Análisis de sentimientos con tweets

Elena Villalobos
Instituto Tecnológico Autónomo de México
villaele14@gmail.com

Carolina Acosta
Instituto Tecnológico Autónomo de México
carolina.acostatovany@gmail.com

Edgar Bazo
Instituto Tecnológico Autónomo de México
ing.edbaz@gmail.com

RESUMEN

Una de las redes sociales más utilizadas para interactuar socialmente es la plataforma de *Twitter*. El objetivo del presente trabajo es analizar el texto contenido en *tweets* y construir una red neuronal que clasifique si el sentimiento del mismo es positivo o negativo. Se utilizó una base de datos que contiene 1.6 millones de *tweets*, con su respectiva clasificación. El modelo de redes neuronales creado se basó en el campo de estudio de Procesamiento de Lenguaje Natural. Se encontró un modelo que nos da resultados relativamente buenos, generando una clasificación adecuada del contenido de los *tweets*. Dicho modelo utiliza capas de *embedding* al inicio, redes de memoria a corto plazo y pocas capas ocultas.

Formato de Referencia: Villalobos, E., Acosta, C., & Bazo, E. (2021). Análisis de sentimientos con tweets.
Disponibilidad de artefactos: El código fuente, los notebooks de trabajo, y otros recursos para llevar a cabo este proyecto están disponibles en: https://github.com/ElenaVillano/sentiment_analysis_tweets.
Palabras clave: Aprendizaje profundo, análisis de sentimientos, minería de texto, LSTM, redes neuronales, Python, redes sociales, Twitter.

1. INTRODUCCIÓN

Hoy en día, las redes sociales son un medio de comunicación muy utilizado para conocer las opiniones actuales sobre diversos temas. Una primera aproximación para conocer y/o describir los procesos de comunicación en estas redes, es saber si los comentarios u opiniones tienen alguna connotación positiva o negativa. Una de las redes sociales más utilizadas para monitorear la conversación pública es *Twitter*; por lo que, nos vamos a enfocar a estudiar dicha red social. Específicamente, el objetivo del presente proyecto es analizar el contenido de *tweets* y poder construir un clasificador que aprenda a distinguir entre una opinión positiva o negativa.

Los datos que utilizaremos fueron recolectados de *Twitter*, por la empresa H20 [1], acerca del Huracán Harvey que tenían como intención identificar *tweets* con diferente connotación de sentimientos. El conjunto de datos tiene las siguientes características:

- Idioma: Inglés.
- Observaciones: 1.6 millones.
- Variables:
  - **`target:`** Polaridad del tweet, positivo o negativo.
  - `ids:` ID tweet.
  - `date:` Fecha y hora del tweet.
  - `flag:` Si hubo algún tipo de *query*.
  - `user:` Usuario del tweet
  - **`text:`** Texto del tweet

La variable que utilizamos para entrenar es `text`, que contiene sólo el *tweet* sin *emojis*; y la variable que utilizamos como etiqueta es la `target`.

1.1. Procesamiento de Lenguaje Natural (NLP)

En este proyecto construimos un modelo con base en Procesamiento de Lenguaje Natural (NLP, por sus siglas en inglés) para realizar el análisis de sentimientos. El Procesamiento del Lenguaje Natural es el campo de estudio que se enfoca en la comprensión del lenguaje humano mediante una computadora, siendo así una rama de aprendizaje profundo [4].

2. TRABAJOS RELACIONADOS

Jacob, en *StreamHacker* [2], describe que se enfocó en la clasificación de sentimientos positivos o negativos. Utilizó datos del paquete *NLTK* para el corpus de *reviews* de películas. Empezó utilizando un simple clasificador de *Naive Bayes* como base, y extrajo las variables de forma booleana. Con poca manipulación de los datos logró obtener una exactitud del 73 %, lo cual es cerca de la exactitud humana; se dice que los humanos concuerdan en sentimientos sólo el 80 % del tiempo.

