrellas de 1 a 5 que dan los usuarios. Proponen varios enfoques, usando dos métodos de extracción de características, y cuatro modelos de *Machine Learning*. Nombran que hay varios obstáculos en análisis de sentimientos como lo son el sentimiento de las palabras dependiendo del contexto, lenguaje usado, ambigüedad del sentimiento cuando se usan palabras que no utilizan un sentimiento en particular o cuando se usa el sarcasmo. Ellos utilizan métodos de *Machine Learning* como lo son *Naive Bayes* y *SVM* (*support vector machine*) utilizando la librería de python scikit-learn [12] con 1,6 millones de reseñas de Yelp (*Dataset*) dando una exactitud de más del 90%, y presentan ideas de cómo mejorar su trabajo. Ellos utilizaron el 80% de los datos para entrenamiento, y un 20% para pruebas, y finalmente concluyen que utilizando el primer método de extracción de características con un método de *SVM* obtuvieron los resultados de mayor exactitud.

En [10] realizan análisis de polaridad de sentimientos para analizar la opinión de clientes de hoteles. Primero hacen obtención y etiquetado de datos. Hicieron encuestas a clientes sobre los servicios de hoteles mediante un portal web, luego un grupo de personas etiqueto los datos, los cuales servirán para entrenamiento y pruebas del modelo, luego realizan el preprocesamiento para dejarlo en un formato adecuado, luego de esto hacen la extracción de características convirtiendo el texto etiquetado en vectores de características mediante el uso de filtros. El siguiente paso es la selección de un algoritmo de clasificación apropiado, finalmente se entrena el modelo de Machine Learning y se prueban los resultados obtenidos. En [8] hacen un resumen del estado del arte actual del análisis de sentimientos donde clasifican los trabajos según los métodos utilizados, relatan algunos trabajos en diferentes lenguajes, **no se evidencia ninguno en español**. Dan a conocer las métricas más utilizadas para medir el rendimiento en el análisis de sentimientos, entre otros. Enuncian los enfoques para la detección de emociones en texto, los cuales son cuatro:

1. Enfoque basado en palabras clave: el cual depende de la existencia de una palabra clave en una oración, se utiliza un diccionario de palabras con sus emociones respectivas. Es fácil y rápido de implementar, pero tiene muchos inconvenientes y fallas cuando las emociones se expresan con palabras interconectadas, o cuando la oración no tiene una palabra explícita que denote la emoción, entre otras.

2. Enfoque basado en léxico/corpus: es una extensión del anterior, pero se asignan probabilidades o pesos a las palabras.
3. Enfoque basado en Machine Learning: es de los más utilizados y que mejores resultados presenta y además no requiere la mención de una emoción directa en la oración. Este enfoque se divide en aprendizaje Supervisado, No supervisado e hibrido.

Finalmente, en [8] enuncian los métodos de aprendizaje para los modelos de *Machine Learning*:

Supervisado: se basa en un conjunto de datos previamente etiquetado, donde parte de ellos servirá para entrenamiento y otra parte para probar la exactitud del modelo.

No Supervisado: se basa en un conjunto de datos no etiquetado, su problema es que necesita muchísima información para que el proceso de entrenamiento sea exacto.

Híbrido: Debido a que los resultados de los enfoques independientes utilizando *Machine Learning* no son lo suficientemente satisfactorios, se utilizan en conjunto para mejorar los resultados. Como tal los métodos de palabras clave y métodos léxicos, son rápidos y fáciles de implementar, pero presentan algunos inconvenientes, los métodos basados en *Machine Learning* son más exactos y más utilizados. Algunos conjuntos de datos famosos en ingles son *ISEAR (International Survey on Emotion Antecedents and Reactions)* el cual es uno de los más confiables, consiste de 7.666 oraciones etiquetadas con emociones como felicidad, miedo, enfado, vergüenza, y culpa. *Fairy Tales*: el cual contiene 15.000 oraciones etiquetadas con ira, disgusto, miedo, felicidad, tristeza, positivamente sorprendido, negativamente sorprendido o neutral, y finalmente *SemEva12007* el cual consiste en titulares de famosos periódicos estadounidenses como *The New York Times, CNN, y BBC News*. Además del buscador de noticias de *Google*.

En [13] al igual que en la mayoría de artículos de resúmenes del estado del arte, se nombra que *SVM* junto a *Naive Bayes*, son unos de los algoritmos más utilizados. Además, en [11] concluyen que mediante un buen método de extracción de características y *SVM* obtuvieron los mejores resultados.

En [3] comentan que en 2006 un campo del *Machine Learning* basado en cómo funciona las neuronas en el cerebro humano (*Deep Learning*) fue revivido, por un grupo de investigadores del *CIFAR (the Canadian Institute for Advanced Research*), ya que anteriormente se consideraba inviable debido entre varias razones a la capacidad de cómputo con la que no se contaba, necesaria para entrenar estos modelos. Ellos utilizaron aprendizaje no supervisado para realizar el proceso de extracción de características. La primera gran aplicación de este enfoque de preentrenamiento, fue en el campo del reconocimiento de voz, que fue posible por el avance de las unidades graficas de procesamiento (*GPUs*), luego en 2009 se batieron *records* utilizando este enfoque de preentrenamiento en el área del reconocimiento de voz, utilizando pocos datos. En 2012, versiones de la red profunda desarrollada en 2009 fueron desarrolladas y fueron desplegadas en dispositivos móviles. Una vez el *Deep Learning* fue vuelto a utilizar los investigadores se dieron cuenta que el proceso de preentrenamiento solo era necesario para *Datasets* pequeños. Uno de los impulsos más grandes al uso del *Deep Learning*, fue en la competencia de *ImageNet* en 2012, cuando *CNN* (Redes Neuronales Convolucionales por sus siglas en inglés) fueron aplicadas para clasificar más de 100 clases, de un *Dataset* de 1 millón de ejemplos, logrando resultados espectaculares, reduciendo el error a la mitad que los mejores enfoques a la época.

En [14] utilizan *Deep Learning* con *RNN* con *LSTM* para realizar análisis de sentimiento en Lenguaje Ingles, con clasificación de positivo, negativo y neutro. Utilizan el Dataset de IMBD [15]. También enuncian varios métodos de *Machine Learning* que se han nombrado anteriormente y alcanzan una Exactitud del 87,42%.

En [16] utilizan un enfoque de *Deep Learning*, utilizando *CNN y word2vec*, para realizar análisis de polaridad de sentimientos en Lenguaje Español, utilizan 100.000 tuits, y obtienen un F-measure de 88,7%. Primero realizan un pre procesamiento, luego el embedding o word2vec y finalmente implementan las *CNN*. Ellos nombran que a pesar que hay muchos *Datasets* en inglés, es difícil encontrar *Datasets* en análisis de sentimiento en español, por eso ellos decidieron utilizar la *API* de Twitter para crear el corpus.

En [17] realizan análisis de sentimiento, comentan que alcanzaron el estado del arte en el subconjunto binario del Stanford Sentiment Treetabank, y que utilizando un método llamado “capacity celling”, alcanzan un Exactitud del 95,22% en *Datasets*

grandes en cuanto al número de palabras por texto, también es un trabajo realizado en Lenguaje Ingles.

En [3] además, nombran que para tareas de Procesamiento de Lenguaje Natural, con *Deep Learning*, fue viable realizar el proceso de extracción de características para representar las palabras en forma de vectores asociados por la cercanía de significado entre ellas, es decir que al convertir las palabras a vectores, la red neuronal profunda, aprenderá a relacionar el significado de las palabras, la representación vectorial de las palabras será más cercana entre las palabras que se parezcan más. Los métodos que se utilizaban antes, como se nombra en los primeros párrafos de este capítulo, estaban basados en métodos de *Machine Learning* que requerían una minuciosa selección de características, maneja a cada palabra como una unidad atómica, así que estos no pueden generalizar sobre secuencia de palabras semánticamente relacionadas. Donde para tareas de *NLP*, las *RNN*, han mostrado grandes resultados, ya que son capaces de recordar estados pasados y manejar información en forma de secuencias como lo son las oraciones.

Luego de empezar a utilizar *Deep Learning* debido a que volvió más sencillo realizar esta tarea gracias a evitar la selección de características de forma manual, los nuevos avances en el campo del procesamiento de Lenguaje Natural, se dieron en tareas más complejas como análisis de sentimientos de aspectos dentro de textos, como por ejemplo en [18] donde utilizaron *Deep Learning*, con dos arquitecturas donde eligen algunos aspectos dentro de las revisiones de usuarios en un sitio web de aparatos de tecnología, entonces el modelo les permite detectar el sentimiento por ejemplo de la cámara, la batería y otras características de un celular. Obteniendo una Exactitud del 99%. Aun así, es un poco limitado porque tienen que seleccionar los aspectos específicos. También en [19] hacen análisis de polaridad de sentimiento de aspectos dentro de textos, utilizan una combinación de *RNN* con *LSTM* y *CNN*, comentan que obtuvieron mejores resultados que utilizando por ejemplo sistemas basados en *RNN* con sistemas basados en atención.

En 2017 Google propone una nueva arquitectura de redes neuronales llamada Transformers [20], la cual establecido un nuevo estado del arte. Está basado en un mecanismo llamado atención, el cual a diferencia de las *RNN* maneja las secuencias con índices que indican la posición de las palabras, por ejemplo, aumentando la capacidad de paralelización. Los Transformers, son modelos no supervisados de aprendizaje de representación, donde primero entrenan la red neuronal en corpus

de gran escala con data no etiquetada, y luego ajustan los modelos con tareas específicas.

Por ejemplo, en 2019 de Google, proponen BERT [21] el cual es un modelo basado en Transformers, que puede ser ajustado agregando una capa de salida para crear modelos que alcanzan rendimientos en algunos casos superiores al estado del arte en diferentes tareas de Procesamiento de Lenguaje Natural como respuesta a preguntas, inferencia de lenguaje, entre otras. En junio de 2020, crean XLNet [22] el cual es un modelo basado en Transformers, que obtiene mejores resultados que BERT en 20 tareas de procesamiento de lenguaje natural, como respuesta a preguntas, interferencia de lenguaje natural, análisis de sentimientos, clasificación de documentos, entre otras. Obtienen un error de 3,2 utilizando el corpus de análisis de sentimiento de IMBD, muestran, además, los resultados de error de otros modelos de forma comparativa como BERT con 4,51 de error.

En 2020 OpenAI, crea GPT-3 [23] el cual es un modelo autorregresivo de lenguaje entrenado con 175 mil millones de parámetros, que también se basa en Transformers. Estos modelos son pre entrenados en corpus grandes, y luego ajustados a tareas específicas, como traducción entre idiomas, respuestas a preguntas y en específico GPT-3 tiene la habilidad de realizar sumas de 3 dígitos, usar nuevas palabras en una oración. Finalmente, puede generar nuevos artículos escritos que evaluadores humanos tienen la dificultad para distinguir de uno escrito humanos. También requieren de miles a cientos de miles de ejemplos.

Estos ultimo trabajos, como se observa se enfocan en diversas tareas de Procesamiento de Lenguaje Natural, entre ellos pueden ser utilizados para análisis de sentimiento, aunque no es su foco central. Sin embargo, a comparación de modelos y tecnologías construidos en Lenguaje Ingles, en español es muy poco el avance que se ha realizado, hay pocos *Datasets* y por ejemplo los modelos basados en Transformers no se encuentran en español.

3. JUSTIFICACIÓN

Millones de personas expresan sus opiniones y percepciones acerca de diferentes temas a través de redes sociales, blogs o sitios de reseñas populares, esto genera una gran cantidad de información diversa y actualizada sobre cada aspecto de su vida cotidiana [24], [9], [11], [2]. Analizar toda esta información representa una gran oportunidad para diversos sectores de negocio o con propósitos sociales, para detectar tendencias de opinión, así influir, y mejorar en la toma de decisiones [9], [25], [2], [26].

El Análisis de Polaridad de Sentimientos es un método que permite encontrar de manera automática las tendencias de opinión de las personas en texto. Es una rama del *NLP (*Procesamiento de Lenguaje Natural, por sus siglas en Ingles*)* que clasifica las emociones detrás de un texto en positivo o negativo, y ha ganado mucha popularidad debido a su gran utilidad. Dando lugar a aplicaciones que permiten ahorrar dinero y tiempo a organizaciones, y les permite tomar mejores decisiones estratégicas para ser competitivas [9], [1], [11], [2], [8].

Sin embargo, debido al poco desarrollo en tareas de Procesamiento de Lenguaje Natural en lenguaje español [6], en comparación de inglés, se pierden muchas oportunidades de aprovechar al máximo estas tecnologías, lo que disminuye la competitividad de las distintas organizaciones y empresas hispanohablantes, en un mundo altamente globalizado.

En la venta de productos o servicios, lo que el público en general piensa acerca de estos, puede influir en que tan bien se venden, entender las opiniones y sentimientos expresados en las respectivas investigaciones de usuario y producto tiene mucha importancia, porque colectivamente estas opiniones reflejan la "sabiduría de las multitudes" (lo que piensa el público en general) y puede ser un muy buen indicador del rendimiento de las ventas futuras [27].

4. OBJETIVOS

4.1 OBJETIVO GENERAL

Identificar la polaridad de sentimientos en texto de manera automatizada para detectar tendencias de opinión en plataformas online mediante técnicas de *Machine Learning*.

4.2 OBJETIVOS ESPECÍFICOS

1. Determinar la fuente de información de opiniones de usuarios y el dominio, que se van a utilizar para extraer los datos.
2. Definir el indicador de sentimiento para realizar el entrenamiento y la métrica de validación del modelo.
3. Elegir el método de *Machine Learning* que se va a usar para realizar el análisis de sentimientos.
4. Realizar el preprocesamiento de la información, eliminando caracteres innecesarios para extraer las características más importantes que se van a usar como entrada para el modelo de entrenamiento.
5. Implementar y entrenar el modelo que se va a utilizar para el análisis de sentimientos.
6. Validar el modelo para verificar su exactitud a partir de la métrica seleccionada.

5. MARCO TEÓRICO

En el marco teórico, se habla de las bases del *Machine Learning* y *Deep Learning*, la función de activación *sigmoid*, el error, y métodos de optimización, métricas de rendimiento, de *Dropout* la cual es una técnica para disminuir el sobreajuste. Además, se explica las *RNN*. Unidades *LSTM*, y con especial énfasis en *GRU*. Se enuncia el corpus de *TASS* y herramientas de desarrollo, como *tensorflow*, *keras*, y *Google colaboratory*.

5.1 Machine Learning

5.1.1 Regresión Lineal

El *Machine Learning* aprende con ejemplos, se ingresa al sistema un conjunto de datos etiquetados. En [28] comentan que los investigadores crearon modelos que se asemejan a cómo funcionan las neuronas. Matemáticamente una neurona puede ser modelada mediante la siguiente ecuación:

$$z = \sum_{i=0}^{n} Wi * Xi \qquad (1)$$

Para la ecuación 1, **z** es la sumatoria de **Wi** * **Xi**, pesos y entradas respectivamente, del ejemplo o iteración **i**.

