Universidad de Cantabria

Facultad de ciencias

Análisis de sentimiento de textos cortos basado en medidas de centralidad en grafos y aprendizaje profundo

Sentiment analysis of short texts based on centrality measures and deep learning

Informe de trabajo de fin de Grado presentado por

juncal blanco, aitor

como requisito de graduación de la carrera de Ingeniería informática de Facultad de ciencias de la Universidad de Cantabria

Supervisor

palazuelos calderon, camilo

Cantabria, julio 2023

Índice general

1. Introducción **3**
- 1.1. Motivación . . . 3
- 1.2. Objetivos . . . 4
- 1.3. Características deseables de la propuesta . . . 4
- 1.4. Estructura del resto del documento . . . 5

2. Estado del arte **7**
- 2.1. Artículo de Lovera et al. (2021) . . . 7
- 2.2. Artículo de Basiri et al. (2021) . . . 10
- 2.3. Otros artículos . . . 13

3. Modelo propuesto **15**
- 3.1. Preprocesamiento . . . 15
- 3.2. Construcción de los grafos de polaridad . . . 16
- 3.3. Ajuste del modelo de Red neuronal . . . 18

4. Experimentación **21**
- 4.1. Conjuntos de datos utilizados . . . 21
- 4.2. Lectura de los conjuntos de datos y construcción del modelo . . . 24
- 4.3. Cálculo de métricas de rendimiento . . . 25
- 4.4. Selección de parámetros y estudio de ablación . . . 26
- 4.5. Comparación con modelos de lenguaje BERT . . . 26
- 4.6. Interpretabilidad del modelo . . . 27

5. Resultados **29**
- 5.1. Presentación de los resultados con nuestro modelo . . . 29
- 5.2. Selección de parámetros y estudio de ablación . . . 29
- 5.3. Comparación con otros trabajos . . . 30
- 5.4. Análisis de interpretabilidad de nuestro modelo . . . 35

6. Conclusiones y trabajo futuro **39**
- 6.1. Conclusiones . . . 39
- 6.2. Líneas de trabajo futuro . . . 40
  - 6.2.1. Análisis de sentimiento a partir de texto e imágenes . . . 40
  - 6.2.2. Análisis de sentimiento mediante movimientos oculares . . . 41

Anexo **43**

Resumen

El análisis de sentimiento (o la minería de opinión) consiste en detectar información subjetiva en textos mediante modelos de aprendizaje automático y procesamiento del lenguaje natural. Una de las tareas clásicas del análisis de sentimiento es la de la identificación de la polaridad de un texto a través de su clasificación en, al menos, dos categorías (positiva y negativa).

En los últimos años, coincidiendo con el auge de los large-language models pre-entrenados, las propuestas de análisis de sentimiento han tendido hacia la definición de técnicas de caja negra con escasa o nula interpretabilidad. Por este motivo, han empezado a surgir modelos menos opacos que hacen hincapié en la representación semántica del texto como herramienta para ponderar la importancia de cada palabra o cada frase para la clasificación del texto completo.

En el caso de nuestro trabajo, optaremos por un modelo más sencillo e interpretable. Con un modelo así, lograremos, por un lado, que el modelo sea más rápido y permita un mayor número de ejecuciones en un mismo intervalo de tiempo, y, por otro lado, que sea más fácil entender como se construye el modelo y como funciona. De esta manera, tendremos un modelo eficaz y eficiente que no se comportará como una caja negra. Para lograr eso, seguiremos el siguiente proceso de trabajo:

Para comenzar, obtendremos los textos a analizar de un conjunto de datos en formato CSV. Posteriormente, realizaremos un preprocesamiento con dichos textos, logrando así un formato óptimo de los mismos para su entrada en el modelo. Luego, dividiremos los textos en dos conjuntos: Uno de entrenamiento y otro de test.

Primero, con los elementos del conjunto de entrenamiento crearemos un grafo de polaridad negativo con los textos negativos y un grafo de polaridad positivo con los positivos. Con esos grafos entrenaremos un modelo predictor, que tomará como entrada 8 métricas para cada texto, estas métricas se realizarán comparando dichos textos con ambos grafos de polaridad.

Por último, tomaremos el conjunto de test, un conjunto de textos no conocidos por el modelo y trataremos predecir si son negativos o positivos. Lo que nos permitirá evaluar la clasificación del modelo.

PALABRAS CLAVE: Grafos, opinión, modelo, predicción, métricas.

Sentiment analysis (or opinion mining) consists of detecting subjective information in texts by means of machine learning and natural language processing models. One of the classic tasks of sentiment analysis is to identify the polarity of a text by classifying it into at least two categories (positive and negative).

In recent years, coinciding with the rise of pre-trained large-language models,

sentiment analysis proposals have tended towards the definition of black-box techniques with little or no interpretability. For this reason, less opaque models have begun to emerge that emphasize the semantic representation of the text as a tool for weighting the importance of each word or phrase for the classification of the whole text.

In the case of our work, we will opt for a simpler and more interpretable model. With such a model, we will achieve, on the one hand, that the model is faster and allows a greater number of executions in the same time interval, and, on the other hand, that it is easier to understand how the model is built and how it works. In this way, we will have an effective and efficient model that will not behave like a black box. To achieve this, we will follow the following work process:

To begin with, we will obtain the texts to be analyzed from a dataset in CSV format. Subsequently, we will perform a preprocessing with these texts, thus achieving an optimal format for their input into the model. Then, we will divide the texts into two sets: A training set and a test set.

First, with the elements of the training set we will create a negative polarity graph with the negative texts and a positive polarity graph with the positive ones. With these graphs we will train a predictor model, which will take as input 8 metrics for each text, these metrics will be made by comparing these texts with both polarity graphs.

Finally, we will take the test set, a set of texts not known to the model, and try to predict whether they are negative or positive. This will allow us to evaluate the classification of the model.

KEYWORDS: Graphs, opinion, model, prediction, metrics.

Capítulo 1

Introducción

1.1. Motivación

Con el avance de Internet, las redes sociales se han convertido en una parte muy importante de nuestra vida: las usamos tanto para informarnos, como para manifestar nuestras opiniones y emociones sobre diversos temas, siendo esto último en lo que nos vamos a focalizar en este trabajo, haciendo uso de lo que se conoce como minería de opinión. Nigro (2020)

Debido a esto, hoy tenemos ingentes cantidades de datos provenientes de millones de personas a lo largo de todo el mundo. Ejemplo de esto son las opiniones, y es que Internet está lleno de opiniones sobre productos, películas, series, etc. Son datos que leídos en bruto pueden carecen de interpretabilidad y significado, pero que bien analizados pueden tener mucha importancia para las empresas. Poder disponer de miles o millones de opiniones sobre un producto permite medir a las empresas el impacto y el éxito de dichos productos. Además, un buen análisis permite realizar estudios de mercado y obtener ventajas competitivas.Itelligent (2017)

El análisis de sentimiento (o la minería de opinión) consiste en detectar información subjetiva en textos mediante modelos de aprendizaje automático y procesamiento del lenguaje natural. Una de las tareas clásicas del análisis de sentimiento es la de la identificación de la polaridad de un texto a través de su clasificación en, al menos, dos categorías (positiva y negativa). Sabiendo esto, vamos a ver algunas de las ventajas que ofrece el análisis de sentimiento: Itelligent (2017)

1. Permite obtener datos claros, con un formato adecuado para su análisis y estudio.
2. Permite descartar, dentro de la masiva cantidad de datos, datos que no son utiles para el análisis.
3. Permite saber la opinión de un producto determinado, conocer su éxito o fracaso, y actuar en consecuencia en el menor tiempo posible.
4. Permite conocer la reputación de la empresa también de una manera más general, lo que nos ayudará a contrarrestar una crisis de opinión.
5. Permite potenciar la toma de decisiones dentro de la empresa, situación imprescindible para el apartado de marketing.

1.2. Objetivos

El objetivo principal de este TFG es, partiendo de unos datos de entrenamiento etiquetados, entrenar un modelo que posteriormente sea capaz de predecir correctamente el sentimiento de cualquier texto negativo o positivo(en este trabajo se obviarán los textos neutrales).