El objetivo del proyecto que realizó Laurent Luce [3] es clasificar un *tweet* positivo o negativo. Utilizó datos que extrajo manualmente, y fueron alrededor de 600 *tweets* positivos y 600 *tweets* negativos para el entrenamiento del clasificador. Utilizó *tweets* sin *hashtags*, sin menciones ni *emojis*. La única imputación que realizó fue eliminar las palabras menores de 2 letras y tener todo en minúsculas. Para crear el clasificador, primero se hizo una extracción de variables para saber cuáles eran las variables relevantes; y el clasificador utilizado fue *Naives Bayes* de *NLTK*. Con sus datos de prueba logró un 80 % de exactitud.

La empresa que desarrolla H20 [1], realizó un proyecto donde el objetivo fue extraer *tweets* relevantes para el caso del huracán Harvey, utilizando el conjunto de datos de `'sentiment140'` que ya se encuentran etiquetados como sentimientos positivos o negativos, y así construir un clasificador que aprenda a diferenciar entre un *tweet* negativo o serio y un *tweet* positivo. El siguiente objetivo fue utilizar este clasificador para *rankear* los *tweets* basados en un porcentaje de severidad y así poder extraer el *top* de *tweets* que necesiten ayuda en la situación actual, en este caso, el huracán Harvey o Irma. Para la transformación de sus datos utilizaron TF-IDF, éste extrae para cada palabra qué tan importante es para ese *tweet*. Después de la transformación, utilizaron *H20 Gradient Boosting* para entrenar y obtener que el clasificador logré diferenciar entre un *tweet* positivo o negativo. No indican cuál es el porcentaje de su exactitud obtenida.

Figura 1: Nube de palabras con etiquetas negativas.

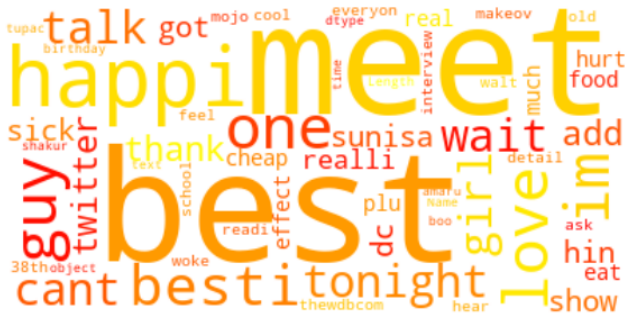

Figura 2: Nube de palabras con etiquetas positivas.

3. ANÁLISIS EXPLORATORIO

Se realizó un análisis exploratorio de las variables `target` y `text`. La primera variable está balanceada; es decir, contiene el 50 % de observaciones positivas y 50 % negativas. Los valores que venían por defecto eran 4 para la asignación positiva y 0 para la asignación negativa. En cuando a la segunda variable, `text`, se muestran gráficos de nubes de palabras, en los cuales el tamaño de la palabra está relacionado a su frecuencia. La figura 1 muestra los *tweets* con etiquetas negativas, y las palabras que más resaltan son: *day, rest, text, aww, bummer, shoulda.* La figura 2 tiene los tweets con etiquetas positivas, donde las palabras que más resaltan son: *meet, best, happi, besti, guy, tonight.* Es importante mencionar que para ambos gráficos de nubes de palabras, se realizó un preprocesamiento de limpieza, descrito a continuación.

4. SOLUCIÓN

4.1. Limpieza de datos

Antes de realizar cualquier tarea de Procesamiento de Lenguaje Natural es importante que nuestros datos tengan cierto grado de limpieza. En el Cuadro 1. mostramos ejemplos de textos procesados por nuestro algoritmo que contiene las siguientes tareas de limpieza:

1. Convertir todo el texto a minúsculas.
2. Quitar:
   a) Caracteres codificados en html (&, ", <, >).
   b) URLs, menciones a usuarios y RT.
   c) # y mantener sólo la palabra o conjunto de palabras.
   d) Espacios o puntos extras.
   e) Caracteres especiales (’¿!,.():;-).
   f) Letras repetidas (*sooooo haaappy*).
   g) Caracteres ascii y Descifrar a utf-8.
3. Separar abreviaciones.
4. Quitar *stopwords.*
5. Realizar *stemming* y lematización.
6. Modificar etiquetas (0 = negativo, 1 = positivo).

Para quitar texto o ciertos caracteres de nuestro texto, utilizamos expresiones regulares, que a continuación presentamos cada una de ellas.