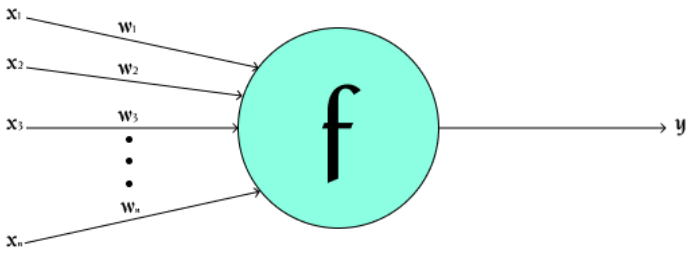

Figura 1. Modelo de Neurona Artificial. Fuente: propia.

La función de activación f como se ve en la figura 1, produce la salida $y = f(z)$ el cuál es el resultado de la neurona. Una neurona puede realizar tareas sencillas, pero para problemas más complicados es necesario usar redes de neuronas.

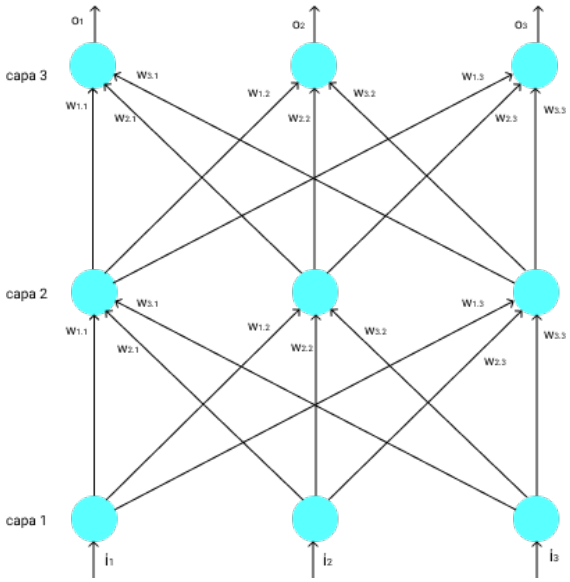

Figura 2. Red Neuronal de adelanto. Fuente: Propia.

En una red neuronal como la figura 2 [28], cada neurona se conecta con las de la siguiente capa. Entre cada conexión hay pesos, los que ayudan a fortalecer o debilitar la participación de cada neurona en el sistema. Una red neuronal puede ser expresada como una serie de operaciones entre vectores.

$$z = W^T x + b \tag{2}$$

$$Y = f(z) \tag{3}$$

Para la ecuación 2, ***x*** es un vector de entradas de longitud *n*, ***W*** es el vector de pesos de longitud *nxm*, ***b*** es un vector *bias* (ajusta posibles sesgos) de longitud *m*, ***z*** es un vector de longitud m y para la ecuación 3, ***f(z)*** es la función de activación, que nos da el vector de salida ***Y***, es decir la hipótesis o predicción.

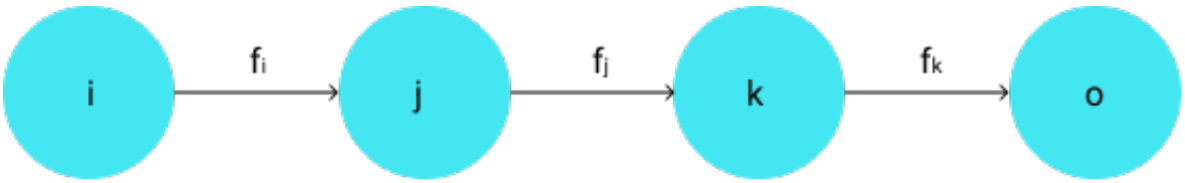

Figura 3. Red Neuronal de una neurona por capa, expresada en grafos. Fuente: propia

Si se simplifica el sistema viendo cada neurona como un nodo, figura 3 [4], se observa que el valor resultante de cada neurona es la sumatoria de las neuronas anteriores por sus pesos, y este patrón se repite a través de todas las capas.

Pero se sabe que un sistema clásico de *Machine Learning* tiene limitantes ya que no puede clasificar muy bien cuando los datos tienen muchas variaciones. Los clasificadores lineales o cualquier otro clasificador básico, para su entrenamiento requieren la selección de las características más importantes de los datos**,** este proceso requiere una considerable cantidad de habilidades de ingeniería y experticia en el campo de aplicación del sistema porque se hace de forma manual [3], [28]. Afortunadamente, la extracción de características puede ser evitada mediante el ***Deep Learning***, ya que la primera capa se encarga de realizar este proceso de manera automática [3].

5.1.2 **Función de Activación *Sigmoid***

La función de activación *Sigmoid* [28] no es lineal, y está definida por la ecuación 4:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{4}$$

Cuando la salida es muy pequeña la función *Sigmoid* da como resultado un valor cercano a cero, y cuando la salida es muy grande da un resultado muy cercano a 1.

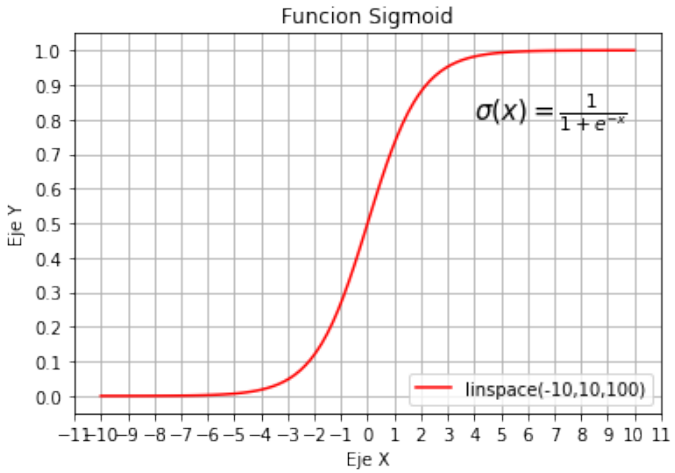

Figura 4. Gráfica de la función Sigmoid, los valores mayores que cero se aproximaran a 1, los menores, se aproximaran a 0. Fuente: propia.

La función Sigmoid [28], [4] (figura 4), permite clasificar en problemas binarios, como en el análisis de polaridad de sentimientos, en donde por ejemplo el sentimiento positivo está definido por los valores cercanos a 1 y el sentimiento negativo está definido por los valores cercanos a 0.

5.2 Deep Learning

El *Deep Learning*, es parte de una familia más grande de métodos de *Machine Learning* (figura 5), basado en Redes Neuronales Artificiales. El aprendizaje puede ser supervisado, semi-supervisado o no supervisado [29], [30], [4].

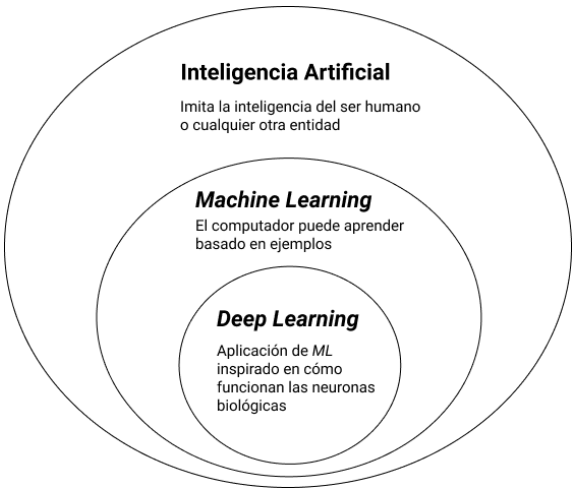

Figura 5. Dominio de la Inteligencia Artificial, Machine Learning y Deep Learning. Fuente: propia

5.2.1 Aprendizaje Supervisado

La mayoría de algoritmos de *Machine Learning* y *Deep Learning*, utilizan aprendizaje supervisado [3], el cual se basa en darle ejemplos al sistema para enseñarle a predecir el comportamiento o resultado según unas etiquetas a esos ejemplos. El sistema aprende a mapear cada entrada con cada salida, ajustando sus parámetros internos de forma iterativa hasta minimizar el error, como se ha explicado en los capítulos anteriores. Solo se enuncia esta clase de aprendizaje debido a que es el que se utiliza en el presente trabajo.

5.2.2 Retos de minimización del error

En un sistema de *Deep Learning*, hay cientos de millones de pesos ajustables, y cientos de millones de ejemplos etiquetados para entrenar la máquina. La función objetivo se presenta como un paisaje multidimensional, el cual tiene muchas curvas [3].

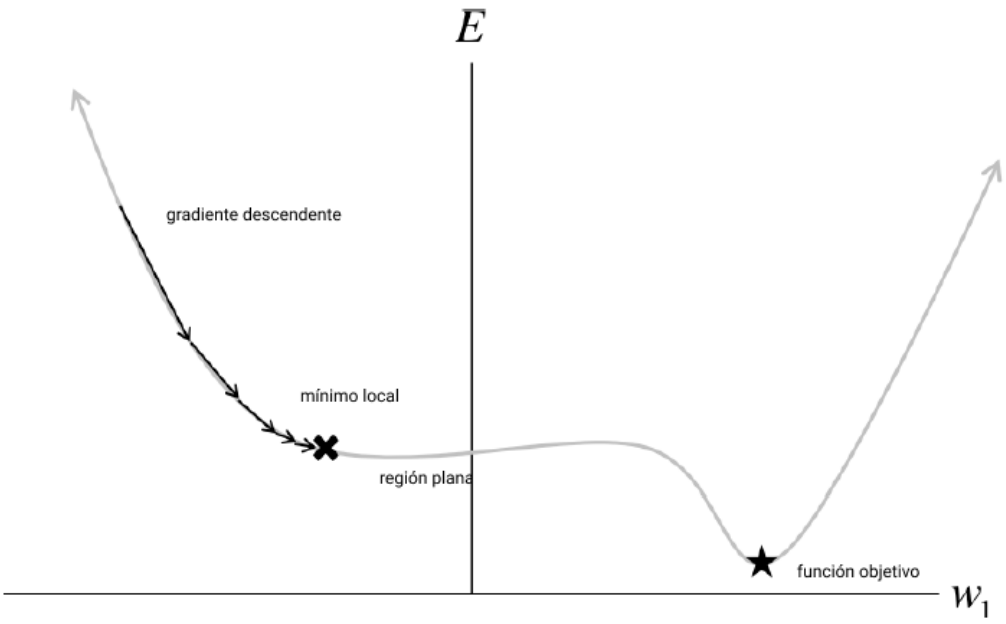

Figura 6. Superficie del error en un espacio unidimensional*. Fuente: [28]*

En la Figura 6 se observa la superficie del error en un espacio bidimensional. El error está en el eje vertical, y el peso (*w1*) en el eje horizontal. Se observan algunos de retos de minimizar el error [4] y llegar a la función objetivo, los cuales son:

1. **Mínimos locales**: Son puntos mínimos en la superficie que podrían confundir al sistema de que ya ha llegado a la solución óptima.
2. **Regiones planas**: Regiones de características planas, en las cuales el sistema puede quedarse atascado.
3. **Acantilados y gradientes que explotan**: Los cuales pueden causar que el gradiente se actualice con pasos muy grandes, provocando que los subsecuentes pasos sean muy grandes y el sistema no pueda encontrar un resultado óptimo.
4. **Dependencias de largo-corto plazo**: Cuando la secuencia es muy grande y el modelo no puede 'recordar' todas las dependencias, por ejemplo, en las redes neuronales recurrentes.
5. **Gradientes Inexactos**: En donde contamos con datos o sistemas con mucho ruido.

Los métodos de optimización más conocidos son el gradiente descendente estocástico, el cual realiza el proceso de optimización sobre lotes pequeños de datos, uno a la vez. Este proceso es repetido para varios lotes pequeños del conjunto de datos de entrenamiento, hasta que el promedio de la función objetivo deja de disminuir [3], [28].

La **tasa de aprendizaje** es la constante que define el tamaño de los pasos que da el gradiente en cada iteración. Esta afecta el rendimiento de todo el modelo y es uno de los hiperparámetros más difíciles de escoger. Si se escoge una tasa de aprendizaje muy pequeña podría no aprender lo suficientemente rápido y si se escoge una muy grande puede tener dificultades para converger [28].

Los algoritmos que son capaces de adaptar su tasa de aprendizaje a medida que realizan el entrenamiento, tienen ventajas muy significativas, y han sido uno de los mejores descubrimientos. Los más populares son *AdaGrad*, *RMSprop*, y *Adam* [28].

No hay un consenso entre los investigadores y expertos de cuál es el mejor método de optimización. Mientras que los resultados sugieren que los algoritmos con tasas de aprendizaje adaptativas son bastantes robustos, la selección depende las preferencias, y familiaridad del practicante con el algoritmo [4].

5.3 Arquitecturas de *Deep Learning*

5.3.1 **Redes Neuronales Convolucionales** (*CNN* por sus siglas en inglés)

Las *CNN* están inspiradas en cómo funciona los sistemas de visión en los mamíferos y en descubrimientos de la neurociencia [4]. Su nombre indica que emplean una operación llamada convolución, sin embargo, la convolución que es usada no corresponde a la que se usa en otros campos como en matemática pura.

Las *CNN* aprovechan la propiedad de las señales naturales respecto a que su representación es jerárquica, es decir la característica de nivel superior se obtiene al componer las características de nivel inferior [3]. Hay cuatro ideas principales detrás de las *CNN*, que toman ventaja de las señales naturales: conexiones locales, pesos compartidos, agrupación (*pooling*) y el uso de muchas capas. Las *CNN* son muy buenas en condiciones de alta selectividad e invarianza. Y Las aplicaciones modernas de *CNN* a menudo incluyen redes que contienen más de un millón de unidades [28].

La arquitectura de *CNN* ha sido una de las que mejores resultados ha mostrado [3]. Desde el año 2000, fueron aplicadas con gran éxito en la detección, segmentación y, reconocimiento de objetos y regiones en imágenes, pero a pesar de este éxito, las redes convolucionales fueron ignoradas hasta la competición de *ImageNet* en el 2012, donde las redes neuronales profundas fueron aplicadas obteniendo

resultados espectaculares [3]. Estos resultados se dieron en parte gracias a la utilización de *GPUs* y otras técnicas para mejorar su rendimiento.

5.3.2 Redes Neuronales Recurrentes (*RNN* por sus siglas en inglés)

Mientras que las *CNN* son buenas manejando datos en forma de cuadrícula (2 dimensiones) como imágenes de gran tamaño, las *RNN* son buenas manejando datos de una dimensión, secuencias largas de datos y secuencias que tienen diferentes tamaños [4]. Las *RNN* pueden formar conexiones en forma de "ciclos" con ellas mismas (Figura 7), y con otras neuronas de la red, permitiendo mapear la historia de las entradas anteriores hasta cada salida [4], [31].