Sabiendo esto, en nuestro caso, lo que proponemos es un modelo de detección de la polaridad (positiva frente a negativa) de textos cortos con una capacidad predictiva similar a la de large-language models como BERT Devlin et al. (2019), pero más rápido de entrenar a partir de una representación semántica de los textos que clasificar. Para ello, este objetivo general se concreta en tres específicos:

- Proponer una representación semántica del texto a partir de las ideas de Lovera, Cardinale y Homsi Lovera et al. (2021) e implementar el mecanismo de transformación del texto a esta representación en Python.
- Ajustar un modelo predictivo que, a partir de la representación del objetivo específico 1, sea capaz de clasificar textos cortos de acuerdo con su polaridad (positiva frente a negativa).
- Evaluar y comparar el modelo del objetivo específico 2 con el BERT de Google Devlin et al. (2019) y alguna de sus variantes en términos de capacidad predictiva, de coste temporal y de interpretabilidad de los resultados.

1.3. Características deseables de la propuesta

Por otra parte, tal y como hemos mencionado anteriormente, la minería de opinión es un campo de estudio que se encuentra en auge, especialmente en el entorno de Internet, por lo que es lógico pensar que antes de esta investigación, ya antes otras personas e incluso empresas, han estudiado de manera más o menos exhaustiva este campo, ya sea desde un prisma externo como haremos nosotros, o desde un prisma más interno como el que puede tener una empresa que quiera conocer la opinión de sus clientes sobre algún tipo de producto y/o servicio. Por lo que, entendiendo esto, queda claro que otro objetivo importante de nuestro modelo es la diferenciación con otros modelos que han intentado lo mismo o algo parecido. Nuestro modelo va a tener dos ventajas claras respecto a los modelos más sofisticados que se suelen utilizar para la minería de opinión, como por ejemplo los modelos BERT que mencionábamos en el objetivo principal.

- Simplicidad: Nuestro modelo, tal y como explicaremos luego en posteriores secciones, ha de tener una arquitectura sencilla. Lo que permitirá que este sea luego más fácil de comprender internamente y lo más importante, mucho más rápido de entrenar y computar, y aunque no logremos tan excelentes resultados como lograría un modelo mucho más complejo, acercarnos lo suficiente en términos de rendimiento como para que, sumando que nuestro modelo será mucho más rápido, conseguir que el nuestro sea más rentable o al menos sea visto como digno de tener en cuenta respecto a otros más complejos. Un ejemplo de esto puede ser el siguiente:

Si, por un lado, tenemos un modelo que obtiene un 99 % en diversas métricas como puede ser el área bajo la curva(AUC), el *Accuracy* o la *precision*, pero que tarda 5 días en entrenarse, y, por otro lado, tenemos un modelo que si bien no llega al 99 % si se acerca(pongamos un rango entre 95 % y 98 %) y este modelo tarda en entrenar unas pocas horas, para muchos casos nos puede convenir el modelo más simple. Sobre todo, teniendo en cuenta un factor también diferencial, y es que es muchísimo más probable que el modelo utilizado en este trabajo sea utilizable por cualquier ordenador en tiempo razonable, que otros modelos mucho más sofisticados, que muchos ordenadores de andar por casa ni soportarían o tardarían semanas en realizar el cómputo deseado.

- Interpretabilidad: Entroncando con el punto anterior, es un punto a favor de un modelo que este sea fácil de entender y sobre todo fácil de interpretar lo que está ocurriendo internamente, una arquitectura más fácil ayuda a la interpretabilidad, unos inputs más sencillos ayudan a la interpretabilidad, y un output claro también ayudará. Esto nos permitirá, tal y como veremos en el apartado de experimentación, entender como clasifica nuestro modelo y sobre todo conocer el porqué de la dicha clasificación. Y es que hoy en día nos encontramos con que muchos de los modelos más complejos, si bien predicen excelentemente, luego no podemos conocer exactamente porque el modelo ha decidido lo que ha decidido, comportándose así como enormes cajas negras de las que tenemos mucha menos información de la que nos gustaría.

1.4. Estructura del resto del documento

En el capítulo 2, se presenta cuál es el estado del arte en el campo de la minería de opinión en Internet. Es decir, se toman y explican trabajos de otros autores que realizaron trabajos similares a lo que nosotros vamos a presentar. Siendo esto así porque algunos de esos trabajos nos han servido como referencia teórica al comenzar con nuestro proyecto.

En el capítulo 3, se establece cuál es el modelo propuesto para nuestro TFG. Explicando en que otros modelos nos hemos inspirado construirlo, así como el funcionamiento del modelo, dejando claro las entradas, el procesamiento interno y las salidas.

En el capítulo 4, se presenta la experimentación realizada para probar nuestro modelo. Básicamente, en este capítulo explicamos como llevamos a la práctica nuestro modelo propuesto.

En el capítulo 5, una vez realizado nuestro trabajo y creado nuestro modelo, presentaremos los resultados que hemos obtenido. Además, también explicaremos por qué hemos elegido dicho modelo y no otro. Y, posteriormente, compararemos los resultados obtenidos con nuestro modelo con los obtenidos con otros modelos presentados en otros trabajos para medir el rendimiento del nuestro.

En el capítulo 6, se presentan las conclusiones finales obtenidas durante la realización del proyecto. Además, se presentan posibles vías para ampliar nuestro TFG actual.

Por último, restan dos capítulos auxiliares que son: un anexo donde se presenta un enlace al repositorio *Github* donde se aloja el código empleado en nuestra implementación, y, por otro lado, la bibliografía empleada para la realización del trabajo.

Capítulo 2

Estado del arte

Para comenzar, empezaremos por explicar detalladamente los dos artículos que más importantes y útiles nos han sido. Exponiendo los modelos mostrados en dichos artículos. su funcionamiento y los resultados obtenidos.

2.1. Artículo de Lovera et al. (2021)

El primer artículo es *Sentiment Analysis in Twitter Based on Knowledge Graph and Deep Learning Classification* Lovera et al. (2021). El principal objetivo de este trabajo es conseguir un modelo híbrido para el análisis de sentimiento. Este modelo estará basado en la construcción de grafos de conocimiento a partir de un conjunto de tweets, y a partir de ahí la utilización de métricas de similitud de los grafos más técnicas de *Deep Learning* LeCun et al. (2015) para predecir la polaridad positiva o negativa de los tweets.

Ahora, vamos a presentar este artículo dividiendo en 5 apartados el proceso de trabajo que siguen:

1. **Lectura del dataset**
   Realizan la lectura de un dataset llamado Sentiment140, que contiene 1.600.000 de tweets, y hacen con él una división 80 % entrenamiento y 20 % test que no conocemos.

2. **Preprocesamiento**
   Eliminan los caracteres que no aportan a determinar el sentimiento de un tweet. Como pueden ser, los caracteres HTML, menciones(@), enlaces o los hashtags(#). También pasan los tweets a minúsculas, expanden las contracciones del inglés, eliminan las *stopwords* o palabras vacías y además también eliminan aquellas palabras demasiado largas(más de 45 caracteres) así como el tweet al que pertenecía dicha palabra.

3. **Construcción de los grafos**
   En esta fase, los autores primero realizan una segmentación de los tweets en sentencias. Esta segmentación consiste en dividir el tweet en sentencias de tal forma que cada una tiene un sujeto y un predicado unidos por una estructura verbal. Una vez que se tienen todos los tweets divididos en una o más sentencias, para cada una de ella hace un anális *Pos tagging* para obtener las

etiquetas de cada palabra. Dicho análisis, en español conocido como *etiquetado gramatical*, consiste en asignar a cada palabra de la sentencia su categoría gramatical. Luego, obtienen la etiqueta gramatical de cada palabra de la frase, construyen un árbol de parseo con todas ellas como se ve en la figura 2.1

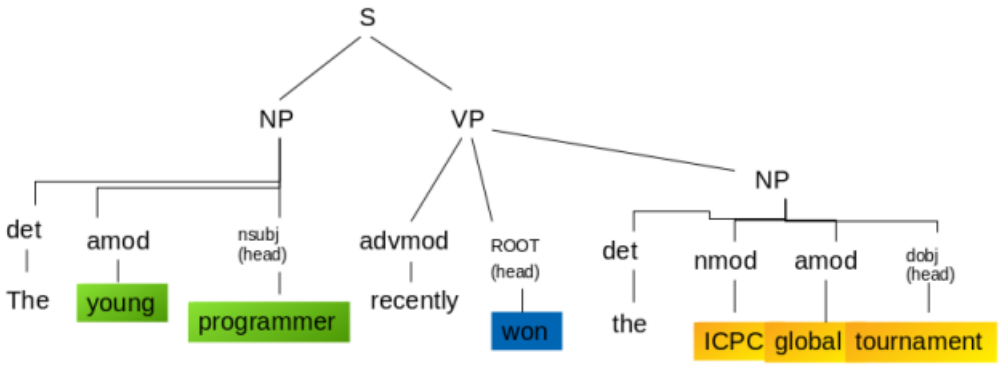

Figura 2.1: Árbol de parseo para la sentencia *"The young programmer recently won the ICPC global tournament"* Lovera et al. (2021)