4.1.1. Caracteres HTML. En los *tweets* encontramos caracteres con código HTML que se utilizan con frecuencia, como: &, “, <y >. Cada uno tiene su código en HTML representado por: “&amp”, “"”, “<” y “>”, respectivamente. Y su expresión regular es exactamente el código representado.

4.1.2. URLs. Las *Uniform Resource Locator*, (URL por sus siglas en inglés), son utilizadas en los *tweets* de manera frecuente para indicar la liga a una página web. La expresión regular que corresponde es `((www\.[\S]+)|(https?://[\S]+))`.

4.1.3. Menciones a usuarios. Es común que los usuarios mencionen a otros usuarios, y ésto se realiza con el caracter @. La expresión regular asociada a identificar las menciones es `@[\S]+`.

4.1.4. Retweets. Los *Retweets* son publicaciones duplicadas desde otro usuario y para identificarlas con expresiones regulares utilizamos `{\brt\b}`.

4.1.5. Hasthags. El procesamiento de los *hashtags*, es simplemente eliminar el caracter # y quedarnos con la palabra. Para identificar un *hasthag* con expresiones regulares utilizamos `#(\S)+`.

4.1.6. Letras repetidas. Es común que los usuarios expresen sus sentimientos repitiendo letras como *“I’m sooooo haaaapppyyyy”*, lo cual es variable entre usuarios. Decidimos reducir la repetición a lo máximo dos letras repetidas por letra, ya que en el idioma inglés es común tener letras repetidas en las palabras como *“happy”*. De igual forma, utilizamos expresiones regulares para identificar estos casos y su correspondiente es `(.)\ 1+', r'\ 1\ 1`.

4.1.7. Stopwords. Para nuestra limpieza también removemos las palabras vacías o *stopwords*, que en el lenguaje humano ayudan en la lógica de las oraciones pero no tienen significado en el procesamiento de lenguaje natural, como lo son, los artículos, pronombres, preposiciones, etc. Las *stopwords* las obtenemos de la paquetería *NLTK* [5], utilizando el idioma inglés.

4.1.8. Stemming y lematización. Se refiere a eliminar sufijos morfológicos a una palabra; es decir, reducir la palabra a un “lema”; por ejemplo, *mention*, *mentioning*, *mentioned* son palabras que su lema llega a ser *ment*. Para realizar este proceso utilizamos la paquetería de *NLTK* [5], el tokenizador `word_tokenizer` y el *stem* `PorterStemmer`.

4.1.9. Modificación de etiquetas. Los datos originales contienen una variable `target` que tomamos como etiqueta, y los valores toma son “0” o “4” si son negativos o positivos, respectivamente. Para nuestro problema decidimos cambiarlo a “0” y “1” para negativos y

Cuadro 1: Ejemplo de limpieza de texto

Texto original	Texto limpio
@switchfoot http://twitpic.com/2y1zl - awww, that's a bummer. you shoulda got david carr of third day to do it. ;d	aww bummer shoulda got david carr third day
@RunningGolfer Glad you picked it up...she didn't	lad pick upsh
@i140 I'm excited I made it on your list. Thnx, Jason.	excit made list thnx jason
@johncmayer Where is that Belgian concert you were talking about? I can't even find it on google	belgian concert talk even find googl
sitting on a field with Emma and Miri watching Sarah and Naomi running around searchinf for elves	sit field emma miri watch sarah naomi run around searchinf elv
Couldn't decide if I wanted to go to Family Fortunes on Sunday but train is £45 now Its the Christmas special being filmed on that day!!	could decid want go famili fortun sunday train 45 christma special film day

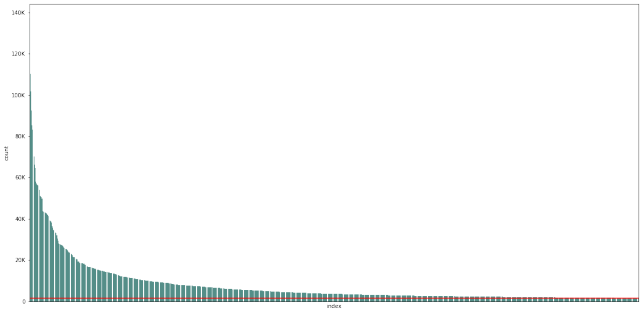

Figura 3: 1K palabras y su frecuencia en nuestro corpus.