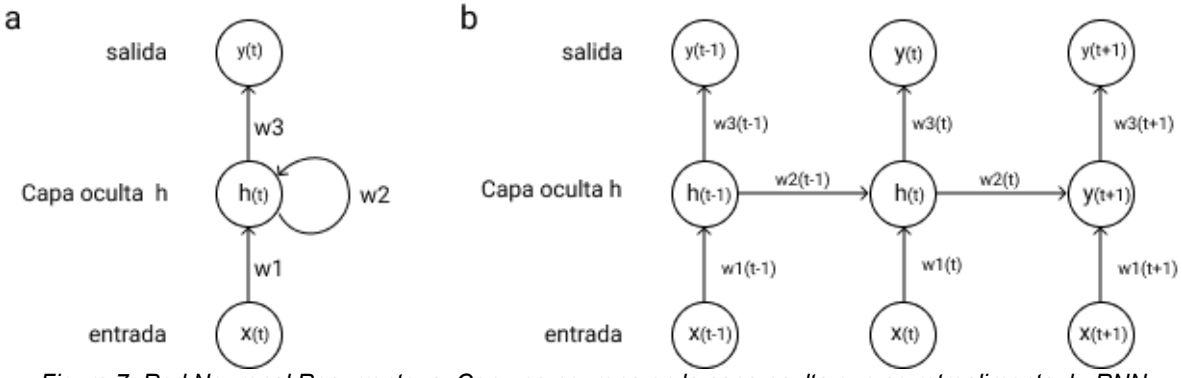

Figura 7. Red Neuronal Recurrente, a. Con una neurona en la capa oculta que se retroalimenta. b. RNN desenrollada, en diferentes tiempos. Fuente: propia.

La arquitectura de las *RNN* permite aprovechar la cualidad de compartir parámetros entre los distintos "módulos" de la red, lo que es muy importante cuando una pieza de información específica puede ocurrir en múltiples posiciones dentro de una secuencia. Por ejemplo, en tareas de *NLP* (*Natural Language Processing*) una palabra puede aparecer en diferentes posiciones en diferentes secuencias u oraciones. Una red neuronal de adelanto, tendría que tener parámetros separados para cada posición distinta. En cambio, una *RNN* comparte los mismos pesos para varias instancias de tiempo [4].

Las *RNN* utilizan los mismos métodos de optimización que se usan para las redes neuronales de adelanto, ya que como se ilustra en la figura 7(b), esta tiene una configuración bastante similar. Sin embargo, aunque las *RNN* teóricamente funcionan muy bien, presentan el problema de que el gradiente tiende a desaparecer, lo cual significa que este disminuye rápidamente cuando se calcula con respecto a entradas de varios pasos en el pasado, limitando severamente la capacidad del modelo para recordar dependencias de largo plazo, es decir la

influencia de las entradas de las capas ocultas, y por ende, la salida de la red decae o explota exponencialmente, lo que vuelve complicado entrenar el modelo con algoritmos de optimización basados en gradientes. Por esta razón los investigadores desarrollaron unidades especiales con la capacidad de recordar las secuencias de tiempo de forma más eficiente, unidades conocidas como LSTM (unidad de corto-largo plazo) y más recientemente *GRU* (unidad recurrente cerrada) [28], [31], [5].

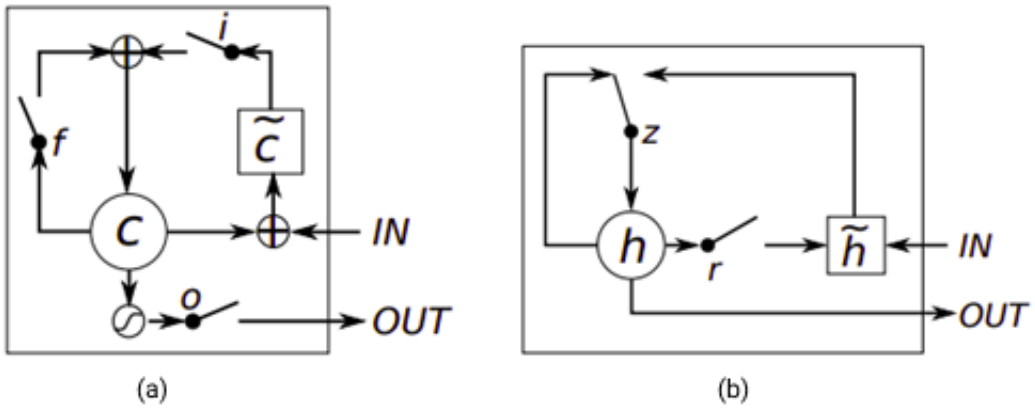

Figura 8. a. LSTM b. GRU. Fuente: [5]

5.3.2.1 *LSTM*

Las unidades *LSTM* [5] a diferencia de las *RNN* simples, seleccionan la información importante y olvida la que no es relevante para predecir el resultado de una secuencia. Su arquitectura se observa en la Figura 8a, tiene una memoria c_t^j en el tiempo t, la salida es h_t^j, o_t^j es una puerta de salida que modula la cantidad de exposición al contenido de la memoria. La celda de memoria c_t^j se actualiza para olvidar parcialmente la memoria existente y agregar un nuevo contenido $\tilde{c}_t^j$, la medida en la que se olvida la memoria existente es modulada por una puerta de olvido f_t^j, y el grado en que el nuevo contenido de memoria se agrega a la celda de memoria es modulado por una puerta de entrada i_t^j. Para ver más detalles de las ecuaciones y arquitectura remitirse a [5].

Mientras que las *RNN* convencional sobrescribe su contenido en cada paso de tiempo, las *LSTM* aprenden a identificar el contenido importante para mantenerlo en el tiempo, gracias a sus unidades de memoria y a olvidar el que encuentra que no es relevante, permiten mantener dependencias de largo plazo [5].

5.3.2.2 *GRU*

En [5] explican que, en 2014 se propuso una arquitectura que puede mantener dependencias a largo plazo, pero con menor complejidad que las *LSTM*, para hacer que cada unidad recurrente capture de forma adaptativa dependencias de diferentes escalas de tiempo. Sin embargo, *GRU* tiene unidades de compuerta que modulan el flujo de información dentro de la unidad sin tener celdas de memoria. La arquitectura de las *GRU* se observa en la figura 8b, h_t^j es la activación en el paso t, se realiza una interpolación lineal entre la activación previa h_{t-1}^j y la activación candidata $\tilde{h}_t^j$, donde la puerta de actualización z_t^j decide cuánto contenido o "activación" actualiza la unidad. Este procedimiento de tomar una suma lineal entre el estado existente y el nuevo estado calculado es similar a como lo hacen las unidades *LSTM*. Sin embargo, la *GRU*, no tiene ningún mecanismo para controlar el grado al que se expone su estado, y lo expone todo a la cada vez. Cuando está apagado, r_t^j cerca de 0, el cual es la puerta de reinicio, hace que la unidad actúe como si estuviera leyendo el primer símbolo de una secuencia de entrada, permitiéndole olvidar el estado previamente calculado. Para ver más detalles de las ecuaciones y arquitectura de las *GRU* remitirse a [5].

La investigación empírica hecha en [5] muestra las unidades *LSTM* y *GRU* son muy buenas manteniendo dependencias de largo plazo, y que las dos presentan rendimientos similares.

5.4 Medición

En [32] explican que la matriz de confusión es una Tabla (Tabla 1) que contiene información acerca de la clasificación predicha y la real para un sistema de clasificación:

Tabla 1. Matriz de confusión, para un problema binario.

		Predicho	
		Negativo	Positivo
Real	Negativo	a	b
	Positivo	c	d

Donde las variables de la matriz de confusión (Tabla 1), significan:

a: número de predicciones **correctas**, de que una instancia es **negativa**,
b: número de predicciones **incorrectas**, de que una instancia es **negativa**,
c: número de predicciones **correctas**, de que una instancia es **positiva,**
d: número de predicciones **incorrectas**, de que una instancia es **positiva**.

Una medida estándar en trabajos de polaridad de análisis de sentimiento es la **Exactitud** (*Accuaracy*) [33] la cual es la proporción del número total de predicciones que fueron correctas. Su ecuación está dada por:

$$AC = \frac{a + d}{a + b + c + d} \tag{5}$$

Además, otra técnica utilizada para corroborar que no haya sesgos en la medición es ***K-Fold Cross Validation*** [34], en la cual se divide el *Dataset* en K partes. Se usa una de estas para testear y las otras para entrenar. Se realizan K iteraciones donde para cada una los datos de entrenamiento y prueba son seleccionados aleatoriamente del *Dataset*, es decir que son distintos. si se divide el *Dataset* en K = 5, se realizan 5 entrenamientos con datos distintos del modelo. Esto asegura que el resultado sea preciso. Luego de que todas las pruebas son realizadas, el rendimiento se evalúa usando el promedio de los resultados de las iteraciones.

Finalmente, también se utiliza F-Measure:

$$F = \frac{2 * P * TP}{P + TP} \tag{6}$$

F-Measure [33]**,** [32] (ecuación 6) es una combinación armónica de Precisión (P) y *recall* o *true positive* (*TP*)

***Recall* o *True Positive Rate* (*TP*),** es la proporción de instancias positivas que fueron correctamente identificadas (ecuación 7):

$$TP = \frac{d}{c + d} \tag{7}$$

Precisión (P), es la proporción de instancias positivas que fueron predichas como positivas que fueron correctas (ecuación 8).

$$P = \frac{d}{b + d} \tag{8}$$

5.5 Dropout

En [35] explican que los sistemas de *Deep Learning* grandes tienen un número elevado de parámetros, lo que causa un sobre ajuste sobre los datos de entrenamiento y causa que los sistemas sean malos generalizando, es decir prediciendo casos que no ha visto antes, como con los datos de *test*. Ellos propusieron una técnica llamada *dropout*, la cual desconecta aleatoriamente algunos parámetros dentro de la red neuronal.

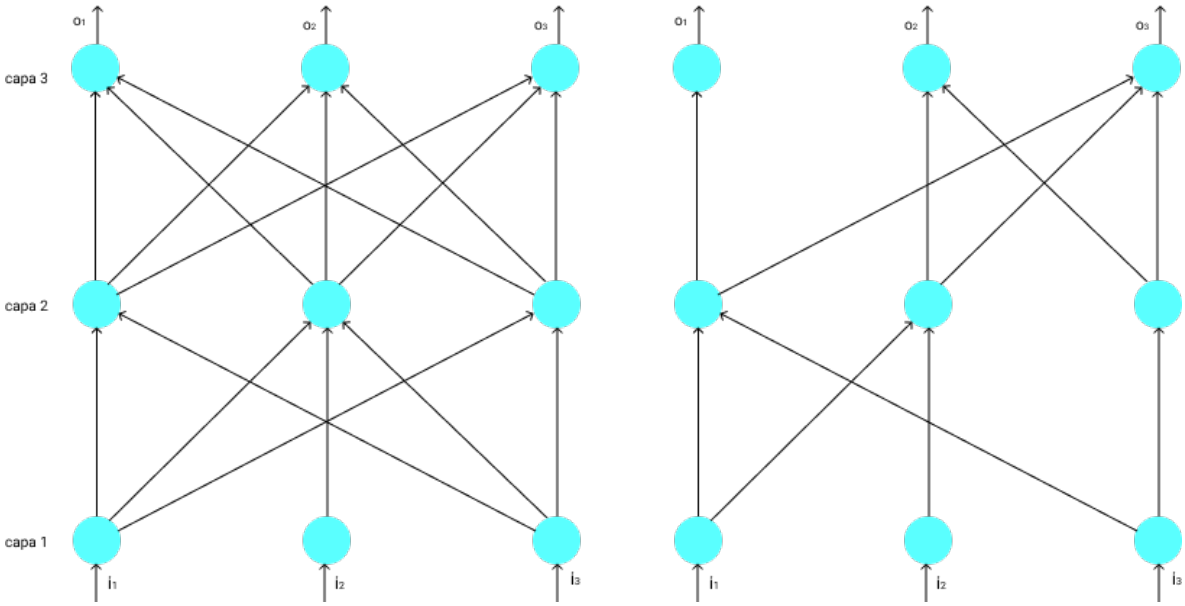

Figura 9. Red neuronal con dropout, la imagen de la izquierda es una red neuronal sin dropout, y la de la derecha es una red neuronal con dropout. Fuente: propia.

Como se observa en la Figura 9, la red neuronal de la izquierda no tiene *dropout*, y la red neuronal de la derecha tiene *dropout*, donde para esta segunda, hay varias conexiones entre sus neuronas desconectados. Según [35] el *Dropout* ha mostrado excelentes resultados en disminución del sobreajuste, lo cual conlleva a mejores valores de exactitud.

5.6 Corpus TASS

TASS [6], es el Taller de Análisis de sentimientos de la SEPLN, como satélite al evento de la Conferencia SEPLN 2012, realizado en la Universidad de Jaume I en Castellón de la Plana, comunidad Valenciana, España. En el cual se creó un corpus que contiene más de 70000 mensajes, basados en mensajes cortos extraídos de Twitter, publicados por personalidades importantes del mundo de la política, economía, medios de comunicación, cultura y otros. Entre noviembre de 2011 y

marzo de 2012. Los temas del corpus son política, fútbol, literatura o entrenamiento. Además, luego se han venido creando nuevos corpus, en diferentes países, haciendo más extensa esta base de datos.

El corpus principal de TASS esta etiquetado con 5 sentimientos: muy positivo (+P), Positivo (P), Neutro (Neu), Negativo (N), muy negativo (+N) y sin sentimiento (None). Además, cada oración tiene entidades o aspectos que también están etiquetadas con su respectivo sentimiento, y con un nivel de acuerdo o desacuerdo. Pero los demás corpus, simplemente tienen la clasificación de Positivo (P), Neutro (N), y Negativo (N). El corpus cuenta con un conjunto de entrenamiento y otro de test. El conjunto de entrenamiento está totalmente etiquetado, para entrenar y validar el modelo. Y el conjunto de test no tiene etiquetas.

El corpus esta presentado en formato *XML* (*Extensible Markup Language*), estructurado jerárquicamente como se observa en la siguiente figura (10):

Figura 10. Estructura del corpus – TASS, formato XML. Fuente: *[6]*

5.7 Herramientas de desarrollo

5.7.1 *Tensorflow*

Tensorflow [28], es una librería de software lanzada en 2015 por Google para facilitar el diseño, construcción, y entrenamiento modelos de *Deep Learning*. *Tensorflow* actualmente está en su versión 2, la cual está deseñada para facilitar el desarrollo de nuevos modelos, centrándose más en la ejecución que en los detalles de implementación [36].

Se basa en el cómputo de la información en forma de grafos, cada ecuación, puede ser descompuesta entre módulos más pequeños, permitiendo realizar operaciones en paralelo, lo que permite acelerar el tiempo de entrenamiento de un modelo de *Deep Learning*. Los nodos en el grafo representan las operaciones matemáticas,

donde las líneas entre estos, representan los datos que son comunicados entre uno y otro. Los datos son representados como tensores, los cuales son matrices multidimensionales [28], [36].