Por último, una vez que tenemos las etiquetas gramaticales de cada palabra y sabemos como se relacionan entre sí, se construyen los grafos de conocimiento siguiendo la premisa de que los nodos han de ser entidades, mientras que las aristas relacionan ambas entidades. De forma que para cada sentencia se logra una representación en la que dos predicados nominales están unidos por un pedricado verbal mediante una arista dirigida, tal y como se puede ver en la figura 2.2

Figura 2.2: Representación del árbol de conocimiento para la sentencia *"The young programmer recently won the ICPC global tournament"* Lovera et al. (2021)

4. **Construcción de grafos de polaridad y cálculo de métricas de similitud**
   En esta fase, los autores, siguiendo la filosofía de los grafos de conocimiento, construirán dos grafos de polaridad, uno negativo y otro positivo. Y posteriormente compararán cada tweet del conjunto test con ambos grafos de polaridad para el análisis de sentimiento, esta comparación se realizará mediante el cálculo de métricas de similitud. Primero, los autores toman todos los tweets negativos del conjunto de entrenamiento. Para cada tweet forman un grafo de conocimiento, la combinación del grafo de cada tweet forma el grafo de polaridad negativo. El mismo proceso se sigue con los tweets positivos del conjunto de entrenamiento para obtener el grafo de polaridad positivo. Una vez tienen

ambos grafos de polaridad, tanto para el proceso de entrenamiento como para el proceso de test, hay que sacar ciertas métricas de similitud para comparar el grafo de conocimiento de un tweet con ambos grafos de polaridad, con el objetivo por supuesto de predecir si el tweet es positivo o negativo. Dichas métricas son las siguientes:

- similitud de contención: Se trata de una medida de similitud de grafos que expresa el porcentaje de aristas comunes entre los grafos, tomando el tamaño del grafo más pequeño como factor de normalización para esta medida. El grafo más pequeño siempre será el del tweet lógicamente.
- similitud con el máximo común subgrafo(nodos): Obtiene el máximo común subgrafo entre el grafo de polaridad y el de un tweet determinado, luego cuenta el número de nodos que están en este subgrafo que a su vez están también en el grafo del tweet. Tomando de nuevo el tamaño del grafo del tweet como factor de normalización para esta medida.
- similitud con el máximo común subgrafo(aristas): Igual que el caso anterior, pero contando las aristas en común en vez de los nodos.
- similitud con el máximo común subgrafo(aristas no dirigidas) : Igual que el caso anterior, pero eliminando la dirección de todas las aristas, es decir, tomando como iguales aristas que antes eran diferentes según la dirección.
- Una vez tenemos las 4 métricas explicadas que nombraremos para simplificar como m1,m2,m3 y m4. Ahora para cada grafo de polaridad tenemos un array de 8 valores para cada tweet [m1-, m2- , m3-, m4-, m1+, m2+, m3+, m4+] donde el símbolo - significa que la métrica está calculada respecto al grafo de polaridad negativo y el símbolo + que lo está respecto al positivo. Este vector de 8 elementos es el que se le introduce por cada tweet al modelo para entrenarlo y luego posteriormente para probarlo.

Por último, una vez se ha explicado la filosofía del modelo, nos presentan como será este. Como modelo predictivo, utilizan una implementación de dos redes neuronales, una LSTM(*Long short-term memory*) y una Bi-LSTM. Estos modelos toman como entrada el vector construido en el paso anterior Estas redes se consideran los modelos más eficaces para la predicción de sentimientos, ya que han demostrado su valor en la tarea de reconocer sentimientos.

5. **Cálculo de métricas de rendimiento**
   Para finalizar, se presentan los resultados finales mediante el cálculo de métricas de rendimiento. Para esta última fase combinan varios modelos y para compararlos utilizan las métricas Precision,F1-Score y Recall para la clase positiva. Para ello prueban cuatro modelos: *LSTM with KG*, *Bi-LSTM with KG*, *Character n-gram based LSTM* y *Character n-gram based Bi-LSTM*. Centrándonos, por ejemplo, en el primero de ellos, se obtienen los siguientes resultados: 0.884 en *F1-Score*, 0.88 en *Precision* y 0.89 en *Recall*. Por otro lado, con el segundo modelo obtiene 0.757 en *F1-Score*, 0.69 en *Precision* y 0.84 en *Recall*.

2.2. Artículo de Basiri et al. (2021)

El segundo artículo es *ABCDM: An Attention-based Bidirectional CNN-RNN Deep Model for sentiment analysis* Basiri et al. (2021). El principal objetivo de este trabajo es construir un modelo profundo de aprendizaje automático para el análisis de sentimiento. Dicho modelo consistirá en una combinación de una arquitectura de red neuronal convolucional bidireccional, una red neuronal recurrente con arquitecturas LSTMStaudemeyer and Morris (2019) y GRUDey and Salem (2017), y mecanismos de atención. Dicho modelo será llamado ABCDM.
Por una parte, la red CNN se utilizará para extraer características relevantes de las palabras y frases del texto. Por otra parte, la red RNN se utilizará para modelar las relaciones secuenciales y capturar el contexto global. Por último, el mecanismo de atención permitirá facilitar que el modelo pueda enfocarse en las partes importantes del texto y darles un peso durante el análisis de opinión.

Ahora, pasaremos a explicar el artículo siguiendo la misma metodología que el anterior. En este caso, dividiremos el proceso de trabajo en 4 apartados.

1. **Introducción**
   En este primer punto, los autores nos explican el creciente auge del análisis de sentimiento en redes sociales como twitter y por qué es importante. Por otro lado, añaden que dado que los modelos tradicionales para esta tarea suelen ser algunos como *support vector machines(SVM)*, *Naive Bayes*, *Latent Dirichlet allocation(LDA)* o redes neuronales simples, ellos van a proponer un modelo más complejo y novedoso como es la combinación de redes CNN y RNN. Por último, lo más importante de este apartado es que mencionan la contribución de su trabajo a los campos del análisis de sentimiento y procesamiendo del lenguaje natural. Dichas contribuciones son las siguientes.

   a) Construcción de una nueva arquitectura profunda para el análisis de sentimientos.

   b) Evaluación del modelo tanto para reviews y tweets largos, como para cortos.

   c) Comparación del rendimiento del modelo propuesto con seis arquitecturas profundas recientes para la clasificación de textos y el análisis de sentimientos.

2. **Preliminares**
   Posteriormente, en el artículo se menciona que, el modelo ABCDM que consiste en una combinación de CNN y RNN, se va a comparar con otros modelos como son *SS-BED*, *HAN*, *ARC*, *CRNN*, *IWV* y *AC-BiLSTM*. Tal y como muestrán en la siguiente tabla:

Research	RNN	CNN	Attention	Multi channel	Dataset type	Num of datasets
SS-BED [16]	✓				tweet	1
HAN [35]	✓		✓		review	4
ARC [36]	✓	✓	✓		tweet + review	2
CRNN [37]	✓	✓			review	3
IWV [18]		✓			review	5
AC-BiLSTM [17]	✓	✓	✓		review	7
ABCDM (This study)	✓	✓	✓	✓	tweet + review	8

Figura 2.3: Sumario de modelos que el artículo utiliza para la comparación Basiri et al. (2021)

Por otro lado, se menciona que las RNN, pueden estar basadas en una arquitectura LSTM o GRU. En el caso de este artículo, los autores mencionan que ellos combinarán ambas implementaciones, qué sumadas al modelo CNN forman un mecanismo de atención en una red bidireccional.

3. **Presentación del modelo**
   Tras haber explicado como funciona el modelo ABCDM, los elementos que lo integran(las redes RNN y sus respectivas células, y las redes CNN y el modelo de atención), en este apartado se presenta una propuesta final de como funciona el modelo. Primero mediante un pseudocódigo y luego mediante un esquema visual. En este caso mostraremos solamente el esquema visual, ya que consideramos que este es mucho más interpretable y fácil de entender que el pseudocódigo completo.