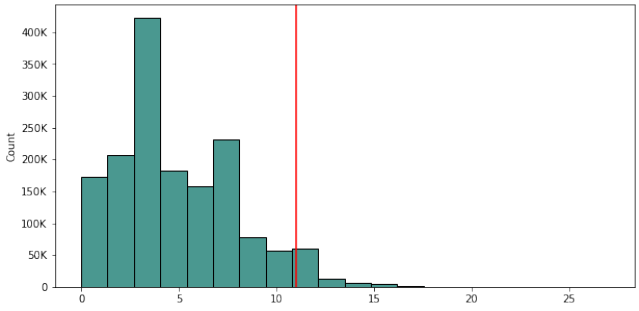

Figura 4: Histograma de número de palabras por oración

positivos, respectivamente; ya que la distancia entre 0 y 1 es más corta que 0 y 4 para un clasificador.

4.2. Tokenizador

Posteriormente a la limpieza de texto, nos enfocamos en hacer otro tipo de pre-procesamiento al texto utilizando un Tokenizador, que se encarga de vectorizar el texto. Es decir, convierte cada palabra en una secuencia de enteros; nosotros utilizamos el Tokenizador de `Tensorflow` [6].

Una vez realizado el proceso de tokenización, obtenemos 330,671 palabras en nuestro diccionario, y para determinar el tamaño máximo de nuestro vocabulario, exploramos la frecuencia de cada palabra (ver figura 3). Seleccionamos aquellas que tuvieran una frecuencia arriba de 1,500, ya que no es necesario agregar palabras a nuestro diccionario que son poco utilizadas por los usuarios. El tamaño final de nuestro vocabulario fue de 1,000 palabras.

Después de obtener el tokenizador con nuestro vocabulario, analizamos la cantidad de tokens que tiene cada oración, (ver figura 4), y obtuvimos la cantidad máxima de tokens que se encuentra en nuestro cuantil al 95 %, que resultó ser de 11 tokens. En otras palabras, el 95 % de nuestros datos tienen 11 tokens o menos.

Entonces, seleccionamos 11 como número máximo de tokens por oración y realizamos un *padding post*, que se refiere a transformar cada secuencia de tokens al mismo tamaño, rellenando el vector al final con ceros para los datos que tenían menos de 11 tokens. El 5 % restante de los datos, no se tomó en cuenta para el modelado. De igual forma utilizamos `Tensorflow` [6] para realizar este procesamiento.

4.3. Arquitectura de la red neuronal

La arquitectura de la red neuronal utilizada se observa en la figura 5. El modelo consta de 4 capas, la primera siendo de entrada y *Embedding*. La segunda es una capa bidireccional *LSTM* con 12 unidades y un *dropout* de 0.5. Las siguiente capa oculta esta formada por 24 neuronas estándar con un *dropout* de 0.5 y una función de activación *ReLu*. Por último, la capa de salida consta de una neurona con función de activación sigmoide.

4.3.1. Capa Embedding. Esta capa se utiliza al principio de la red neuronal debido a que realiza específicamente un procesamiento de texto que asigna un significado similar a la representación numérica de las palabras; en otras palabras, toma en cuenta el contexto relativo de las palabras para representarlas en un espacio numérico. Como parámetros de entrada especificamos el tamaño del vocabulario, la dimensión de nuestra salida y la longitud de las secuencias [4, 7, 8].

4.3.2. Red Long Short Term Memory. Que hace referencia a una red neuronal recurrente (i.e. *RNN*) de memoria a corto plazo (*LSTM* por sus siglas en inglés). Esta capa permite retroalimentación entre neuronas y procesa secuencias en los datos, por lo que puede recordar información previa para decidir los siguientes estados. Esto es muy útil para el procesamiento de texto, pues el lenguaje