5.7.2 *Keras*

Keras [37] es una *API* de alto nivel escrita en Python [38], construida con el enfoque de permitir una rápida experimentación. Pone la experiencia de usuario como factor importante. Es modular, lo que significa que cada parte de la red neuronal es un módulo, haciéndola una *API* flexible y también muy poderosa para investigaciones avanzadas. Puede correr en Tensorflow o en otros frameworks [37]. Tensorflow tiene una implementación propia de Keras, llamada tf.keras, la cual tiene mejor mantenimiento e integración [39].

Métodos

En Keras [37] se ensamblan capas para construir modelos. Un modelo es usualmente un grafo de capas. Y el más común es el modelo secuencial:

model = tf.keras.Sequential()

Para crear una red neuronal, luego de definir que el modelo va a ser secuencial, se agregan las demás capas, algunos módulos disponibles:

- **Input**: es usada para instanciar un tensor de Keras.
- **Dense**: capa convencional de redes neuronales densamente conectada.
- **Dropout**: Aplica dropout a la entrada. Consiste en asignar 0 a una fracción de entradas, lo que ayuda a prevenir el sobre ajuste.
- **RNN**: Clase básica para capas de redes recurrentes.
- **GRU**: Gated Recurrent Unit.
- **Embedding**: Convierte los índices en vectores, puede ser usada solo como la primera capa del modelo.
- **Add**: capa que agrega una lista de entradas, o capas.

Mediante **model.add()** se agregan las capas. Luego se usa el método **model.compile(optimizer, loss, metrics)** este compila el modelo, lo prepara para ser ejecutado y sus parámetros significan:

- **optimizer**: el optimizador que usará el modelo.
- **loss**: la función de pérdidas a minimizar.

- **metrics**: array de métricas que se quieren obtener.

El optimizador para problemas binarios utilizado es, **binary_crossentropy**: crossentropy binario.

model.fit(Dataset, epochs, batch_size): Método para entrenar el modelo, toma tres argumentos:

- **Dataset**: el conjunto de datos de entrenamiento, con sus respectivas etiquetas (x_train, y_train).
- **epochs:** Número de épocas.
- **batch_size**: Entero que especifica el tamaño de cada batch. Los datos de entrenamiento se dividen en pequeñas partes, y el modelo itera sobre esos fragmentos durante el entrenamiento.

model.evaluate(x_test, y_test, batch_size): Evaluar el modelo con los datos de test.

- **x_test**: los ejemplos de test
- **y_test**: las etiquetas de los ejemplos de test

model.predict(x_test, batch_size): predicción sobre nuevos datos

- **x_test**: datos que se quieren predecir

Los métodos principales para **Procesamiento de Lenguaje:**

Tokenizer: esta clase permite vectorizar un corpus de texto, asignando un índice a cada palabra, creando un diccionario de índices, los valores más frecuentes de palabras en el corpus, tendrán índices más cercanos a 1.

text_to_word_sequence: convierte una secuencia de índices a vectores, que puedan ser entendidos por el modelo.

Preprocesamiento de secuencias:

pad_sequences: permite trabajar con secuencias de diferente longitud, creando una matriz con el mismo número de columnas, longitud definida por el usuario, donde se alojarán los datos de ejemplo de, para que puedan ser entrenados por el modelo. Llena los espacios vacíos con ceros.

- **secuences**: Lista de listas, donde cada elemento es una secuencia.
- **maxlen**: int, máxima longitud de todas las secuencias.
- **padding**: String, ‘pre’ o ‘post’, rellena los espacios vacíos de los vectores antes o después de la secuencia.

- **truncating**: String, ‘pre’ o ‘post’, remueve valores de la secuencia de longitud superior a *maxlen*, al inicio o final de la secuencia.

5.7.3 *Google Colaboratory*

Google Colaboratory [40] Permite escribir y ejecutar Python en el navegador de internet, con cero configuraciones, acceso libre a GPUs, y facilidad de compartir y colaborar. El código se escribe y se ejecuta en **Colab Notebook**, el cual permite combinar código ejecutable y texto enriquecido en un solo documento, con imágenes, *HTML*, *LaTeX*, *Markdown*, entre otros. Cuando se crea un *Colab notebook*, se guarda directamente en drive, lo que también permite almacenar datos, y consumirlos directamente de drive, como por ejemplo un *Dataset* [40].

Colab Notebooks, permite escribir código en bloques, y ejecutarlos independientemente, lo que permite mayor organización a la hora de realizar desarrollos e investigaciones en áreas de *Machine Learning*.

Para consumir un *Dataset* de drive, se debe crear una instancia acceso a drive:

```
from google.colab import drive
drive.mount('/content/drive')
```

Luego, se puede acceder a cualquier carpeta de drive, simplemente instanciando la dirección a la cual se quiere acceder.

Para ejecutar el código de un bloque, los Notebooks, cuentan con un botón de ejecución del código.

6. DISEÑO METODOLÓGICO

En este capítulo se describirá la metodología seguida, para llevar a cabo el análisis de polaridad de sentimiento.

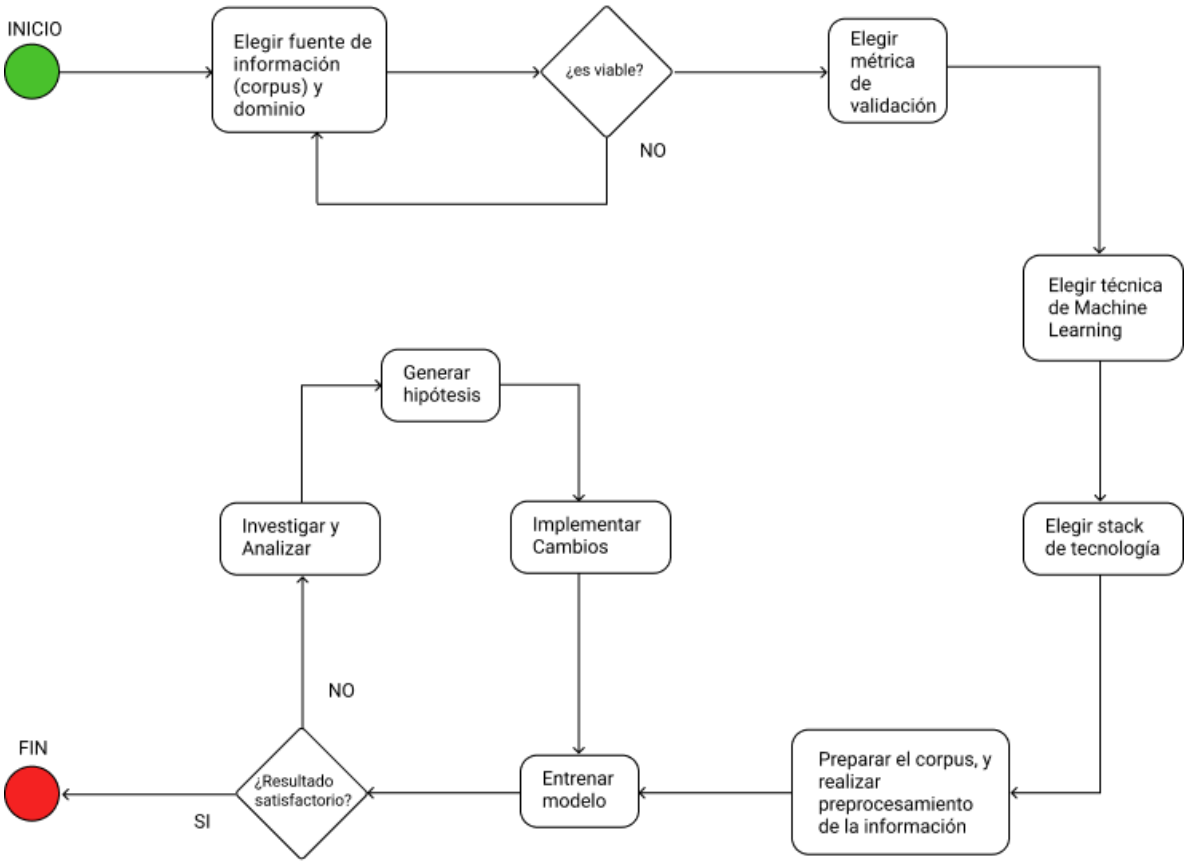

Figura 11. Algoritmo de la metodología empleada. Fuente: propia.

La figura 11, muestra el proceso metodológico seguido para lograr los objetivos planteados, el cual costa en términos generales de: Elección de la fuente de información y dominio. Selección del método de *Machine Learning*, y las tecnologías a usar. Preprocesamiento de la información. Implementación y entrenamiento del modelo. Finalmente, evaluación de los resultados, y documentación. Explicados a continuación:

6.1 Elegir fuente de información y dominio.

6.1.1 Fuente de información para entrenar el modelo

Se utilizó el corpus de TASS, fue necesario pedir acceso a Eugenio Camara-Martinez, uno de sus autores y llenar un formulario con fines de confidencialidad, sin embargo, es fácil acceder para quien desee trabajar en estos temas. Se permitió el acceso a una página web, donde se encontraba el corpus generado por el trabajo del 2012 en [6], además otros corpus realizados en los siguientes años por diversos grupos de otros países de Costa Rica, España, Uruguay, México y Perú. Logrando un total de 49.444 oraciones etiquetadas.

Tabla 2. Lista de tópicos TASS del trabajo de [6], sin contar los tópicos de los demás corpus en otros países.

Tópico	Frecuencia
Política	3119
Otros	2337
Entretenimiento	1677
Economía	942
Música	566
Futbol	252
Cine	245
Tecnología	217
Deportes	113
Literatura	99

Como se observa en la Tabla 2, son varios los tópicos de los tuits. Se observa que el segundo tópico más frecuente es otros tópicos, es decir que tiene bastante variedad de temas. Además, se aclara que esta Tabla solo recuenta los tópicos de [6] que fue el corpus inicial realizado en España. Para el resto de corpus, es decir los generados por grupos asociados a TASS en otros países, no fue posible acceder al detalle de los tópicos, sin embargo, se considera útil la información de la Tabla 2 ya que da una noción general de la composición de los corpus, los cuales fueron generados en subsecuentes años, generando corpus algunos países como Perú, Costar Rica, México y Uruguay.

Antes de conseguir el corpus de TASS, se intentó construir un *Dataset* con base en las revisiones y comentarios de usuarios acerca de locales comerciales, de plataformas como TripAdvisor, o Google Maps, las cuales cuentan con *APIs* desde las cuales se pueden bajar los datos, pero se encontró que eran de pago, lo que hizo inviable esta opción. Especialmente se investigó acerca de Google Maps

Plataform, ya que da 200USD de dólares gratis por mes, y un extra de 300USD la primera vez que se abre una cuenta en su plataforma, sin embargo, para descargar una cantidad de 50000 datos, tamaño del corpus de IMBD [15] que se consideró estándar, se estimó un costo de 2.133,33 USD, lo cual lo hizo inviable, además de que es necesario poner una tarjeta de crédito para utilizar los créditos gratuitos

También se encontró que en inglés hay múltiples corpus etiquetados de análisis de sentimiento como IMBD [15] que específicamente es un sitio donde los usuarios hacen revisiones de películas y ha generado un corpus muy popular en lenguaje Ingles, para análisis de polaridad de sentimientos.

6.1.2 Dominio

Según el numeral anterior, el dominio del corpus es: Comentarios cortos en Twitter de lenguaje español de políticos, famosos, periodista y usuarios en general, en diferentes países como España, Uruguay, Perú, México y Costa Rica, acerca de diferentes temas.

6.2 Elección de la métrica de validación

Investigando en la Literatura [1], [10], [11] se encontró que la Exactitud (ecuación 5) es una medida generalmente utilizada y aceptada para validar los modelos de polaridad de análisis de sentimiento.

6.3 Elección del modelo de Machine Learning

Se escoge como método *Deep Learning* con arquitectura de *RNN* con unidades de *GRU,* ya que como se enuncia en el marco teórico, el *Deep Learning* evita la selección de características de forma manual y ha presentado muy buenos resultados en el estado del arte. Las *RNN* se especializan en procesar secuencias y la Polaridad de Sentimiento de las oraciones, depende del contexto y la secuencia de las palabras. Las unidades *GRU* ayudan a recordar dependencias de largo plazo y son más simples que las unidades *LSTM*.

6.4 Selección del *stack* de tecnología

Como se nombra en el marco teórico, *tensorflow* y *keras* son herramientas poderosas, y de fácil uso, para entrenar modelos basados en *Deep Learning* y bastante aceptados, con un continuo soporte por parte de Google. El lenguaje de programación elegido fue *python*, que es uno de los más usados para tareas de *Machine Learning*. Además, el entorno de desarrollo elegido es *Google Colaboratory*. La versión de tensorflow utilizada fue 2.2.0 y de tf.keras 2.3.0-tf.

6.5 Preparación del *Dataset* y preprocesamiento

6.5.1 Preparación del *Dataset* TASS

Debido a que los archivos estaban en formato XML, era necesario poder extraerlos y filtrarlos para crear el *Dataset* de entrenamiento y pruebas, con etiquetas según su polaridad, se siguió la secuencia de pasos:

6. Cargar los corpus al **espacio de trabajo (notebook)**
7. Añadir a un arreglo ***files*** cada corpus
8. Itera ***files*** para filtrar los datos, donde:
    a. Se crea un arreglo ***x_data*** con las oraciones
    b. Se crea un vector ***y,*** con valor 1 para sentimientos positivos, y 0 para negativos, en la posición **i**, que coincide con su respectiva oración en ***x_data*.**
9. Luego se dividieron los ejemplos para los datos de *train* y *test*, según un % x.

6.5.2 Preprocesamiento

Debido a que los modelos de *Machine Learning* solo entienden números, fue necesario convertir cada palabra a un vector representativo de esta, y además, preparar los datos de una manera adecuada. También es necesario filtrar el 'ruido' del corpus, como caracteres, tildes. Procesos se enunciarán a continuación.

6.5.2.1 Tokenizador

La fase de tokenización consiste en asignar un valor numérico en forma de vector a cada palabra, creando un vocabulario de palabras. Según la frecuencia de aparición, las palabras se ubicarán en la primera parte del vocabulario, es decir con un índice menor, entre más frecuente sean. Para esto, se implementó a nivel de código el siguiente algoritmo:

1. Se importa la librería Tokenizer de *keras*
2. Se define el número límite de palabras en el vocabulario
3. Se crea el vocabulario, mediante la función Tokenizer, ingresándole como parámetro el corpus. El tokenizador vuelve todas las palabras a minúsculas, y elimina todos los símbolos, como puntos, comas, comillas, arrobas, entre otros. Lo que garantiza que el corpus para todos los experimentos esté en minúscula, y con filtro de símbolos.

6.5.2.2 Truncamiento de los Datos y rellenado

Las oraciones en el corpus tienen diferentes longitudes. Se necesitan que los vectores representativos de cada oración tengan el mismo tamaño para realizar el entrenamiento. Se define un tamaño máximo por oración, que recoja la mayoría de palabras.