   Vemos en la figura 2.4 que el modelo se divide en 9 partes:

   - *input layer* : que es un conjunto de palabras que conforman un tweet.
   - *embedding layer* : capa en la que se pasa de un tweet a representación basada en una incrustación en una dimension d determinada.
   - *bidirectional layer* : Aquí tenemos las redes RNN con arquitectura LSTM y GRUB.
   - *concatenation* : Se concatenan los resultados obtenidos en la capa bidireccional para lograr un formato óptimo para el modelo de atención.
   - *Attention layer* : Esta es la capa de atención que permite que el modelo sea capaz de discernir las palabras más o menos importantes y actuar en consecuencia.
   - *Convolutional layer*: Esta es la capa en la que se encuentran las redes CNN. En esta fase se pasa a una representación agrupada de la información derivada del modelo de atención. Dicha representación agrupada se muestra mediante vectores de dimension d.
   - *concatenation* En esta fase se concatenan los distintos vectores obtenidos de la capa convolucional.
   - *Fully conected layer*. El proceso ya casi ha acabado, tras pasar el input por las capas de RNN, modelo de antención y CNN, se pone en esta última capa toda la información agrupada necesaria para la predicción final.
   - *Output layer*: Aquí la red neuronal arroja el resultado de la polaridad del tweet(negativo o positivo).

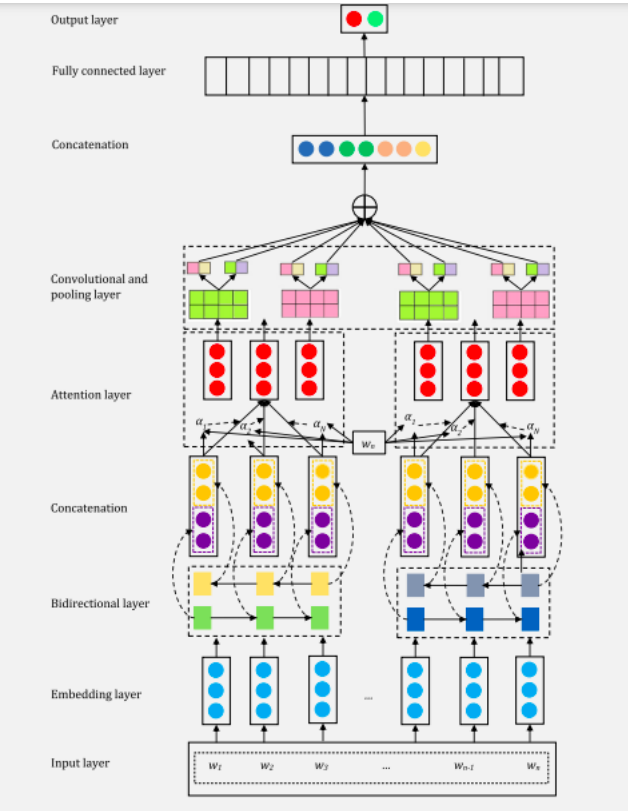

Figura 2.4: Representación del modelo completo [Basiri et al. (2021)]

4. **Presentación de resultados**:
   Primero, se presentan los conjuntos de datos que utilizan, que entre otros son *T4SA*, *Sentiment140* y *Twitter US Airlines*, conjuntos de datos también utilizados por nosotros. Por otro lado, realizan un estudio de ablación para ver con qué configuración del modelo ABCDM se quedan, como se ve en esta tabla:

BiLSTM Memory Units	CNN Filters	CNN Kernel Size	Dense Size	Avg. Accuracy
64	16	5, 7	128	0.890
64	32	4, 6	32	0.859
64	64	3, 5	128	0.882
128	16	3, 5	128	0.892
128	32	4, 6	64	0.905
128	64	4, 6	32	0.893
256	16	3, 5	32	0.879
256	32	5, 7	64	0.887
256	64	4, 6	32	0.881

Figura 2.5: Estudio de ablación para obtener el mejor modelo posible Basiri et al. (2021)

Por último, realizan una comparación entre el modelo ABCDM y 6 modelos más, que como mencionabamos anteriormente eran : *SS-BED*, *HAN*, *ARC*, *CRNN*, *IWV* y *AC-BiLSTM*. Para realizar dicha comparación utilizan las siguientes métricas:

- *Accuracy*
- *Precision* para la clase negativa y positiva.
- *Recall* para la clase negativa y positiva.
- *F1 score* para la clase negativa y positiva.

Y posteriormente, utilizando varios conjuntos de datos, muestran en unas tablas comparativas como se comporta cada modelo en cada métrica. Siguiendo la misma filosofía que en el artículo [Lovera et al. (2021)]. Por ejemplo, para el conjunto de datos *Twitter US Airlines*, utilizando el modelo *ABCDM* presentando obtienen una *Accuracy* de 0.93, un *Recall* de 0.96 en la clase positiva y 0.81 en la positiva, un *F1 Score* de 0.95 en la clase negativa y 0.84 en la positiva, y por último una *Precision* de 0.95 en la clase positiva y 0.84 en la positiva. También viendo un ejemplo de un conjunto de datos mucho más grande como es *Sentiment140*, se obtienen estos resultados: *Accuracy* de 0.82, un *Recall* de 0.90 en la clase positiva y 0.74 en la positiva, un *F1 Score* de 0.83 en la clase negativa y 0.81 en la positiva, y una *Precision* de 0.77 en la clase positiva y 0.88 en la positiva
Finalmente, como conclusión, los autores obtienen de forma fidedigna una manera de representar el comportamiento de su modelo. Ya que no solo prueban un número amplio de métricas, sino que para algunas de ellas también presentan los resultados para ambas clases(negativa y positiva). Esto permite conocer como funciona el modelo para conjuntos de datos desbalanceados en los que una clase predomina mucho más que la otra. Además, dado que nos presentan los datos para más de un conjunto de datos. Nos permite conocer como se comporta el modelo para conjuntos de diferentes características, como puede ser el tamaño de dichos conjuntos, si se trata una única temática o puede haber varías, si los textos son cortos o largos, etc.

2.3. Otros artículos

Para terminar con este capítulo vamos a ver otras tres implementaciones adicionales utilizadas en el estudio de análisis de sentimiento:

1. Construcción de grafos de conocimiento: Esta implementación realmente ya la hemos visto en el artículo Lovera et al. (2021), así que la explicaremos teóricamente: Esta trata de construir y consultar un grafo de conocimiento de entidades extraídas mediante procedimientos de procesamiento del lenguaje natural y ontologías extraídas Everywhere (2022). Una vez extraídos los grafos de conocimientos, hay que decir que estos son ricos en información, ya que permiten obtener grafos que representan tweets o incluso grafos de polaridad en la que las relaciones entre palabras o incluso frases quedan bien expuestas, tal y como se vio en las construcciones del artículo Lovera et al. (2021).

2. Herramientas de procesado de texto en Tensorflow (2022b): Estas técnicas se basan normalmente en estas tres fases:

   - Procesamiento previo de texto: El modelo aprende a generar un vocabulario a partir del texto de entrada. Este procesamiento se puede automatizar con modelos más sofisticados como BERT Tensorflow (2022a).

- Clasificación del texto: Mediante modelos clásicos como RNN, CNN, etc. u otro tipo de modelos más sofisticados como *BERT*.
- Generación de texto: Aquí bien se pueden emplear modelos de transformación para traducir texto o se pueden emplear modelos de secuencia a secuencia(*seq2seq*) para la misma finalidad.

3. Incorporación del movimiento ocular o *eye tracking* a las técnicas de análisis de opinión. Para esta parte utilizaremos el artículo *SEMGraph: Incorporating Sentiment Knowledge and Eye Movement into Graph Model for Sentiment Analysis*Wang et al. (2022) como referencia teórica.
   En este artículo los autores nos explican que el procesamiento léxico realizado por las personas va de la mano con los movimientos oculares de manera inconsciente. Por lo que poder medir los movimientos oculares puede ser un mecanismo mucho más potente para conocer la opinión real del usuario cuando interactúa con algo.

   Para este modelo, para obtener correctamente los inputs oculares, se crea lo que los autores llaman *un paradigma de movimientos oculares de sondeo lingüístico* que está basado en 3 posibles métricas/características lingüísticas:

   - Característica de un carácter: número de caracteres, palabras de la frase que empiezan por mayúscula.
   - Característica de la palabra: comprobar si una palabra en la frase de anotación, número de sentidos de la palabra en la red de palabras.
   - Característica de complejidad: puntuación de complejidad referida al número de palabras de la frase, distancia de dependencia máxima.