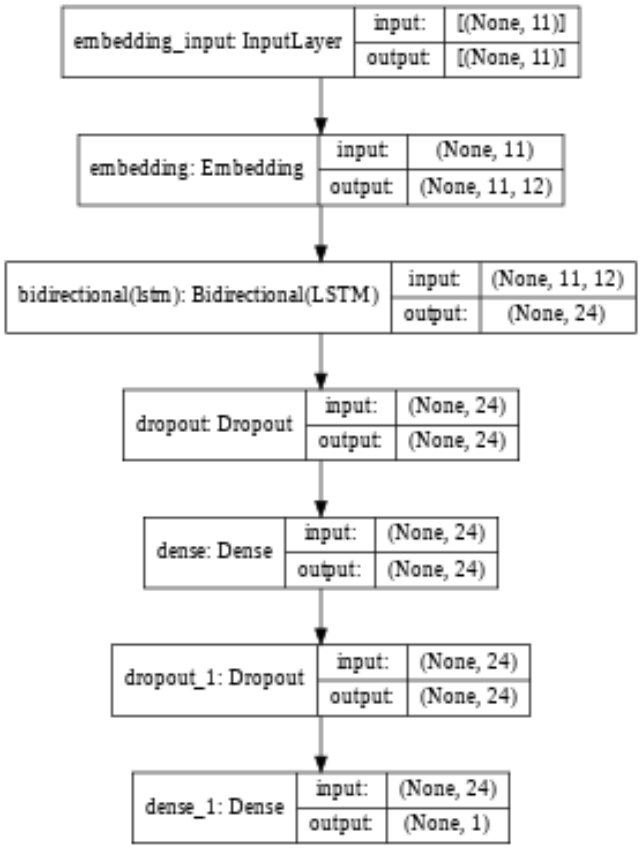

Figura 5: Extracción del diagrama de nuestro modelo en *Tensorflow.Keras*

natural utiliza una estructura similar. Además, esta red soluciona el problema de perder cohesión cuando la red no es capaz de conectar entre dependencias, que sucede cuando las entradas de datos son muy grandes; por lo que funciona como buena alternativa a las *RNN* [4, 9, 10].

4.3.3. Capa bidireccional. Esta es una extensión de la red neuronal recurrente de memoria a corto plazo, que de acuerdo a algunas referencias [4, 11, 12], puede mejorar el desempeño del modelo en problemas secuenciales de clasificación, tal como la estructura de nuestro conjunto de datos.

4.3.4. Funciones de activación y modelo. Es común que en la práctica se utilice la función de activación *ReLU* para capas ocultas [4], debido a que permite a la red neuronal aprender dependencias no lineales.

$$f(x) = max(0, x) = \begin{cases} 0 & \text{si x<0} \\ x & \text{si x>=0} \end{cases} \tag{1}$$

Se consideró la función sigmoide como función de activación para la capa de salida, ya que nuestro problema a resolver consta de una clasificación binaria; 0 si es negativo y 1 si es positivo.

$$f(x) = \frac{1}{1 - \exp^{-x}} \tag{2}$$

4.3.5. Configuración del modelo.

- Optimizador: RMSprop,

$$C = C + d \cdot C \cdot (1 - d) \cdot \Delta W^2 \tag{3}$$

$$W = W - \frac{\alpha \cdot \Delta W}{\sqrt{C + \epsilon}} \tag{4}$$

- Función de pérdida: Binary Cross Entropy,

$$H_q(q) = -\frac{1}{n} \sum_{i=1}^{n} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i)) \tag{5}$$

donde y es la etiqueta y $p(y)$ la probabilidad de que la predicción sea 1.

- Métricas: Binary accuracy y accuracy.

4.3.6. Ajuste del modelo.

- Épocas: 20.
- Tamaño de lote: 64.
- Tamaño de conjunto de validación: 20 %.

Después de la limpieza de texto a nuestros 1.6 millones de datos, obtuvimos 1,591,506 que realmente tenían contenido y éstos los separamos en 70 % para entrenar y 30 % para pruebas. Por lo tanto, para entrenar utilizamos 1,114,054 datos de los cuales 20 % se asignaron para validación.

4.4. Comparaciones de modelos

En el presente proyecto experimentamos con alrededor de 30 modelos con diversas configuraciones e hiper-parámetros. El modelo que mejor se ajustó y predijo mejor los datos fue el presentado en la figura 5. Sin embargo, a continuación presentaremos algunas de las configuraciones probadas más relevantes y diversas reflexiones sobre el uso de las mismas.