Se definió el tamaño del vector de entrada al sistema, calculando el promedio de palabras por oración de todo el corpus y agregándole dos desviaciones estándar para lograr tener en cuenta la mayor cantidad de datos, sin hacer el vector demasiado extenso, dando un resultado de 29. Se calculo en porcentaje de oraciones que tienen una longitud menor o igual a 29, dando como resultado 99,19%, es decir que con un tamaño de vector de 29 se cubren totalmente el 99,19 de las oraciones, sin truncarlas.

Hay muchas oraciones que tienen una longitud inferior al límite máximo, los espacios vacíos deben ser rellenados con un valor que le indique al sistema que es un espacio vacío y no una palabra desconocida. Se rellenaron los espacios vacíos antes de la oración.

Se siguieron los siguientes pasos para llevar a cabo el truncamiento y rellenado:

1. Se calcula en número de tokens por oración.
2. Se calcula el promedio de tokens por oración, siendo 17,4.
3. Se calcula el número de tokens máximo en las oraciones, dando 47.
4. Se define el número máximo de tokens, como el promedio de tokens, más dos desviaciones estándar, dando 29.
5. Se calcula el porcentaje de datos del corpus que se cubren al hacer el truncamiento, dando 99.19%.
6. Se define el pad (rellenado) como pre, es decir que se rellenaran los espacios faltantes para completar la longitud máxima, antes de empezar la oración.
7. Finalmente se convierte en tokens, representando las palabras de cada oración con los índices respectivos. Se realiza para los *Datasets* de *train* y *test*.

```
El amanecer respirando o2 puro, es más que un regalo... más si es a las 6:00 am frente al mar... preparando la mente con meditación.
[    0     0     0     4  3027 13636 17488  1375    15    36     8    13
   471    36    35    15     5    20   287   711  2930   364    25  1269
  1851     2  2392    16 17489]
```

Figura 12. Ejemplo de representación vectorial de una oración, luego de tokenizar, truncar y rellenar. Fuente: propia.

E la figura 12, se observa que los 3 primeros datos del vector resultante, contienen 0. Esto sucede porque la oración tiene 26 palabras, y sobran 3 espacios para completar la longitud del vector. Cada número, es el índice de las palabras en el vocabulario, resultante del proceso de tokenización.

Con esto, se han dejado los datos listos para el entrenamiento. Ahora, se procede a implementar el algoritmo de *Deep Learning*, el flujo de preprocesamiento y de entrenamiento se muestra a continuación, en la figura 13:

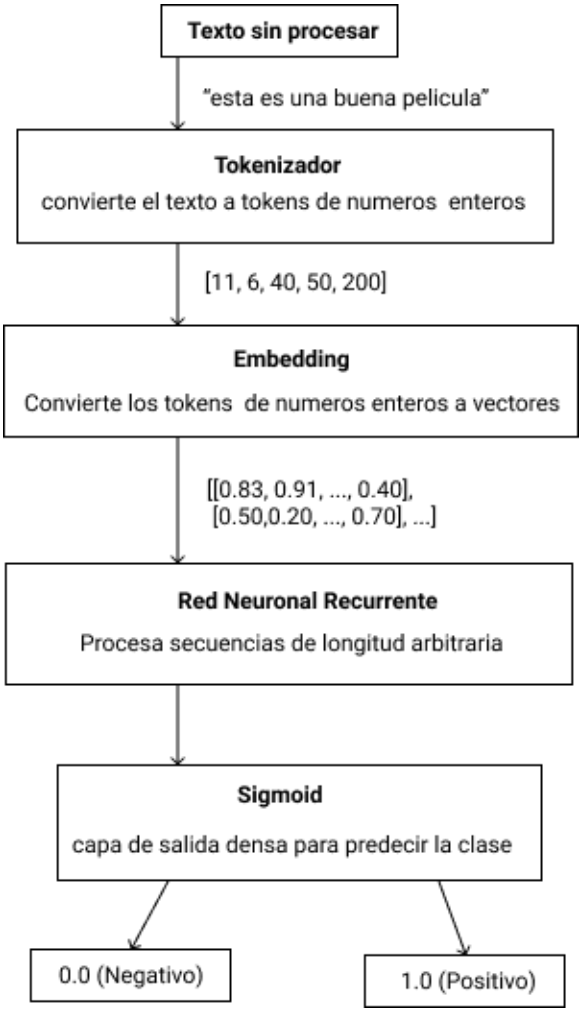

Figura 13. Diagrama de flujo del algoritmo implementado, preprocesamiento de la data, y la red neuronal profunda, dividida en capa embedding, red neuronal recurrente, y capa densa sigmoid. Fuente: propia.

6.6 Implementación de modelo de *Deep Learning*

Como se muestra en la Figura 12, el modelo de *Deep Learning*, consta de tres partes:

1. **Capa *embedding***: la cual se encarga de realizar de forma automática la selección de características. Convierte los tokens, en vectores de longitud n, donde entre más similitud semántica tengan las palabras más cerca estarán en el espacio vectorial.
2. ***RNN***: la cual está compuesta de varias capas de unidades GRU.
3. Función de activación **sigmoid**: la cual da como resultado un valor entre 0 y 1. Entre más este cerca de 1 este valor, es más probable que la predicción sea de una polaridad positiva, y entre más cercano a 0, más probable será que la polaridad sea negativa.

Los pasos a nivel de keras para implementar el modelo de *Deep Learning*, son:

1. Crear un modelo secuencial, con el método de *keras*, Sequential()
2. Definir el tamaño de cada vector ***embedding***
3. Se agrega la capa ***embedding*** al modelo
4. Se agregan, las capas GRU, con n número de unidades por capa, y n cantidad de dropout
5. Se compila el modelo, definiendo la función de perdida loss (binary_crossentropy), el optimizador (*Adam*) y la métrica a optimizar (exactitud):

6.7 Entrenamiento

Se procede a entrenar el modelo, mediante la función model.fit()

6.8 Evaluación

La métrica de validación elegida fue Exactitud. Se evalúa, con datos de prueba que el sistema no ha visto durante la fase de entrenamiento. Luego se calcula la matriz de confusión confusion_matrix de sklearn.metrics, para asegurarse de que los datos no estén sesgados hacia alguna polaridad.

Se establece un límite de 0,5 para establecer la polaridad de las predicciones, debido a que haciendo pruebas empíricas se observó que para cada experimento este valor limite podía cambiar y 0,5 se ajustaba en general bien para producir los

valores de exactitud esperados. Quedando así el rango de [0 a 0,5] para polaridad negativa, y de (0,5 a 1] para polaridad positiva.

6.9 Iteración y cambios

Con el resultado de Exactitud, se evalúa si es satisfactorio según los resultados del estado del arte. Si no lo es, se analiza los resultados, se investiga en la literatura, a partir de ello se genera una hipótesis de que cambio se puede hacer, se implementan los cambios, se vuelve a entrenar el modelo y se repite el ciclo, hasta que los resultados de Exactitud sean satisfactorios.

Los cambios realizados, pueden ser en el corpus, como cambiar el porcentaje de ejemplos para entrenamiento y *test*, o filtrar tildes y etiquetas, o los cambios pueden ser en hiperparámetros del modelo, como agregar *dropout* a las capas GRU, modificar el número de unidades por capa, cambiar el tamaño de la capa *embedding* por oración entre otros.

Este proceso se realizó varias veces, y esta condensado en cada uno de los experimentos documentados en la sección de resultados en el capítulo 7, en el marco teórico, con en el sustento teórico de cada cambio y con la generación de hipótesis en cada análisis de los resultados.

6.10 Analizar, documentar y compartir.

Esta parte, comprende la documentación del trabajo y de las conclusiones, presentes en este documento. Luego de la aprobación metódica por parte de los jurados, se realiza la presentación formal y se comparte con la comunidad académica mediante los repositorios de la Universidad Santo Tomás, Colombia.

7. RESULTADOS

El capítulo de resultados está compuesto por la documentación de los Experimentos realizados. En cada experimento se expone una Tabla de hiperparámetros, matriz de confusión, y gráfica de exactitud para los conjuntos de *train* y *test*. Cada gráfica tiene las líneas de exactitud a través de las épocas para el experimento actual, y con líneas entrecortadas para el anterior. Además, se analiza cada uno de los resultados y se generan nuevas hipótesis para mejorar la exactitud. El total de ejemplos entre datos de *train* y *test* es de 49.444.

La máquina de Google Colaboratory utilizada, cuenta con dos procesadores del modelo: `Intel(R) Xeon(R) CPU @ 2.30GHz.`

Los Hiperparámetros que serán configurados son:

1. **Unidades *GRU* por capa**, inicialmente se definen 3 capas *GRU*, y cada una de ellas tiene un numero x de unidades.
2. ***W2V***, es el tamaño por vector de la primera capa de la red neuronal, o capa *embedding* que representa vectorialmente a cada palabra del vocabulario como se observa en la figura 12.
3. **Épocas**, es el número de iteraciones sobre un mismo *Dataset*
4. ***# train,*** cantidad de ejemplos para la fase de *train*.
5. ***# test,*** cantidad de ejemplos para el *test.*
6. **Filtrado**, se refiere a los caracteres y símbolo que se filtró en el corpus.
7. ***Adam,*** es el optimizador utilizado, y el *lr* es el *learning rate*, o tasa de aprendizaje por su traducción al español.
8. **Parámetros***,* es el número de parámetros que se generan dentro de la red neuronal.
9. **Tiempo *train***, es el tiempo que duró entrenando el modelo.
10. ***AC***, es la exactitud del modelo, en este caso para t*rain* y *test.*

7.1 Experimento 1. Se definen valores iniciales de Hiperparámetros.

Tabla 3. Hiperparámetros y datos relevantes Experimento 1.

<table>
<tr><th>#</th><th>GRU 1</th><th>Dropout</th><th>GRU 2</th><th>dropout</th><th>GRU 3</th><th>Dropout</th><th>AC
Train</th></tr>
<tr><td rowspan="6">1</td><td>16</td><td>-</td><td>8</td><td>-</td><td>4</td><td>-</td><td rowspan="2">99,63%</td></tr>
<tr><td>W2V</td><td>Épocas</td><td># train</td><td># test</td><td>filtrado</td><td>Adam</td></tr>
<tr><td>8</td><td>10</td><td>24.722</td><td>24.722</td><td>símbolos</td><td>lr = 3</td><td>Test</td></tr>
<tr><td colspan="2">Parámetros</td><td colspan="2">Tiempo train</td><td colspan="2">RAM</td><td rowspan="2">84,75%</td></tr>
<tr><td colspan="2">402.045</td><td colspan="2">5min 24s</td><td colspan="2">1,14/12,72 GB</td></tr>
</table>

Como se observa en la Tabla 3, los hiperparámetros iniciales fueron:

1. Se definen 16, 8 y 6 unidades *GRU* por capa *GRU*.
2. Un W2V de tamaño 8 (es decir que cada palabra estará representada por un vector de 8 posiciones).
3. Se entrena con 10 épocas.
4. El optimizador Adam, se ajusta con un *lr* de 3.
5. Solo se filtran símbolos, estos son: '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n'.
6. El número de ejemplos para train es el 50% de los datos y 50% para *test*, resultando en 24.722 y 24.722 ejemplos respectivamente.

Además, como se puede ver en la tabla de hiperparámetros (Tabla 3) se muestran valores relevantes de cada experimento, como el número de parámetros del modelo, el tiempo de entrenamiento, la RAM usada en el entrenamiento del modelo, y los resultados de Exactitud en *train* y *test*. Así se seguirá manejando para los demás experimentos.

Inicialmente de las 24.722 instancias de test, 11.018 son negativas y 13.704 positivas, 44,57% y 55,43% respectivamente.

Tabla 4. Matriz de Confusión Experimento 1.

<table>
<tr><th rowspan="2">Experimento</th><td rowspan="2">1</td><th colspan="2">Predicho</th></tr>
<tr><th>Negativo</th><th>Positivo</th></tr>
<tr><td rowspan="2">Real</td><td>Negativo</td><td>9266</td><td>1752</td></tr>
<tr><td>Positivo</td><td>2017</td><td>11687</td></tr>
</table>

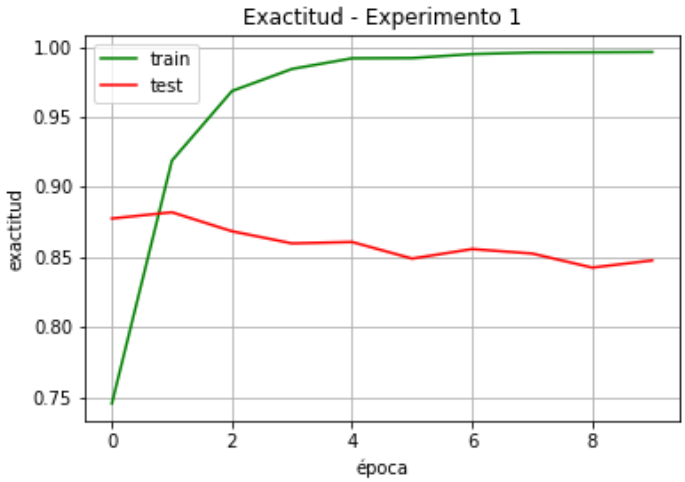

Figura 14. Exactitud Experimento 1, train: 99,63%, test: 84,75%. Fuente: propia.

El experimento 1 (Tabla 3), resultó en una exactitud de 99,63% sobre los datos de *train*, y en **84,75%** sobre los datos de *test* (Figura 14). Se observa un sobreajuste con los datos de *test* (Figura 14) desde la época 1.

Según la matriz de confusión (Tabla 4), el porcentaje de oraciones negativas predichas correctamente fue de 84,1% y el de oraciones positivas predichas correctamente fue de 85,28%, siendo valores cercanos y confirmando que no hay sesgos hacia alguna polaridad. Además, el F-Measure fue de 85,61%.

7.2 Experimento 2. Se agrega un *Dropout* de 0,5 a cada capa *GRU* para disminuir el sobreajuste**.**

Tabla 5. Hiperparámetros y datos relevantes Experimento 2, dropout 0,5.

<table>
<tr><th>#</th><th>GRU 1</th><th>Dropout</th><th>GRU 2</th><th>dropout</th><th>GRU 3</th><th>Dropout</th><th>AC
Train</th></tr>
<tr><td rowspan="6">2</td><td>16</td><td>0,5</td><td>8</td><td>0,5</td><td>4</td><td>0,5</td><td rowspan="2">97,88%</td></tr>
<tr><td>W2V</td><td>Épocas</td><td>train</td><td>test</td><td>filtrado</td><td>Adam</td></tr>
<tr><td>8</td><td>10</td><td>24.722</td><td>24.722</td><td>símbolos</td><td>lr = 3</td><td>Test</td></tr>
<tr><td colspan="2">Parámetros</td><td colspan="2">Tiempo train</td><td colspan="2">RAM</td><td rowspan="2">86,19%</td></tr>
<tr><td colspan="2">402.045</td><td colspan="2">5min 42s</td><td colspan="2">1,14/12,72 GB</td></tr>
</table>

Tabla 6. Matriz de Confusión Experimento 2.