   Por último, también vemos que estos datos oculares son especialmente útiles dado que se pueden incluir a un modelo que, como vemos en el esquema, es similar al modelo ABCDM explicado en el último artículo.

Capítulo 3

Modelo propuesto

En esta sección se va a establecer la estructura así como el funcionamiento de nuestro modelo. Antes de comenzar, es importante mencionar que nuestro modelo está inspirado en las ideas de Lovera et al. (2021).
Como vimos en el estado del arte, Lovera, Cardinale y Homsi (2021) propusieron un modelo de análisis de sentimiento que se construye en dos fases: (1) se preprocesan los textos con técnicas de procesamiento del lenguaje natural para construir dos grafos de conocimiento (de polaridades positiva y negativa) y, (2) a partir de estos grafos de conocimiento, se extraen métricas de las palabras de cada texto para ajustar una red neuronal profunda con que clasificarlos posteriormente. El modelo de Lovera, Cardinale y Homsi (2021), como es natural en aquellos con mayor interpretabilidad, sacrifica parte de su capacidad predictiva en favor de la habilidad de explicar mejor por qué clasifica como lo hace.
Ahora, dividiremos la explicación de nuestro modelo en tres partes: las dos primeras(preprocesamiento y construcción de grafos de polaridad), serán los preliminares necesarios para generar las entradas de nuestro modelo, mientras que la tercera(ajuste del modelo) servirá para explicar que tipo de modelo predictor vamos a utilizar y como se realiza el proceso de entrenamiento y de test del mismo.

3.1. Preprocesamiento

En esta primera fase leeremos los textos a estudio desde un conjunto de datos etiquetado en formato CSV. Estos conjuntos tendrán, entre otros campos, el texto a analizar y la polaridad del mismo.
Una vez tenemos nuestros textos con su polaridad(positiva o negativa), realizaremos un preprocesamiento para lograr un formato óptimo de los mismos. El preprocesamiento en cuestión seguirá los siguientes pasos.
Pasamos el texto a String, ponemos todas las palabras en minúsculas, borramos los caracteres hexadecimales, así como las etiquetas/hashtags(#), las menciones a otros usuarios(@) y puntos suspensivos y demás signos de puntuación.
Luego, expandimos las contracciones del inglés, eliminamos las *stopwords* o palabras vacías(artículos, pronombres, preposiciones, etc.).
Para terminar, procedemos a dividir el texto en palabras y a obtener la etiqueta de cada palabra del texto(preposición, verbo, nombre... Etc.).

Para poder realizar todo esto, simplemente hemos ido recorriendo la lista de textos

aplicando las distintas modificaciones pertinentes. Para ello, se utilizan distintas funciones propias de Python, como *str()* para convertir un texto a string, *lower()* para convertirlo a minúsculas, así como librerías como re, utilizada para poder manipular expresiones regulares que nos ayudan a modificar con facilidad un texto, eliminando enlaces, caracteres especiales... Etc.
Por último, para la parte de la separación de los textos en palabras, así como la posterior obtención de categorías gramaticales para cada una, utilizaremos herramientas de procesamiento del lenguaje natural, gracias a la librería *nltk*.

3.2. Construcción de los grafos de polaridad

Una vez todos los textos han sido preprocesados, debemos generar los grafos de polaridad, para ello seguiremos los siguientes pasos.
Primero, se realiza la división train/test del conjunto de datos, separando los datos que se utilizarán para entrenar el modelo y los que se utilizarán para evaluar su rendimiento. A continuación, se divide el conjunto de entrenamiento en textos negativos y positivos, categorizando los textos según su polaridad.
Luego, se construye un grafo de polaridad negativo con los textos de entrenamiento negativo y un grafo de polaridad positivo con los textos de entrenamiento positivos. Para lograr esto, cada texto se representa como un grafo, donde cada palabra del texto se convierte en un nodo del grafo. La construcción del grafo de polaridad implica componer todos los pequeños grafos que representan los textos individuales, creando un grafo más grande que captura la polaridad general de los textos en el conjunto de entrenamiento.
Para la creación de los grafos de polaridad se ha optado por la elaboración de una función propia *build graph* que explicaré ahora mediante un pseudocódigo.

Algorithm 1 build graph(distancia, textos)

Require: $distancia \geq 0$
  $G \leftarrow (V, E)$ donde V es el conjunto de vértices del grafo y E es el conjunto de aristas del mismo.
  $grafos \leftarrow []$
  for `texto in textos` **do**
    $F \leftarrow (V', E')$
    for `palabra in texto` **do**
      $V' \leftarrow V' \cup palabra$
    end for
    $nodos \leftarrow V'$
    for $i \leftarrow 0$ `hasta` $i \leftarrow longitud(nodos)$ **do**
      for $j \leftarrow i + 1$ `hasta` $j \leftarrow i + distancia$ **do**
        $E' \leftarrow E' \cup (nodos(i), nodos(j))$
      end for
    end for
    $grafos \leftarrow grafos \cup F$
  end for
  $G \leftarrow composicion(grafos)$
  return G

Como se puede ver, la idea principal es crear un grafo para cada texto, y posteriormente, componer todos los diferentes grafos para crear el grafo de polaridad resultante. Sin embargo, hay un concepto muy importante que no habíamos introducido previamente, que es el de la distancia.
Partiendo de una lista de los nodos de un grafo, la distancia es el número de nodos posteriores con los que un nodo n tendrá conexión. Por ejemplo, para distancia uno, un nodo iría conectado con el siguiente, para distancia dos, con los dos siguientes, etc.
Por ejemplo, veamos como se representaría el texto "Mañana jueves lloverá mucho", con distancias 1,2 y 3.

Figura 3.1: Frase *Mañana jueves lloverá mucho* con distancia = 1 admin@graphonline.ru (2015)

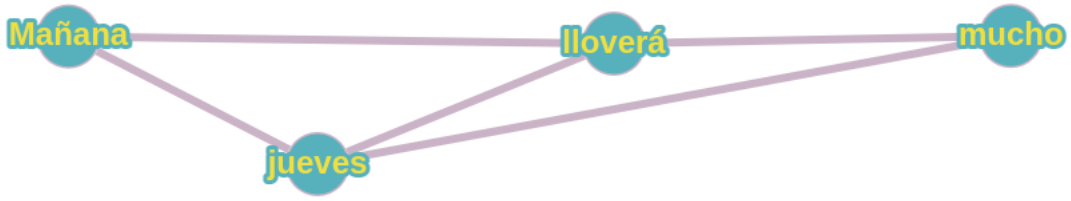

Figura 3.2: Frase *Mañana jueves lloverá mucho* con distancia = 2 admin@graphonline.ru (2015)

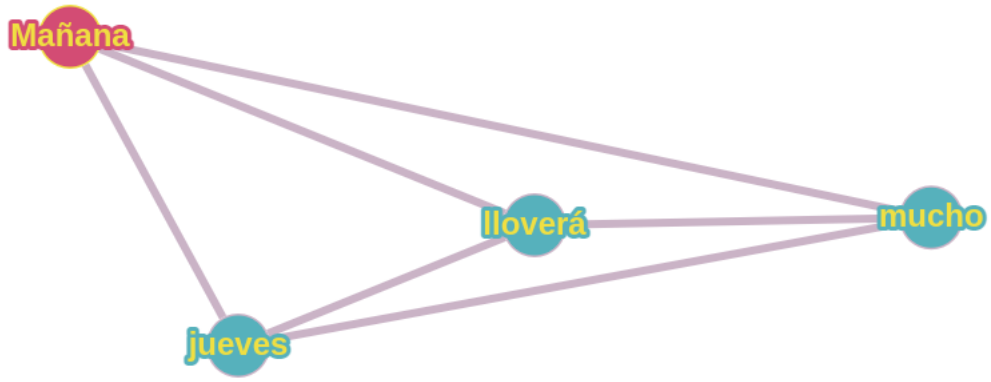

Figura 3.3: Frase *Mañana jueves lloverá mucho* con distancia = 3 admin@graphonline.ru (2015)

En las figuras 3.1, 3.2 y 3.3 podemos ver como se representa la frase *mañana jueves lloverá* empleando distancias 1, 2 y 3 respectivamente. De esta forma podemos ver de manera gráfica como se cumple que en el caso de distancia 1, cada nodo está unido al siguiente, como en el caso de distancia 2 está unido a los dos siguientes y como en el caso de distancia 3 está unido a los 3 siguientes. Dicho esto, podemos ver que cuanto más aumentemos la distancia, más conexo estarán los grafos que creemos, tanto los asociados a un texto como los grafos de polaridad. Sin embargo, eso trae consigo un aumento de la dificultad a la hora de calcular las métricas de centralidad, por lo que nuestro predictor puede mejorar o empeorar según la distancia utilizada. Y también parece claro que cuanto mayor sea la distancia mayor será el tiempo de cómputo requerido para la creación de los grafos y el cómputo de dichas métricas. Por último, también es importante mencionar que cuando empleamos una distancia *n-1* siendo n el número de nodos del texto se logra un grafo completo, ya que cada nodo estará unido al resto de ellos. Esto lo podemos ver precisamente en la figura 6.6, en la que tenemos 4 nodos y un grafo totalmente conexo mediante la utilización de una distancia igual a 3.