- Tamaño de datos: En las primeras iteraciones probamos evaluar los modelos con datos de entre 50K y 200K para reducir el tiempo de cómputo, en dichas iteraciones salieron precisiones de alrededor del 70 %. Al utilizar los 1.6 millones aumentó la precisión en 5 puntos porcentuales, por lo que elegimos trabajar con la base completa.
- Cantidad de palabras: Encontramos que utilizar un número mayor de palabras (10K) mejoraba mucho el ajuste del modelo. Sin embargo, esto a su vez hace que tengamos muchas palabras con frecuencia muy baja, por lo que al final la predicción de estos modelos no era tan buena. Los modelos que tenían menor número de palabras predecían mejor los sentimientos de los *tweets*. Debido a esto, utilizamos 1,000 palabras, de las 330,671 generadas en el diccionario de tokens (ver figura 3).
- Longitud de la secuencia: Cada *tweet* se convierte en un vector de enteros (secuencia) que depende de su mismo número de palabras, probamos modelos con diferente secuencia y encontramos que utilizar la longitud de 11, mejoraba también el desempeño del modelo (ver figura 4).
- *Padding*: El relleno de las secuencias se puede realizar al inicio o al final, observamos que hacerlo al final era mejor que hacerlo al inicio.

- Número de capas: Debido a que estamos utilizando capas recurrentes, no es necesario colocar demasiadas capas para el modelo, por lo que en general probamos modelos con capas de no más de 8.
- Número de unidades en capas: Para este hiper-parámetro utilizamos unidades de 8, 12, 24 o 64; las unidades que mejor funcionaron fueron 12 y 24.
- Función de activación: El desempeño fue muy similar entre los optimizadores RMSprop y Adam, al final, se decidió mantener el optimizador RMSprop.
- Capa bidireccional: El uso de esta capa no se observó que fuera determinante para mejorar el desempeño de los modelos en general. Sin embargo, resultó ser parte del modelo final elegido.
- *Dropouts*: Existieron algunas evaluaciones en los modelos donde la precisión del conjunto de validación era más alto que la precisión del conjunto de entrenamiento. Se encontró que esto se puede deber a varias razones [4, 13, 15], una de ellas es el uso de la técnica de regularización *dropout* en la red, que oculta temporalmente neuronas de la red en el conjunto de entrenamiento, pero no necesariamente en el conjunto de validación. A consecuencia de esto, realizamos iteraciones donde se quitó esta técnica y se encontró que efectivamente, ya no generaba el problema de que la precisión en el conjunto de validación era más grande que en el de entrenamiento.

5. RESULTADOS

Las métricas en los diferentes conjuntos de entrenamientos se pueden observar en el Cuadro 2. En ésta se observa que no hay una diferencia muy grande entre los conjuntos de datos, tanto para la precisión como para la pérdida.

Cuadro 2: Resultados obtenidos en entrenamiento, validación y prueba

	Entrenamiento	Validación	Prueba
Precisión	75.09 %	74.97 %	75.18 %
Pérdida	0.5236	0.5118	0.5102

La figura 6 se muestra la pérdida del modelo, a través de las épocas, para el conjunto de entrenamiento y el de validación. Se observa que en las primeras épocas la pérdida para el conjunto de entrenamiento (línea azul) disminuye en varias unidades, sin embargo, a partir de la época 8 parece incrementar unas cuántas unidades. En cuanto al conjunto de validación (línea naranja), se mantiene a lo largo de todas las épocas por debajo del conjunto de entrenamiento, y parece tener un comportamiento consistente, excepto por el pico observado en alguna de las últimas épocas.

La figura 7 muestra la precisión del modelo a lo largo de las épocas para el conjunto de entrenamiento (línea azul) y el de validación (línea naranja). La precisión de entrenamiento sobrepasa el 75.2 % durante algunas épocas intermedias y la precisión de validación oscila entre 74.9 % y 75 %. La línea naranja siempre se observa por debajo de la azul, y ambas tienen un comportamiento similar; que puede ser un indicador de que nuestro modelo no tiene algún tipo de sobre-entrenamiento o sub-entrenamiento.