Experimento	2	**Predicho**	
		Negativo	**Positivo**
Real	**Negativo**	9215	1803
	Positivo	1605	12049

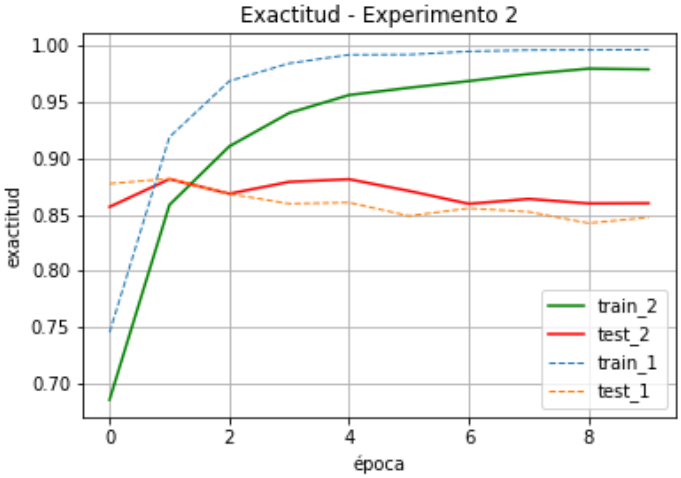

Figura 15. Exactitud Experimento 2, dropout 0,5, train: 97,88%, test: 86,19%. Fuente: propia.

El experimento 2 (Tabla 5), resultó en una exactitud de 97,88% sobre los datos de *train*, y en **86,19%** sobre los datos de *test (Figura 15)*. La exactitud sobre los datos de *test* fue de +1,43% frente al experimento 1, en esta gráfica se observa la exactitud en las épocas, las líneas continuas son los resultados de este experimento, y las entrecortadas, los resultados del experimento anterior (1), se observa que con los datos de test el resultado fue mejor. Aunque mejoró la Exactitud, aun se observa sobreajuste.

Según la matriz de confusión (Tabla 6), el porcentaje de oraciones negativas predichas correctamente fue de 83,74% y el de oraciones positivas predichas correctamente fue de 88,25%, siendo valores cercanos y confirmando que no hay sesgos hacia alguna polaridad. Además, el F-Measure fue de 87,99%.

7.3 **Experimento 3.** Se aumenta el *Dropout* a 0,7.

Tabla 7. Hiperparámetros Experimento 3, dropout 0,7.

#	GRU 1	Dropout	GRU 2	dropout	GRU 3	Dropout	AC Train
3	16	**0,7**	8	**0,7**	4	**0,7**	95,18%
	W2V	**Épocas**	**train**	**test**	**filtrado**	**Adam**	
	8	10	24.722	24.722	símbolos	lr = 3	**Test**
	Parámetros		**Tiempo train**		**RAM**		87,04%
	402.045		5min 48s		1,22/12,72 GB		

Tabla 8. Matriz de Confusión, Experimento 3.

Experimento	3	**Predicho**	
		Negativo	**Positivo**
Real	**Negativo**	9077	1941
	Positivo	1264	12440

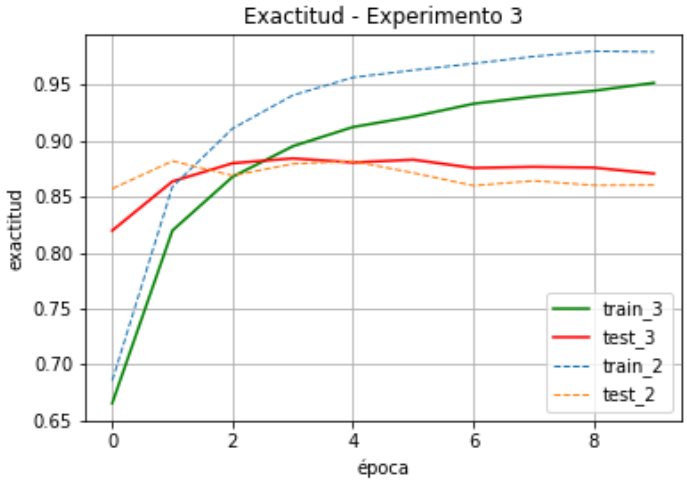

Figura 16. Exactitud experimento 3, dropout: 0,7, train: 95,18%, test: 87,04%. Fuente: propia.

Agregando un *dropout* de 0,7 (Tabla 7) la exactitud fue de 95,18% sobre los datos de *train*, y de **87,04**% sobre los datos de *test*. La exactitud aumentó en 0,85% en

test, y disminuyó en 2,75% en train (Figura 16). Aunque el resultado de exactitud mejoro, aun se observa un sobreajuste. Se procede a aumentar el *dropout* a 0,8.

La matriz de confusión (Tabla 8), muestra que el porcentaje de oraciones negativas predichas correctamente fue de 82,38% y el de oraciones positivas predichas correctamente fue de 90,70%, siendo valores cercanos y confirmando que no hay sesgos hacia alguna polaridad. Además, el F-Measure fue de 89,89%.

7.4 **Experimento 4.** Se aumenta el *Dropout* a 0,8.

Tabla 9. *Hiperparámetros y datos relevantes* Experimento 4, dropout 0,8.

#	GRU 1	Dropout	GRU 2	dropout	GRU 3	Dropout	AC Train
4	16	**0,8**	8	**0,8**	4	**0,8**	91,08%
	W2V	**Épocas**	**train**	**test**	**filtrado**	**Adam**	
	8	10	24.722	24.722	símbolos	lr = 3	**Test**
	Parámetros		**Tiempo train**		**RAM**		88,54%
	402.045		5min 26s		1,17/12,72 GB		

Tabla 10. Matriz de Confusión, Experimento 4.

Experimento	4	Predicho	
		Negativo	**Positivo**
Real	**Negativo**	9582	1436
	Positivo	1396	12308

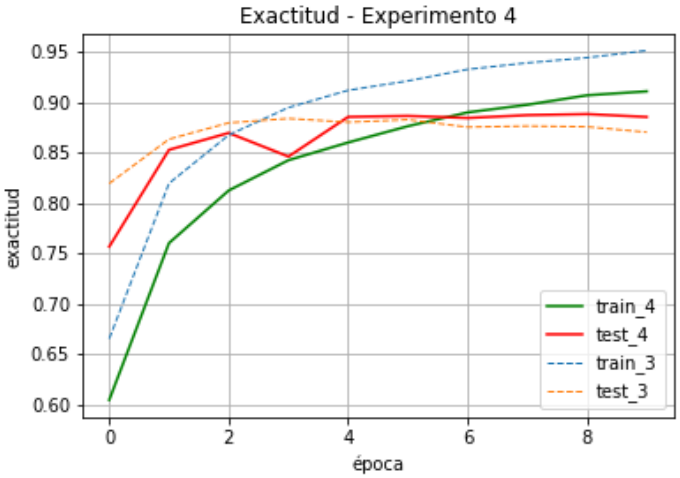

Figura 17. Exactitud experimento 4, dropout: 0,8, train: 91,08%, test: 88,54%. Fuente: propia.

Al agregar un *dropout* de 0,8 (Tabla 9) se obtuvo una exactitud de 91,08% sobre los datos de *train*, y de **88,54%** sobre los datos de *test (Figura 17)*. La exactitud sobre los datos de *test* aumentó 1,51% frente al experimento 3, y la exactitud sobre los datos de *train* disminuyo 4,05%, es decir que con 0,8 de *Dropout* los resultados mejoraron. Se observa un sobreajuste luego de la época 5 (Figura 17), además en los demás experimentos como en el 2 y 3 (Figura 15 y 17 respectivamente) se observa que luego de la mitad, ocurre con mayor sobreajuste, y la exactitud empieza a disminuir.

Según la matriz de confusión (Tabla 10), el porcentaje de oraciones negativas predichas correctamente fue de 86,97% y el de oraciones positivas predichas correctamente fue de 89,81%, siendo valores cercanos y confirmando que no hay sesgos hacia alguna polaridad. Además, el F-Measure fue de 89,76%.

7.5 **Experimento 5.** Se pasa de 10 a 5 épocas de entrenamiento.

Tabla 11. Hiperparámetros y datos relevantes Experimento 5, 5 épocas.

#	GRU 1	Dropout	GRU 2	dropout	GRU 3	Dropout	AC Train
5	16	0,8	8	0,8	4	0,8	86,65 %
	W2V	**Épocas**	**train**	**test**	**filtrado**	**Adam**	
	8	**5**	24.722	24.722	símbolos	lr = 3	**Test**
	Parámetros		**Tiempo train**		**RAM**		88,53%
	402.045		2min 54s		1,19/12,72 GB		

Tabla 12. Matriz de Confusión, Experimento 5.

Experimento	5	Predicho	
		Negativo	Positivo
Real	Negativo	9863	1155
	Positivo	1681	12023

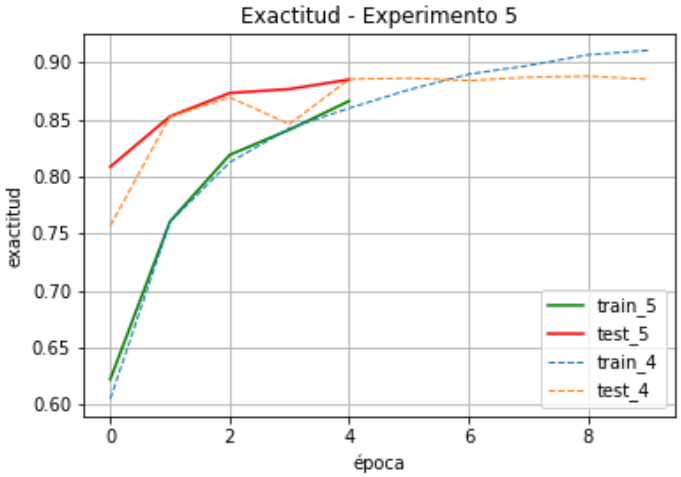

Figura 18. Exactitud experimento 5, 5 épocas, train: 86,65%, test: 88,53%. Fuente: propia.

Al disminuir las épocas a 5 (Tabla 11) se obtuvo una exactitud de 86,65,08% sobre los datos de *train*, y de **88,53%** sobre los datos de *test (Figura 18)*. La exactitud sobre los datos de test disminuyó en 0,01% frente al experimento 4, la exactitud

sobre los datos de train disminuyó 4,43%. No se presentó sobreajuste (Figura 18). Aunque disminuyó la exactitud sobre los datos de test, se considera un valor despreciable, ya que los resultados con los mismos hiperparámetros pueden variar en cada ejecución, y se valora más la disminución del sobreajuste.

Según la matriz de confusión (Tabla 12), el porcentaje de oraciones negativas predichas correctamente fue de 89,52% y el de oraciones positivas predichas correctamente fue de 87,73%, siendo valores cercanos y confirmando que no hay sesgos hacia alguna polaridad. Además, el F-Measure fue de 88,41%.

7.6 **Experimento 6.** Se aumenta tamaño de W2V a 100.

Tabla 13. Hiperparámetros y datos relevantes Experimento 6, tamaño word2vec = 100.

<table>
<tr><th rowspan="2">#</th><th>GRU 1</th><th>Dropout</th><th>GRU 2</th><th>dropout</th><th>GRU 3</th><th>Dropout</th><th>AC</th></tr>
<tr><th></th><th></th><th></th><th></th><th></th><th></th><th>Train</th></tr>
<tr><td rowspan="5">6</td><td>16</td><td>0,8</td><td>8</td><td>0,8</td><td>4</td><td>0,8</td><td rowspan="2">93,67 %</td></tr>
<tr><td>W2V</td><td>Épocas</td><td>train</td><td>test</td><td>filtrado</td><td>Adam</td></tr>
<tr><td>100</td><td>5</td><td>24.722</td><td>24.722</td><td>símbolos</td><td>lr = 3</td><td>Test</td></tr>
<tr><td colspan="2">Parámetros</td><td colspan="2">Tiempo train</td><td colspan="2">RAM</td><td rowspan="2">87,98%</td></tr>
<tr><td colspan="2">5’006.461</td><td colspan="2">5min 48s</td><td colspan="2">1,19/12,72 GB</td></tr>
</table>

Tabla 14. Matriz de Confusión, Experimento 6.

Experimento	6	**Predicho**	
		Negativo	**Positivo**
Real	**Negativo**	9534	1484
	Positivo	1487	12217

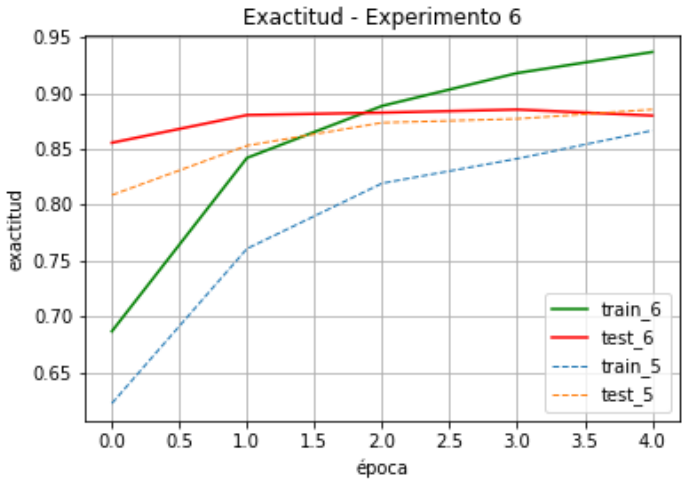

Figura 19. Exactitud experimento 6, tamaño word2vec: 100, train: 93,67%, test: 87,98%. Fuente: propia.

Al aumentar el tamaño del W2V a 100 (Tabla 13) se obtuvo una exactitud de 93,67% sobre los datos de *train*, y de **87,98%** sobre los datos de *test (Figura 19)*. La exactitud sobre los datos de test disminuyó en 0,55% frente al experimento 5, y la exactitud sobre los datos de train aumento en 7,02%. Se presentó sobreajuste luego de la 2° época (Figura 19). Se observa que aumentar el tamaño de word2vec a 100 no es una buena elección, ya que disminuye la exactitud sobre los datos de test, e introduce sobreajuste, por lo cual se descarta.

Según la matriz de confusión (Tabla 14), el porcentaje de oraciones negativas predichas correctamente fue de 86,53% y el de oraciones positivas predichas correctamente fue de 89,15%, siendo valores cercanos y confirmando que no hay sesgos hacia alguna polaridad. Además, el F-Measure fue de 89,15%.

7.7 **Experimento 7.** Se dejan 5 Unidades *GRU* por capa *GRU*.

Tabla 15. Hiperparámetros y datos relevantes Experimento 7, 5 unidades GRU por capa.

#	GRU 1	Dropout	GRU 2	dropout	GRU 3	Dropout	AC Train
7	**5**	0,8	**5**	0,8	**5**	0,8	85,63%
	W2V	**Épocas**	**train**	**test**	**filtrado**	**Adam**	
	8	5	24.722	24.722	símbolos	lr = 3	**Test**
	Parámetros		**Tiempo train**		**RAM**		88,0%
	400.591		2min 44s		1,2/12,72 GB		

Tabla 16. Matriz de Confusión, Experimento 7.