3.3. Ajuste del modelo de Red neuronal

En esta última fase, explicaremos el proceso de entrenamiento y test del modelo, así como la arquitectura y el funcionamiento del mismo.
Una vez tenemos ambos grafos de polaridad, los usaremos como medio de comparación y obtendremos las métricas de centralidad para comparar los grafos de cada texto con los grafos de polaridad.
Para ello obtendremos las métricas de *eigen vectors*, *betweenness*, *nodos en común* y *aristas en común*. Y dado que tenemos dos grafos de polaridad, cada texto se verá representado por 8 métricas. La forma de obtener dichas medidas se explica en el algoritmo 2.

Algorithm 2 Obtención de las métricas de centralidad

$X\_TRAIN \leftarrow []$ Lista donde se almacenan las métricas para el conjunto de entrenamiento.
$G \leftarrow (V, E)$ donde V es el conjunto de vértices del grafo y E es el conjunto de aristas del mismo.
for `texto in conjunto_train` **do**
  $G \leftarrow$ build_graph(G, texto)
  $m1 \leftarrow$ eigen vector(G , grafo de polaridad negativo)
  $m2 \leftarrow$ eigen vector(G , grafo de polaridad positivo)
  $m3 \leftarrow$ betweenness(G , grafo de polaridad negativo)
  $m4 \leftarrow$ betweenness(G , grafo de polaridad positivo)
  $m5 \leftarrow$ nodos en común(G , grafo de polaridad negativo)
  $m6 \leftarrow$ nodos en común(G , grafo de polaridad positivo)
  $m7 \leftarrow$ aristas en común(G , grafo de polaridad negativo)
  $m8 \leftarrow$ aristas en común(G , grafo de polaridad positivo)
  $X\_TRAIN \leftarrow X\_TRAIN \cup (m1, m2, m3, m4, m5, m6, m7, m8)$
end for

Antes de proseguir explicando el proceso de entrenamiento, vamos a explicar lo que significa cada métrica:

- eigen vector = "Este algoritmo evalúa las relaciones de los nodos con un esquema de puntuaciones. Los nodos de alta puntuación representan a aquellos vértices que tienen mayores conexiones y, por lo tanto, poseen un nivel de relevancia superior "everywhere (2022)
- betweenness = "La centralidad entre aristas se define como el número de caminos más cortos que pasan por una arista de un grafo. A cada arista de la red se le puede asociar un valor de centralidad entre aristas"Lu and Zhang (2013)
- nodos en común = se calcula el número de nodos en común que tiene un grafo de polaridad con el grafo asociado al texto en cuestión.
- aristas en común = se calcula el número de aristas en común que tiene un grafo de polaridad con el grafo asociado al texto en cuestión.

Una vez tenemos el array con las 8 métricas para representar un texto, introducimos dicho array al conjunto *X train* de nuestro modelo, mientras que el conjunto *Y train* serán las etiquetas conocidas para esos textos.
El modelo en cuestión será una red neuronal *Multilayer perceptron*. Dicha red neuronal seguirá una arquitectura de 5 capas con 32,128,128,128,32,8 respectivamente con el parámetro *solver = 'adam'*. Se utilizó ese *solver*, ya que favorece el entrenamiento para conjuntos de datos grandes como serán los nuestros.

Por último, con los textos del conjunto test, seguiremos el mismo proceso que anteriormente, crearemos grafos para cada texto, también empleado distancia 1. Tras ello, compararemos dichos grafos con ambos grafos de polaridad y tendremos así las 8 métricas para cada uno, y pasaremos dicho array al modelo para posteriormente predecir su polaridad. Después de eso obtendremos un nuevo array llamado *predicciones* que devolverá para cada texto un valor de 0 si el modelo ha decidido que el texto sea negativo y un 1 si el modelo decide que sea positivo. Dichas predicciones podremos compararlas con las etiquetas reales para conocer el rendimiento de nuestro modelo tal y como veremos más adelante.

Capítulo 4

Experimentación

4.1. Conjuntos de datos utilizados

Antes de comenzar a explicar el proceso seguido para la experimentación de nuestro modelo, se debe establecer los conjuntos de datos que utilizaremos. Por lo que en esta primera parte estudiaremos los conjuntos de datos utilizados para el trabajo.
El primero será Sentiment140 Stanford (2010). Este conjunto de datos es el más grande de todos los que utilizaremos, siendo que tiene 1.600.000 tweets, de los cuales 800.0000 son negativos y 800.000 son positivos. Cada tweet del conjunto de datos estará clasificado por los siguientes campos: La polaridad del tweet(0 = negativo, 2 = neutral, 4 = positivo), el ID del tweet, la fecha del tweet, la consulta(lyx o si no hay consulta, entonces este valor es "NO_QUERY"), el usuario que tuiteó y el texto en sí del tweet. En este conjunto de datos, al tener tal cantidad de tweets, los tweets no se centrarán en un conjunto de temáticas en particular, sino que podemos encontrarnos cualquier tipo de tweet hablando de cualquier tema.

El siguiente conjunto de datos a mencionar será Amazon fine food reviewsStanford (2012), que como su nombre indica, trata sobre las opiniones de los usuarios que compraron ciertos productos de amazon. Este conjunto contiene 568.454 reviews de 256.059 usuarios distintos y de 74.258 productos distintos. Para almacenar y poder analizar posteriormente cada review, estas se identifican mediante los siguientes campos: El ID del producto, el ID del usuario que escribió la review, así como su nombre, la fracción de usuarios que encontraron útil la review, la puntuación del producto (1-2 negativo, 3 neutral(se obviarán) y 4-5 positivo), la fecha de la review(en tiempo UNIX), un breve sumario de la review y el texto completo de la propia review.

Posteriormente, trataremos con el conjunto de datos T4SA Vadicamo et al. (2017), que contiene 550.391 tweets, de los cuales 179.050 son negativos, 371.341 son positivos y son 629.566 neutrales(que se obviarán en nuestro caso). Para identificar cada tweet, este se verá reflejado por solo dos campos: El Texto del tweet y la polaridad del mismo(0=negativo, 1=positivo). Por otro lado, hay que mencionar que al igual que en el caso del conjunto de datos *Sentiment140*, en este conjunto también puede haber tweets sobre cualquier temática.

El siguiente conjunto de datos a analizar será Twitter US Airline Sentiment Crowdflower (2015), que contiene 14640 tweets de los cuales 9178 son negativos, 3099 son

neutrales y 2363 son positivos(Como siempre, obviaremos las neutrales). Para representar cada tweet se utilizan los siguientes campos: El ID del tweet, la polaridad de la review positivo/neutral/negativo, una breve razón del voto negativo(2-6 palabras), el nombre de la aerolínea, el nombre del usuario que escribió el tweet, el número de retweets del tweet, el texto de la review en sí, las coordenadas del tweet, la fecha en la que se publicó el tweet, el lugar donde se escribió el tweet y su zona horaria.
Respecto al tema que trata este conjunto, este recoge opiniones de clientes que tomaron recientemente algún avión de las principales aerolíneas de Estados Unidos.

El último conjunto de datos a estudio será SST2 Socher et al. (2013), que contiene 69170 reviews(30692 negativas y 38478 positivas) y cuyos campos para representarlas son los siguientes: El ID del tweet, una indicación de si el tweet forma parte del conjunto de entrenamiento o de test. la polaridad del tweet(0 = negativo, 1 = positivo) y el texto de la review. Respecto a las temáticas que toca este conjunto de datos, al igual que pasaba con algunos casos anteriores, no hay una temática concreta.

Para finalizar, para cada conjunto de datos, vamos la distribución del número de palabras por tweet que estos están preprocesados.