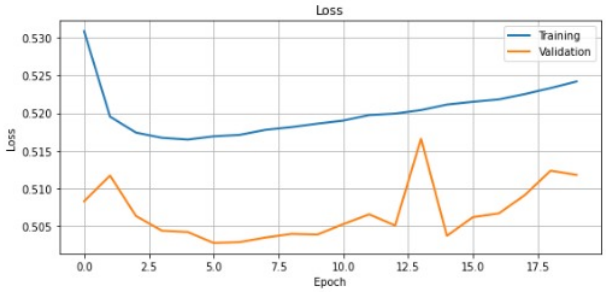

Figura 6: Pérdida del modelo a lo largo de las épocas.

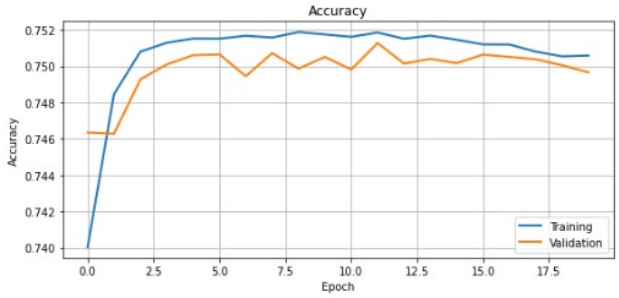

Figura 7: Precisión del modelo a lo largo de las épocas.

Además de tener un conjunto de prueba proveniente de la base de 1.6 millones de observaciones. Obtuvimos una base que tenía 359 *tweets* seleccionados, junto con su etiqueta. Decidimos evaluar la precisión de nuestro modelo con este nuevo conjunto de datos, para analizar de manera más detallada el comportamiento de nuestro modelo. A dicho conjunto, le aplicamos la misma limpieza de datos ya presentada anteriormente y mostramos tres ejemplos en el Cuadro 3. En este cuadro, observamos tres ejemplos, dos donde el modelo acierta correctamente y uno en el que no. Este último parece tener un poco más de ambigüedad que los predichos correctamente, lo cual puede ser la razón por la cuál el modelo se equivocó en predecir dicho texto.

Asimismo en cuanto a las métricas obtenidas en este nuevo conjunto de datos, la precisión obtenida fue del 76.6 %, teniendo 84 *tweets* clasificados erróneamente.

5.1. Análisis de desempeño obtenido y problemas encontrados

La razón por la que se pudo obtener un desempeño "relativamente bueno" recae en diversas razones, una de ellas siendo el uso de pocas capas ocultas y una recurrente con memoria a corto plazo. Como se mencionó anteriormente, una capa recurrente permite retroalimentación entre neuronas y procesa la secuencia que esperamos obtener tomando en cuenta el contexto del texto, que es muy útil para la naturaleza de nuestros datos.

Uno de los problemas encontrados se relaciona con el *tweet*, hay ciertos usuarios que utilizan muchas letras repetidas para expresar sus opiniones y lo que logramos fue reducirlo a dos palabras repetidas, pero nunca a la palabra original. De igual forma, buscar

Cuadro 3: Ejemplo predicciones en la base de prueba adicional.

Texto original	Texto pre-procesado	Etiqueta real	Etiqueta predicha
how can you not love Obama? he makes jokes about himself.	love obama make joke	Positive	Positive
cant sleep... my tooth is aching.	cant sleep tooth ach	Negative	Negative
I just created my first LaTeX file from scratch. That didn't work out very well. (See @amandabittner , it's a great time waster)	creat first latex file scratch work well see great time waster)	Negative	Positive

una base que tenga el texto completamente limpio, sin faltas de ortografía o con completa coherencia, es difícil de alcanzar por la cantidad de palabras y todas las posibles variaciones que usamos los seres humanos al comunicarnos con texto; sobre todo teniendo la libertad de escritura que proporciona la plataforma de *Twitter*.

A pesar de lo anterior, consideramos que una de las posibles razones por la cual obtuvimos resultados relativamente buenos, fue la limpieza del texto. Esto ayudó a generalizar la forma de las palabras en sus diferentes tiempos, quitar palabras sin sentido para el análisis, y quitar fuentes de ruido que pueden afectar la estimación del modelo. Al final, todo esta limpieza generó una reducción en el tamaño del vocabulario y que la frecuencia aumentara para cada una de las secuencias.