Experimento	7	Predicho	
		Negativo	**Positivo**
Real	**Negativo**	9876	1142
	Positivo	1825	11879

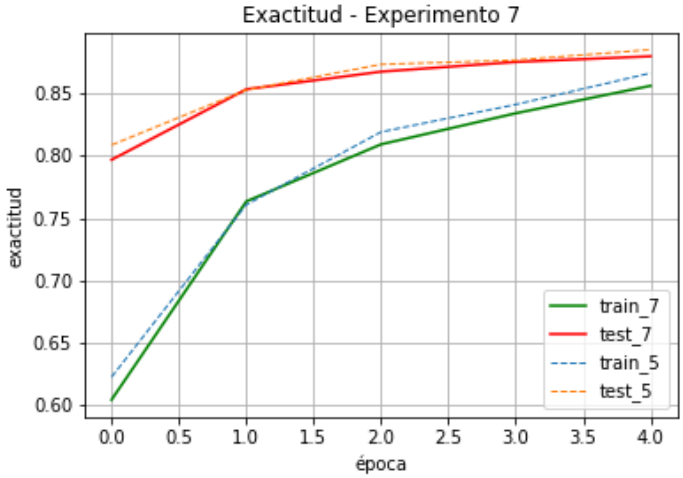

Figura 20. Exactitud experimento 7, 5 unidades GRU por capa, train: 85,63%, test: 88,0%. Fuente: propia.

El experimento 7 (Tabla 15), se compara con el experimento 5 (Tabla 11), ya que se utilizan los mismos hiperparámetros, exceptuando el número de unidades GRU con lo cual se prueba en este experimento.

Al fijar 5 unidades GRU por capa (Tabla 15) se obtuvo una exactitud de 85,63% sobre los datos de *train*, y de **88,0%** sobre los datos de *test (Figura 20)*. La exactitud sobre los datos de test disminuyo en 0,53%, y la exactitud sobre los datos de *train* disminuyó en 1,02% con respecto al experimento 5. Modificando el número de unidades GRU por capa, disminuye la exactitud del sistema, por ese motivo se considera el Experimento 5 como el mejor hasta el momento.

Según la matriz de confusión (Tabla 16), el porcentaje de oraciones negativas predichas correctamente fue de 89,64% y el de oraciones positivas predichas correctamente fue de 86,68%, siendo valores cercanos y confirmando que no hay sesgos hacia alguna polaridad. Además, el F-Measure fue de 87,58%.

Experimento 8. Se utiliza el 80% de ejemplos para *train* y 20% para *test.*

Tabla 17. Hiperparámetros y datos relevantes Experimento 8, 80% del corpus para train, y 20% para test.

#	GRU 1	Dropout	GRU 2	dropout	GRU 3	Dropout	AC Train
8	16	0,8	8	0,8	4	0,8	87,83%
	W2V	**Épocas**	**train**	**test**	**filtrado**	**Adam**	
	8	5	**39.555**	**9.889**	símbolos	lr = 3	**Test**
	Parámetros		**Tiempo train**		**RAM**		89,31%
	402.045		3min 58s		1,2/12,72 GB		

Tabla 18. Matriz de Confusión, Experimento 8.

Experimento	8	Predicho	
		Negativo	Positivo
Real	**Negativo**	3741	643
	Positivo	414	5091

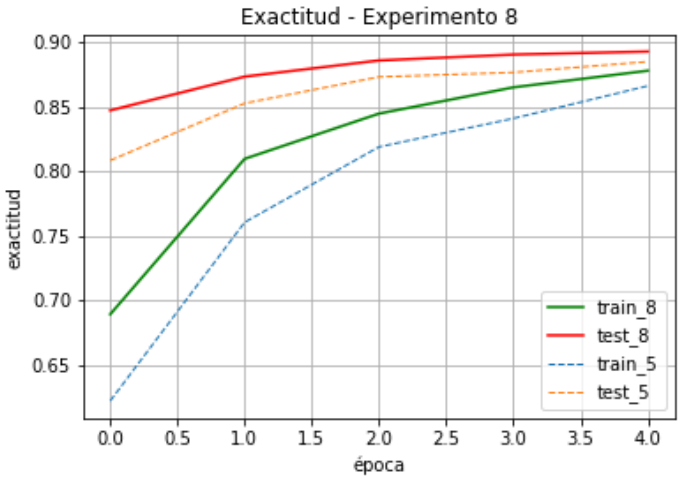

Figura 21. Exactitud experimento 8, 80% del corpus para train, 20% para test. Exactitud train: 87,83%, test: 89,31%. Fuente: propia.

Al utilizar el 20% de los datos para *test*, el número de oraciones para este pasó a ser 9.889, con 4348 oraciones positivas, y 5505 negativas, representando el 44,13% y 55,87 respectivamente.

El experimento 8 (Tabla 17) se compara con el experimento 5 (Tabla 11), ya que se utilizan los mismos hiperparámetros, excepto que en este se utiliza el 80% de datos para *train*, y el 20% para *test*. Se obtuvo una Exactitud de 87,83% sobre los datos de *train*, y de **89,31%** sobre los datos de *test (Figura 21)*. La Exactitud sobre los datos de *test* aumentó en 0,78%, y la exactitud sobre los datos de *train* aumentó en 1,18% con respecto al experimento 5.

Según la matriz de confusión (Tabla 18), el porcentaje de oraciones negativas predichas correctamente fue de 85,33 % y el de oraciones positivas predichas correctamente fue de 92,48%, siendo valores cercanos y confirmando que no hay sesgos hacia alguna polaridad. Además, el F-Measure fue de 91,72%.

7.8 **Experimento 9.** Se filtran tildes, *hashtags*, *links* y *usernames*.

Se procede a filtrar el corpus en:

1. Eliminar tildes.
2. Remplazar los *links*, *hashtags* y usuarios por etiquetas de <dominio>, <hashtag> y <username> respectivamente.

Tabla 19. Hiperparámetros y datos relevantes Experimento 8, filtro de etiquetas y tildes.

Tabla 20. Hiperparámetros y datos relevantes Experimento 9. Se filtran tildes, hashtags, links *y* usernames.

#	GRU 1	Dropout	GRU 2	dropout	GRU 3	Dropout	AC Train
9	16	0,8	8	0,8	4	0,8	87,28%
	W2V	**Épocas**	**train**	**test**	**filtrado**	**Adam**	
	8	5	39.555	9.889	Símbolos, **Etiquetas y tildes**	lr = 3	**Test**
	Parámetros		**Tiempo train**		**RAM**		89,03%
	402.045		4min		1,2/12,72 GB		

Tabla 21. Matriz de Confusión, Experimento 8.

Experimento	9	**Predicho**	
		Negativo	**Positivo**
Real	**Negativo**	3741	643
	Positivo	414	5091

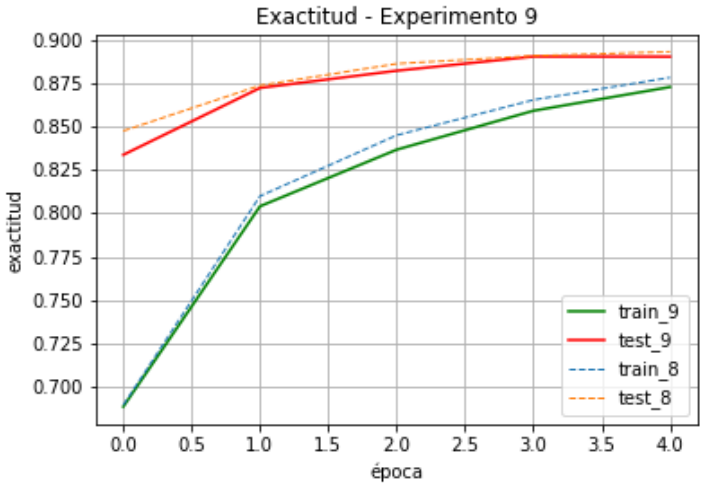

Figura 22. Experimento 9, filtro de etiquetas y tildes. Exactitud train: 87,28%, test: 89,03%. Fuente: propia.

Al filtrar las tildes y etiquetas (Tabla 19), se obtuvo una exactitud de 87,28% sobre los datos de *train*, y **89,03%** sobre los datos de test (Figura 22). La exactitud sobre los datos de *test* disminuyó 0,28% y sobre los datos de *train* disminuyó 0,55%, con respecto al experimento 8 (Tabla 17), lo que sugiere que es mejor cuando no se filtran las etiquetas, ni tildes. Sin embargo, este resultado no es esperado, ya que se supone que cuando se filtran los datos deben dar mejores resultados.

Según la matriz de confusión (Tabla 20), el porcentaje de oraciones negativas predichas correctamente fue de 85,54% y el de oraciones positivas predichas correctamente fue de 91,81%, siendo valores cercanos y confirmando que no hay sesgos hacia alguna polaridad. Además, el F-Measure fue de 91,29%.

7.9 **Experimento 10.** Se implementa *KFold Cross Validation*, para corroborar los resultados.

Para asegurar la validez de los experimentos 8 y 9 (Tabla 17 y 19 respectivamente), se procede a implementar *kfold cross validation*. Se dividen los datos en 5 partes, donde 4 se utilizan para *train* y 1 para *test* (80% y 20% respectivamente), se entrena el modelo y se calcula su exactitud, este procedimiento se realiza 5 veces con partes distintas del corpus elegidas de forma aleatoria. Se realiza el mismo procedimiento para los dos experimentos, con los mismos parámetros respectivamente:

Tabla 22. Comparación de la exactitud entre el experimento 8 y 9, mediante KFold Cross Validation.

	Exp 8 AC test	Exp 9 AC test	diferencia E9 - E8
fold1	80,00%	79,65%	-0,34%
fold2	88,85%	88,11%	-0,74%
fold3	90,22%	89,97%	-0,25%
fold4	90,58%	89,88%	-0,70%
fold5	89,16%	89,16%	0,00%
promedio	87,76%	87,35%	-0,41%

Como se observa en la Tabla 21, realizando la comparación mediante *kfold cross validation* de los experimentos 8 donde no se filtraron etiquetas ni tildes y el experimento 9 donde se si se filtraron, en promedio el experimento 8 obtiene 0,41% más de exactitud, lo que confirma que sin filtro se obtienen mejores resultados, lo cual sigue siendo un resultado no esperado.

Puede ser que el modelo esté interpretando la polaridad de sentimiento de acuerdo a las etiquetas, y como el corpus de TASS contiene entre tantos, tuits de políticos, y personas famosas, puede ser que aparezca varias veces un mismo usuario o hashtags, entonces el algoritmo aprendería a reconocer que usuarios generalmente evocan sentimientos positivos o negativos, igualmente con los *hashtags*. También, las palabras con tilde y sin tilde podrían influir en la polaridad de sentimiento, sin embargo, no parece tan razonable. Por eso se procede a experimentar con estas hipótesis, remplazando en primer lugar solo las tildes y dejando las etiquetas.

7.10 **Experimento 11.** Se remplazan solo las tildes.

Tabla 23. Hiperparámetros y datos relevantes Experimento 11, filtro de tildes.

#	GRU 1	Dropout	GRU 2	dropout	GRU 3	Dropout	AC Train
11	16	0,8	8	0,8	4	0,8	87,50%
	W2V	**Épocas**	**train**	**test**	**filtrado**	**Adam**	
	8	5	39.555	9.889	símbolos y **tildes**	lr = 3	**Test**
	Parámetros		**Tiempo train**		**RAM**		89,80%
	402.045		3min 44s		1,09/12,72 GB		

Tabla 24. Matriz de Confusión, Experimento 11.

Experimento	11	Predicho	
		Negativo	Positivo
Real	Negativo	3974	410
	Positivo	599	4906

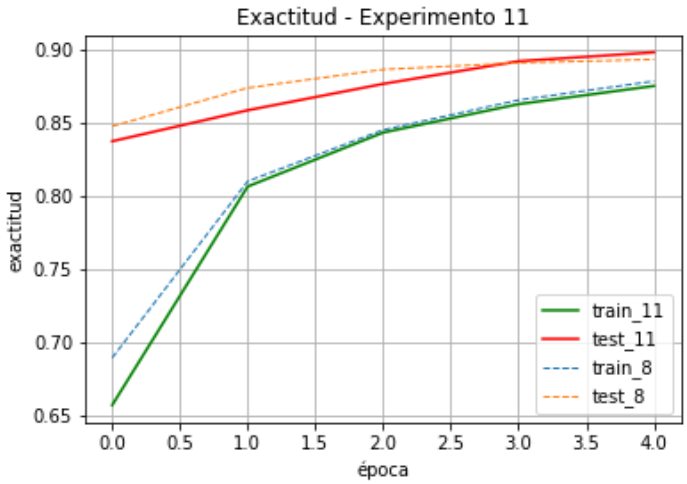

Figura 23. Experimento 11, filtro solo de tildes y símbolos. Exactitud train: 87,50%, test: 89,80. Fuente: propia.

Este experimento (Tabla 22) en el que se filtran solo símbolos y tildes, comparándolo con el experimento 8 (Tabla 17) en el que se filtran símbolos, tildes y etiquetas, obtuvo una exactitud de 87,50% sobre los datos de *train*, y de **89,8%** sobre los datos de *test (Figura 23)*.

La exactitud sobre los datos de *test* aumentó en 0,49 %, y la exactitud sobre los datos de *train* disminuyo 0,33% con respecto al experimento 8. **Se obtiene una mejor exactitud sobre los datos de *test***. Entonces, cuando no se filtran tildes ni etiquetas (experimento 8), se obtienen mejores resultados que al filtrarlas (experimento 9). Cuando se filtran solo tildes (experimento 11) y no etiquetas se obtienen mejores resultados.

Según la matriz de confusión (Tabla 23), el porcentaje de oraciones negativas predichas correctamente fue de 90,65 % y el de oraciones positivas predichas correctamente fue de 89,12%, siendo valores cercanos y confirmando que no hay sesgos hacia alguna polaridad. Además, el F-Measure fue de 89,74%.

7.11 **Experimento 12,** se remplazan solo etiquetas.

Tabla 25. Hiperparámetros y datos relevantes Experimento 12, filtro de etiquetas.

#	GRU 1	Dropout	GRU 2	dropout	GRU 3	Dropout	AC Train
12	16	0,8	8	0,8	4	0,8	86,99%
	W2V	**Épocas**	**train**	**test**	**filtrado**	**Adam**	
	8	5	39.555	9.889	símbolos y **etiquetas**	lr = 3	**Test**
	Parámetros		**Tiempo train**		**RAM**		88,88%
	402.045		3min 44s		1,09/12,72 GB		

Tabla 26. Matriz de Confusión, Experimento 12.