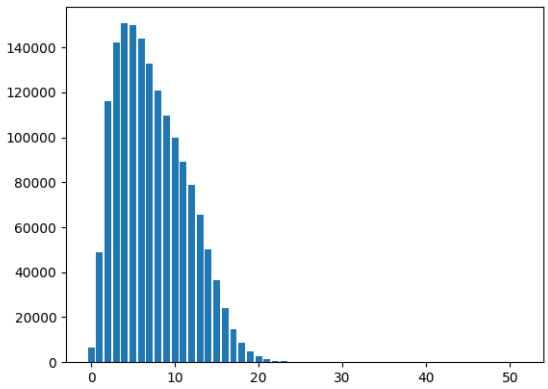

Figura 4.1: Distribución del número de palabras de los tweets del conjunto de datos sentiment140

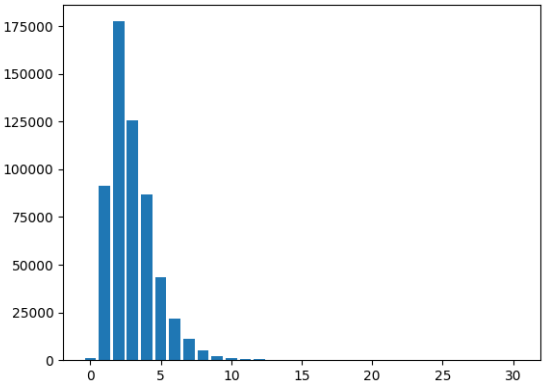

Figura 4.2: Distribución del número de palabras de los tweets del conjunto de datos Amazon fine food reviews

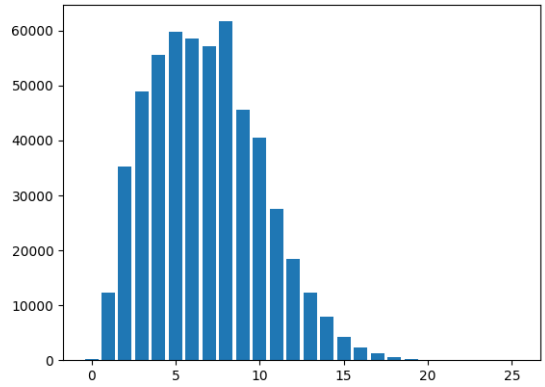

Figura 4.3: Distribución del número de palabras de los tweets del conjunto de datos T4SA

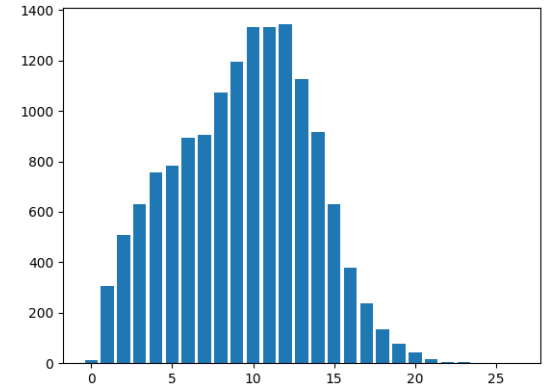

Figura 4.4: Distribución del número de palabras de los tweets del conjunto de datos Twitter US Airlines

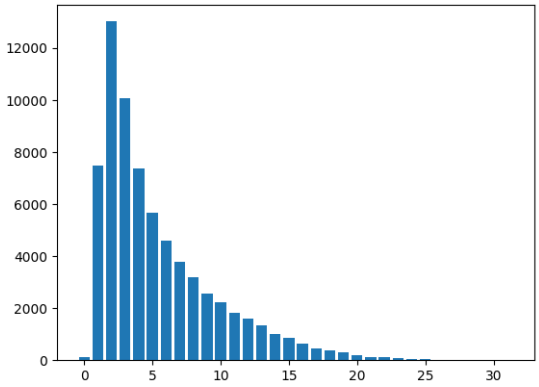

Figura 4.5: Distribución del número de palabras de los tweets del conjunto de datos SST2

4.2. Lectura de los conjuntos de datos y construcción del modelo

Una vez hemos explicado los conjuntos de datos que utilizaremos, proseguiremos explicando la primera parte del proceso de experimentación, que es la lectura de los conjuntos de datos y la construcción del modelo. Tal y como se explicó anteriormente, los conjuntos de datos contendrán, entre otros campos, el texto del tweet/review así como su polaridad, siendo estos dos campos los únicos que necesitaremos en la práctica.
Para cada conjunto de datos, se realizará una partición en conjunto de entrenamiento y conjunto de test. El primer conjunto será para probar el modelo y el segundo para probarlo. La división será de 80 % de los tweets para el conjunto de entrenamiento y el 20 % de los tweets para el conjunto test. Y para que no haya una única partición, el conjunto de datos se reordenará aleatoriamente siempre que el programa python lo lea. Además de eso, para alcanzar mayor fiabilidad con los resultados obtenidos, a partir de un conjunto de datos inicial, se generarán 5 versiones del conjunto de datos, emulando así una separación del mismo en 5 folds. Por lo que los resultados de nuestro modelo se obtendrán realmente ponderando el rendimiento del mismo en cada uno de los 5 folds. Esto es interesante, ya que, debido a que en cada fold cambiará la distribución de tweets, también cambiarán los grafos de polaridad que son la parte clave para el entrenamiento del modelo. Por lo que podremos ver como se comporta nuestra red neuronal en varios escenarios de distintos inputs.
Una vez se ha entendido que trabajaremos con 5 versiones de un mismo conjunto de datos, las cuales hacen las veces de un 5 folds, solo quedaría explicar como se genera cada fold.

El proceso es sencillo, primero leemos todos los tweets etiquetados desde un archivo con extensión CSV. Posteriormente, una vez tenemos todos los tweets, conseguimos los 5 folds utilizando la función "StratifiedKFold"Pedregosa et al. (2011) pasando como parámetro $k = 5$. Es decir, una partición en 5 del conjunto de datos, de esta forma también nos encargamos de que cada partición train/test sea del 80/20. Una vez sabemos, para cada fold, que tweets son para entrenamiento y cuáles son

para test, solo nos queda saber cuáles son positivos y cuáles negativos. Ese dato lo podemos saber fácilmente por qué todos los tweets están etiquetados en el CSV inicial. Por lo que al final tenemos para cada fold 4 conjuntos de tweets: Los positivos de entrenamiento, los negativos de entrenamiento, los positivos de test y los negativos de test. Finalmente, para poder guardar las nuevas versiones del conjunto de datos, guardamos estos 4 conjuntos que hemos mencionado en una carpeta cada uno. Por lo que cada fold tendrá esas 4 carpetas. Lo que facilitará futuras lecturas de los folds.

Figura 4.6: Ejemplo de como se obtienen 5 conjuntos de datos a partir de uno incial

Figura 4.7: Cada fold se divide en train/test

Figura 4.8: Tanto para el conjunto train como para el test, lo dividimos en negativos y positivos

4.3. Cálculo de métricas de rendimiento

Una vez hemos entrenado el modelo con los elementos del conjunto de entrenamiento y lo hemos probado con los elementos del conjunto test. Sacaremos algunas métricas para poder entender como de bien ha clasificado el modelo. Dichas métricas serán: *AUC*, *Accuracy*, *Precision*, *Recall* y *F1-Score*. Dichas métricas se definen de la siguiente manera(Pedregosa et al. (2011)):

- Área bajo la curva(AUC) = es una herramienta estadística que se utiliza para medir el acierto en la predicción de eventos binarios
- $Accuracy = \frac{número\ de\ predicciones\ correctas}{número\ de\ predicciones\ totales}$
- Precision para la clase negativa y positiva. $Precision = \frac{True\ Positive}{True\ Positive\ +\ False\ Positive}$
- Recall. $Recall = \frac{True\ Positive}{True\ Positive\ +\ False\ Negative}$
- F1-Score. $F1 - SCORE = \frac{2 * Precision * Recall}{Precision + Recall}$

Por último, en el apartado temporal se presentarán dos métricas. Estas serán el *Tiempo entrenamiento*, que será el tiempo necesario para que el modelo realice su entrenamiento, y por último el *Tiempo total*, que es la medida temporal que abarca todo el programa, desde su primera instrucción hasta la última.