Otro problema encontrado fue que a pesar de probar de diferentes configuraciones de hiper-parámetros, hacer un pre-procesamiento de texto detallado, utilizar secuencias de tokens asociadas al tamaño de los *tweets*, e incluso hacer uso de diferentes números y tipos de capas, ningún modelo pudo subir su precisión a más de 75 %. Creemos que esto se puede deber a la base de datos o a la misma característica del lenguaje que tiende a ser muy complejo.

6. CONCLUSIONES

Uno de los principales logros del presente trabajo fue obtener un modelo relativamente bueno y adecuado que obtuvo buenas métricas a través de los diferentes conjuntos de datos, e incluso con un conjunto de datos de prueba adicional. Esto se podría considerar un buen aproximado a lo que un humano puede predecir cuando se le pide que categorice sentimientos positivos o negativos, que éste califica correctamente los sentimientos con un 80 % de precisión, como lo menciona el artículo de Jacob en StreamHacker [2].

Un logro adicional, fue el aprendizaje del comportamiento de modelos de redes neuronales profundas enfocados a evaluar lenguaje. Si bien, la complejidad de la estructura en la comunicación escrita puede ser muy grande, estos modelos nos presentan una buena aproximación a segmentar dicha complejidad y estudiar partes específicas de la misma para ir generando mayor conocimiento en el área.

Además, la limpieza y descripción de estos *tweets*, nos mostró que a pesar de tener una gran variedad de palabras en el idioma inglés, realmente no se utiliza mucha de esta riqueza para expresar sentimientos positivos y/o negativos. Lo cual no es algo malo necesariamente, pues habla de que estos sentimientos pueden ser más sencillos de lo que realmente pensamos. Sin embargo, una siguiente aproximación a este problema podría ser agregar una categoría neutral, pues no necesariamente todos los textos entran sólo en estas dos categorías, y muchas veces tienen una intención de informar más allá de expresar un sentimiento.

En este proyecto el procesamiento de todos los modelos evaluados se utilizaron unigramas (una palabra completa como token) del texto por lo que se podría sugerir que en futuras investigaciones se utilice bigramas (dos palabras como token) pensado como una posible mejora para el modelo.

Finalmente, el presente proyecto se realizó con *tweets* en el idioma inglés, por lo que un camino muy adecuado sería probar este tipo de modelos en el idioma de español. Además, se podrían buscar *tweets* referentes a un sólo tema, como elegir un *hashtag* específico o un acontecimiento social en cierto periodo tiempo, para ayudar al estudio del Procesamiento de Lenguaje Natural.

REFERENCIAS

[1] *Social Machine Learning with H20, Twitter, Python.* [https://www.linkedin.com/pulse/social-machine-learning-h2o-twitter-python-marios-michailidis/].

[2] *Jacob, 2010. StreamHacker. Text classification for sentiment analysis - Naive Bayes Classifier.* [https://streamhacker.com/2010/05/10/text-classification-sentiment-analysis-naive-bayes-classifier/].

[3] *Laurent Luce, 2012. LaurentLuce. Twitter sentiment analysis using Python and NLTK.* [http://www.laurentluce.com/posts/twitter-sentiment-analysis-using-python-and-nltk/].

[4] *Notas del curso de Aprendizaje profundo con el profesor Edgar Roman-Rangel.*

[5] *Documentación NLTK* [https://www.nltk.org/].

[6] *Documentación Tensorflow, keras, pre-procesamiento.* [https://www.tensorflow.org/guide/keras/preprocessing_layers].

[7] *Documentación Keras embedding.* [https://keras.io/api/layers/core_layers/embedding/].

[8] *How to use word Embedding Layers for Deep learning with Keras.* [https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/].

[9] *Documentación keras LSTM layer.* [https://keras.io/api/layers/recurrent_layers/lstm/].

[10] *Implementation of LSTM layers.* [https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47].

[11] *Documentación keras de bidirectional layer.* [https://keras.io/api/layers/recurrent_layers/bidirectional/].

[12] *How to Develop a Bidirectional LSTM for Sequence Classification in Python with Keras.* [https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras].

[13] *Redes Neuronales, "Sanity checks".* [https://cs231n.github.io/neural-networks-3/#sanitycheck].

[14] *Documentación Tersonflow layers dropout.* [https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout].

[15] *What is Dropout?* [https://www.machinecurve.com/index.php/2019/12/16/what-is-dropout-reduce-overfitting-in-your-neural-networks/].