Experimento	12	**Predicho**	
		Negativo	**Positivo**
Real	**Negativo**	3770	614
	Positivo	486	5019

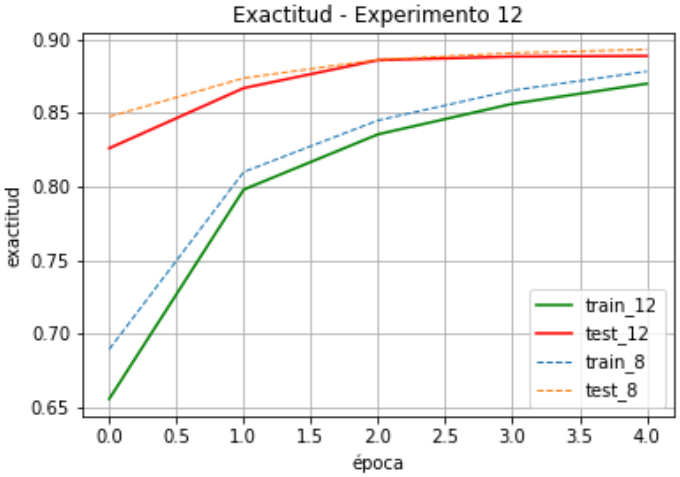

Figura 24. Experimento 12, filtro solo de etiquetas y símbolos. Exactitud train: 86,99, test: 88,88. Fuente: propia.

Este experimento (Tabla 24) consistió en filtrar solo las etiquetas y símbolos, se obtuvo una exactitud sobre los datos de train de 86,99% y sobre los de test de **88,88%** (Figura 24). La exactitud disminuyó 0,43% en *test* y 0,84% en *train*, frente al experimento 8 (Tabla 22). Lo que demuestra que no filtrar los datos es mejor que filtrar solo las etiquetas y símbolos. Además, en relación al anterior experimento 11 donde solo se filtran tildes y símbolos, y se dejan etiquetas, la exactitud sobre los datos de test disminuyó 0,92%.

Se puede observar que el mejor resultado es el experimento 11, donde se filtran tildes y símbolos, seguido del 8 donde no se filtran tildes ni etiquetas, seguido de este experimento en el cual se filtran solo etiquetas. **Lo que confirma la hipótesis de que las etiquetas influyen en la predicción de la polaridad de sentimiento**, es decir que probablemente debido a que el corpus fue adquirido de comentarios de Twitter de políticos y famosos, **hay usuarios que aparecen varias veces**, y estos usuarios pueden representar un peso en la polaridad de sentimiento, como por ejemplo un tuit acerca de un político que generalmente evoca sentimientos negativos, igualmente con los o hashtags, o links.

Finalmente, según la matriz de confusión (Tabla 25), el porcentaje de oraciones negativas predichas correctamente fue de 85,99 % y el de oraciones positivas predichas correctamente fue de 91,17%, siendo valores cercanos y confirmando que no hay sesgos hacia alguna polaridad. Además, el F-Measure fue de 90,75%.

8. CONCLUSIONES Y TRABAJO FUTURO

El experimento con la mayor exactitud fue el 11 (Tabla 22), con una exactitud sobre los datos de test de 89,8%. Los hiperparámetros principales que se modificaron para obtener esta exactitud fueron un *Dropout* de 0,8 para las 3 capas GRU. 80% del corpus para *train*, y 20% para *test*, con un total de 49.444 ejemplos. 5 épocas para entrenamiento. Longitud de 8 para cada representación vectorial de las palabras (W2V o *embeddig*). 16, 8 y 4 unidades *GRU* para cada capa de *GRUs* respectivamente. Con filtro de símbolos y tildes.

Este resultado, está muy cerca de los resultados conseguidos con técnicas de *Machine Learning*, por ejemplo, 9,3% por encima de [10] en donde obtuvieron 80,5% de exactitud. 2,8% por debajo de [11] donde obtuvieron 92,6%. También de trabajos realizados con *Deep Learning* con *RNN* y *LSTM* como [14], en el que alcanzaron una Exactitud del 87,53%, es decir 2,37% por debajo. Finalmente, frente a [16] donde realizaron análisis de polaridad de sentimiento y utilizaron *CNN,* y midieron respecto a F-Measure, obteniendo 88,7% F, es decir 3,02% por debajo del mejor resultado obtenido en el Experimento 8 donde se obtuvo 91,72% F.

Se venía presentando un sobreajuste, en los experimentos 1, 2, 3, 4 que al aumentar el *Dropout* hasta 0,8 y pasar de entrenar con 10 a 5 épocas, desapareció. De lo cual se concluye que, al entrenar demasiado el modelo, se entrena muy bien con los datos de *train*, pero se vuelve malo generalizando con datos que no ha visto antes como los de *test*. Además, se observa que en general a medida que mejora la exactitud con los datos de *test*, desmejora con los datos de entrenamiento.

El mayor tiempo de entrenamiento fue de 5:48 min en el experimento 3 y 6, el primero con un *Dropout* de 0,7, 10 épocas y 402.045 parámetros, y el segundo con un tamaño de *word2vec* de 100, 5 épocas y 5’006.461 parámetros. Se observa que, entre más épocas, y numero de parámetros, mayor tiempo de entrenamiento. Sin embargo, son tiempos bajos en comparación con otros modelos que procesan imágenes o video, ya que las palabras en *RNN* son datos bidimensionales, y por lo tanto más livianos de procesar.

Al realizar *Kfold cross validation* (experimento 10), se corrobora que sin filtrar o remplazar tildes, y etiquetas (*hashtags*, *links* y *usernames)*, se obtienen mejores resultados de exactitud, que cuando se filtran. Al filtrar las tildes (experimento 11), se evita redundancia y que el sistema interprete por separado su versión con tilde y sin tilde, lo cual mejora la exactitud, siendo el mejor resultado. Cuando se filtran solo etiquetas (experimento 12) disminuye la exactitud, ya que estás ayudan al sistema

en su predicción, debido a que el corpus de TASS contiene etiquetas repetidas de famosos, políticos y entre otros, entonces el sistema aprende que cuando aparece un *username*, *hashtag* o *link* especifico, evoca una polaridad de sentimiento similar. Como podría ser los comentarios acerca de un político, donde mayoritariamente las opiniones son negativas, o un hashtag que hace alusión a un tema controvertido.

Sin embargo, el modelo del experimento 11 que obtuvo la mejor exactitud, no se considera la mejor opción para analizar nuevos datos de Twitter u otros dominios, ya que está sesgado. En cambio, el modelo del experimento 9 (Tabla 19) en el que también se filtran las etiquetas, y que obtuvo una exactitud de 89,03% sobre los datos de *test* (0,77% por debajo del 11) podría ser más idóneo para un fin más general.

Debido a que hay pocos *corpus* y trabajos en español de análisis de sentimiento como se nombra en [6], y al contrario en inglés hay bastante literatura, y *corpus* como IMBD [15], o Yelp utilizado en [11], se considera este trabajo como un aporte muy importante a la investigación y desarrollo de esta área en lenguaje español.

Se sugiere como posibles trabajos futuros para los interesados en continuar por esta área, en agregar a la predicción de polaridad de sentimiento la clasificación Neutro, que también aparece etiquetada en el corpus de TASS. Otra opción interesante es agregar un rango más amplio de emociones, no solo polaridad. Incluso es posible pensar en analizar aspectos específicos dentro de las oraciones. Otra buena opción a explorar es la implementación y utilizar modelos basados en Transformers [20] no solo en análisis de sentimientos, sino también en otras tareas como traducción de textos, respuesta de preguntes, entre otras, explorar por ejemplo *BERT* [21]*, GPT-3* [23], pero con el reto de que requieren cantidades muy grandes de datos y no están configurados en español. Finalmente, aplicaciones como predecir el mercado de valores basados en las emociones de los comentarios acerca de las acciones de las empresas en redes sociales [7].

BIBLIOGRAFÍA

[1] V. Kharde y P. S. Sonawane, «Sentiment Analysis of Twitter Data: A Survey of Techniques,» *International Journal of Computer Aplications,* vol. 139(11), pp. 5-15, 2016.

[2] M. Atique y . H. P. Patil , «Sentiment Analysis for Social Media: A Survey,» de *2nd International Conference on Information Science and Security (ICISS)*, Seoul, 2015.

[3] Y. LeCun, Y. Bengio y G. Hinton, «Deep Learning,» *NATURE,* vol. 521, pp. 436-444, 2015.

[4] I. Goodfellow, Y. Bengio y A. Courville, Deep Learning, MIT Press, 2016.

[5] J. Chung, C. Gulcehre, K. Cho y Y. Bengio, «Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,» de *NIPS Deep Learning and Representation Learning Workshop*, 2014.

[6] E. Martínez-Cámara y J. Villena-Román, «TASS - Workshop on Sentiment Analysis at SEPLN,» *Procesamiento del Lenguaje Natural,* vol. 50, pp. 37-44, 2013.

[7] L. Rosenberg y N. Pescetelli , «Artificial Swarm Intelligence Amplifies Accuracy when Predicting Financial Markets,» de *IEEE Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, New York City, 2017.

[8] S. Al-Saqqa, H. Abdel-Nabi y A. Awajan, «A Survey of Textual Emotion Detection,» de *2018 8th International Conference on Computer Science and Information Technology (CSIT)*, Amman, Jordan, 2018.

[9] H. S. Kisan, H. A. Kisan y A. P. Suresh , «Collective intelligence & sentimental analysis of twitter data by using StandfordNLP libraries with software as a service (SaaS),» de *IEEE international Conference on Computational Intelligence and Computing Research (ICCIC)*, Chennai, 2016.

[10] K. Zvarevashe y O. O. Olugbara , «A framework for sentiment analysis with opinion mining of hotel reviews,» de *2018 Conference on Information Communications Technology and Society (ICTAS)*, Durban, South Africa, 2018.

[11] A. Salinca, «Business Reviews Classification Using Sentiment Analysis,» de *17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, Timisoara, Romania, 2015.

[12] the sckit-learn comunity, «sckit-learn,» 2010. [En línea]. Available: https://scikit-learn.org/stable/index.html. [Último acceso: 10 May 2019].

[13] H. Kaur, V. Mangat y Nidhi , «A survey of sentiment analysis techniques,» de *International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, India, 2017.

[14] A. Patel y K. T. Arvind , «Sentiment Analysis by using Recurrent Neural Network,» de *2° Internacional conference on advanced computing software engineering (ICACSE-2019)*, 2019.

[15] «IMBD Datasets,» [En línea]. Available: https://www.imdb.com/interfaces/. [Último acceso: 10 May 2020].

[16] M. A. Paredes-Valverde, R. Colomo-Palacios, M. d. P. Salas-Zárate y R. Valencia-García, «Sentiment Analysis in Spanish for Improvement of Products and Services: A Deep Learning Approach,» *Hindawi, Scientific Programming,* vol. Volume 2017, 26 Oct 2017.

[17] A. Radford, R. Jozefowicz y I. Sutskever, «Learning to Generate Reviews and Discovering Sentiment,» *arVix preprint arXiv:1704.01444v2 [cs.LG],* 6 Apr 2017.

[18] J.̃. Paulo Aires, C. Padilha, C. Quevedo y F. Meneguzzi, «A Deep Learning Approach to Classify Aspect-Level Sentiment using Small Datasets,» de *International Joint Conference on Neuronal Networks (IJCNN)*, Rio de Janeiro, Brazil, 2018.

[19] S. Chen, C. Peng y L. Cai, «A Deep Neural Network Model for Target-based Sentiment Analysis,» de *International Joint Conference on Neural Networks (IJCNN)*, Rio de Janeiro, Brazil, 2018.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, G. Aidan N. , L. Kaiser y I. Polosukhin, «Attention Is All You Need,» *arXiv preprint arXiv:1706.03762v5 [cs.CL],* 6 Dec 2017.

[21] «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,» *arVix preprint arXiv:1810.04805v2 [cs.CL],* 24 May 2019.

[22] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov y Q. V. Le, «XLNet: Generalized Autoregressive Pretraining for Language Understanding,» *arVix preprinted arXiv:1906.08237v2 [cs.CL],* 2 Jan 2020.

[23] T. B. Brown∗, B. Mann, N. Ryder∗ y M. Subbiah∗, «Language Models are Few-Shot Learners,» *arVix preprint arXiv:2005.14165v4 [cs.CL],* 22 Jul 2020.

[24] L. You y B. Tuncer, «Exploring public sentiments for livable places based on a crowdcalibrated sentimet analysis mechanism,» de *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, San Francisco, CA, 2016.

[25] Q. Song, S. Almahdi y S. Y. Yang, «Entropy based measure sentiment analysis in the financial market,» de *IEEE Symposium Series on Computational Intelligence (SSCI)*, Honolulu, HI, 2017.

[26] C. W. Park y D. R. Seo, «Sentiment analysis of Twitter corpus related to artificial intelligence assistants,» de *5th International Conference on Industrial Engineering and Applications (ICIEA)*, Singapore, 2018.

[27] X. Yu, Y. Liu , X. Huang y A. An, «Mining Online Reviews for Predicting Sales Performance: A Case Study in the Movie Domain,» vol. 24, nº 4, pp. 720 - 734, 2010.

[28] N. Buduma, Fundamentals of Deep Learning, First Edition ed., Sebastopol., CA 95472.: O'Reilly Media, Inc., 2017, pp. 1-37.

[29] Y. Bengio, A. Courville y P. Vincent, «Representation Learning: A Review and New Perspectives,» *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 35, nº 8, pp. 1798-1828, Agosto 2013.

[30] J. Schmidhuber, «Deep learning in neural networks: An overview,» *Neural networks,* vol. 61, pp. 85-117, 2015.

[31] A. Graves, Supervised Sequence Labelling with Recurrent Neural Networks, vol. 385, New York: Springe, 2012.

[32] F. Provost, T. Fawcett y R. Kohavi, «The Case against Accuracy Estimation for Comparing Induction Algorithms,» de *Fiftteen Internacional Conference on Machine Learning (IMLC)*, Madison, Wi, 1998.

[33] N. Chinchor, Ph.D, «MUC-4 EVALUATION METRICS,» de *Fourth Message Uunderstanding Conference (MUC-4): Proceedings of a Conference Held in McLean*, Virginia, June 16-18, 1992.

[34] «scikit-learn, k-fold cross validation,» [En línea]. Available: http://scikit-learn.sourceforge.net/stable/modules/generated/sklearn.cross_validation.KFold.html. [Último acceso: 15 Jul 2020].

[35] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, «Dropout: A Simple Way to Prevent Neural Networks from,» *Journal of Machine Learning Research,* vol. 15, pp. 1929-1958, 2014.

[36] Google, «tensorflow,» [En línea]. Available: https://www.tensorflow.org/guide/. [Último acceso: 17 Abril 2020].

[37] F. Chollet, «keras,» [En línea]. Available: https://keras.io/. [Último acceso: 17 04 2020].

[38] «https://www.python.org/,» [En línea]. [Último acceso: 15 Feb 2020].

[39] Google, «tensorflow keras,» [En línea]. Available: https://www.tensorflow.org/guide/keras/overview. [Último acceso: 10 Abril 2020].

[40] Google, «Google Colaboratory,» 25 May 2020. [En línea]. Available: https://colab.research.google.com/notebooks/intro.ipynb.