4.4. Selección de parámetros y estudio de ablación

Durante todo el trabajo, se ha repetido que el modelo parte de una entrada de 8 métricas de centralidad, obtenidas comparando el grafo que representa un texto con los dos grafos de polaridad. También se ha explicado el concepto de distancia para medir la forma en la que se construye el grafo en términos de conexiones(aristas) entre nodos. Sin embargo, lo que no se ha explicado es porque hemos tomado 8 métricas, porque esas métricas en concreto, o porque no otra combinación más simple de las mismas, o que distancia se ha elegido para la representación de los textos y por qué. Entonces, para justificar nuestra elección, vamos a considerar 4 modelos, que presentaremos a continuación, viendo que distancia emplea cada uno para la creación de los grafos, además de las medidas de centralidad que utilizan.

Cuadro 4.1: **Estudio de ablación**

Modelo	distancia	eigen_vector	betweenness	aristas en común	nodos en común
Predictor total	2	X	X	X	X
Predictor total(d1)	1	X	X	X	X
Predictor nodos+aristas en común	2			X	X
Predictor nodos+aristas en común(d1)	1			X	X

Ahora que sabemos los 4 modelos que se van a probar, vamos a establecer que nuestro modelo de partida es el modelo *Predictor total(d1)*. Primero, consideramos que las métricas de centralidad utilizadas son las necesarias. Ya que añadir más complicaría el modelo y aumentaría el tiempo de cómputo, cosa que no queremos, mientras que eliminar métricas simplificaría demasiado el modelo y obtendríamos peores resultados.
Segundo, creemos que, dado que nuestro conjunto de datos trabaja mejor para conjuntos de datos y textos no muy grandes, con distancia 1 conseguimos el mejor compromiso de rendimiento/tiempo requerido de cómputo. Más adelante en el capítulo de resultados veremos si nuestra hipótesis se ha cumplido y se decidirá basándonos en eso que modelo emplear finalmente.

4.5. Comparación con modelos de lenguaje BERT

Por último, una vez que tenemos el modelo elegido, así como nuestras métricas de rendimiento, es hora de ver como se ha comportado nuestro modelo comparándolo con modelos más complejos. Dichos modelos serán los modelos de lenguaje *BERT*, que se explicarán con más detalle en el capítulo 5, dedicado a la presentación de resultados. Aunque si mencionaremos que utilizaremos tres modelos:
small bert/bert en uncased L2 H128 A2 Turc et al. (2019),
small bert/bert en uncased L4 H128 A2 Turc et al. (2019) y
small bert/bert en uncased L2 H256 A4 Turc et al. (2019).

Por último, para finalizar este apartado, es importante mencionar que para probar los modelos *BERT*, ejecutaremos un solo fold en vez de los 5 creados. Esto es así porque un solo fold ya de por sí tomará mucho más tiempo de cómputo utilizando estos modelos complejos, tal y como veremos luego en los resultados.

4.6. Interpretabilidad del modelo

Para finalizar con la experimentación, vamos a realizar un análisis de interpretabilidad en el que vamos a ver un ejemplo concreto de como se compara el grafo respectivo a un texto con los dos grafos de polaridad. Dicho análisis, va a seguir los siguientes pasos:
Primero, para que la presentación de los resultados sea manejable, vamos a tomar un conjunto de datos pequeño, como es el conjunto de datos STS Wang et al. (2022)que contiene 2034 tweets, 632 positivos y 1402 negativos. Hemos elegido un conjunto de datos pequeño, ya que de elegir otros más grandes, la presentación de los grafos de polaridad en el programa que utilizaremos(*gephi*)Bastian et al. (2009) sería ininteligible. Y nuestro principal objetivo en este análisis es poder observar bien ambos grafos de polaridad para poder posteriormente analizarlos. Además, vemos que este conjunto de datos posee una cantidad relevante de tweets de ambas clases. De nada nos serviría un conjunto de datos extremadamente desbalanceado para este análisis.

Una vez que ya tenemos el conjunto de datos, vamos a tomar los 632 positivos y elegiremos al azar 632 negativos de los 1402 que hay. De esta forma tendremos dos grupos de tweets, 632 negativos y 632 positivos. Posteriormente, tomamos 316 tweets negativos para el conjunto de entrenamiento y otros 316 positivos. Luego hacemos lo mismo con los positivos, de forma que al final tenemos 316 tweets negativos para el entrenamiento, 316 tweets positivos para el entrenamiento, 316 tweets negativos para el test y 316 tweets positivos para el test.
Posteriormente, crearemos los grafos de polaridad con los tweets del conjunto de entrenamiento, entrenaremos el modelo y luego lo probaremos con el conjunto test. Por último, a la vez que realizamos la predicción de etiquetas para los tweets del conjunto test, guardaremos la probabilidad que tiene cada uno de ser clasificado como positivo por el predictor. De esta forma, aquel tweet con mayor probabilidad será considerado como el tweet más positivo según el predictor, y aquel con menor probabilidad será considerado como el tweet más negativo. Por último, representaremos esos dos tweets junto con los grafos de polaridad y veremos en que se ha basado el modelo para clasificar como ha clasificado.

Capítulo 5

Resultados

Esta sección la vamos a dividir en 4 partes:

- Como se presentarán los resultados del modelo.
- Selección de los mejores parámetros y el estudio de ablación.
- Una vez obtenidas las pruebas, comparación de nuestros resultados con otros modelos más sofisticados utilizando los mismos conjuntos de datos.
- Análisis de interpretabilidad de nuestro modelo.

5.1. Presentación de los resultados con nuestro modelo

En esta primera fase, como expliqué anteriormente. Hemos realizado las pruebas para obtener las métricas mencionadas para los 5 conjuntos de datos(Sentiment140, Twitter US airlines, Amazon fine food reviews, T4SA y SST2). Dichas métricas, además de los resultados temporales(tiempo de entrenamiento del modelo y tiempo total del programa) , se calcularán realizando la media de los valores obtenidos en cada uno de los 5 folds.
En definitiva, para tener una representación gráfica del rendimiento de un modelo determinado, crearemos una tabla en las que las columnas serán las siguientes: *AUC* o el área bajo la curva, la *Accuracy*, la *Precision* tanto para la clase negativa como para la positiva, el *Recall* también para las clases negativa y positiva y el *F1 score* siguiendo la misma dinámica.
Por último, en el apartado temporal se presentarán dos métricas. Estas serán el *Tiempo entrenamiento*, que será el tiempo necesario para que el modelo realice su entrenamiento, y por último el *Tiempo total*, que es la medida temporal que abarca todo el programa, desde su primera instrucción hasta la última.

5.2. Selección de parámetros y estudio de ablación

Si recordamos lo mencionado en el capítulo de experimentación, probaremos cuatro modelos para los distintos conjuntos de datos. También, como habíamos establecido,

partiremos del modelo *Predictor total (d1)*, que empleaba las métricas *eigen vectors*, *betweenness*, *nodos en común* y *aristas en común*, construyendo los grafos con distancia 1.
Ahora, en la tabla 5.1, veremos si nuestra hipótesis ha sido correcta y el modelo elegido realmente es el mejor.
A la vista de los resultados obtenidos en la tabla, podemos afirmar que el *predictor total*, ofrece unos resultados sustancialmente mejores que el predictor que solo toma nodos y aristas en común, todo esto sin que el tiempo de cómputo se dispare inasumiblemente. Por lo que esto justifica la elección de nuestras métricas y el porqué no hemos elegido un modelo más simple. Por otro lado, respecto a las distancias empleadas, vemos que el salto en rendimiento cuando empleamos distancia 2 respecto a cuando empleamos distancia 1 es ínfimo, mientras que si se ve claramente que la distancia afecta considerablemente en el tiempo de cómputo.
Por lo que, en resumen, podríamos decir que nuestra hipótesis se ha cumplido y nos quedaremos con el *predictor total(d1)* como nuestro modelo definitivo. Consideramos que es un modelo equilibrado porque, por un lado, se simplifica el modelo eligiendo distancia 1 en vez de 2 o superiores, pero, por otro lado, no es demasiado simple al tener un número de medidas de centralidad más grande. Por lo que en definitiva es un modelo en esencia sencillo, pero equilibrado, es decir, no demasiado sencillo.

5.3. Comparación con otros trabajos

En esta sección, vamos a partir de los resultados obtenidos por nuestro modelo elegido en la tabla anterior(*predictor(d1)*) y lo vamos a comparar con los obtenidos mediante los modelos de lenguajes *BERT*. *"BERT es una técnica basada en redes neuronales para el pre-entrenamiento del procesamiento del lenguaje natural desarrollada por Google"* Tensorflow (2022a).
Para realizar las pruebas con estos modelos de lenguaje, no hemos necesitado desarrollar código alguno, pues la implementación se ha podido obtener de esta página Clasificar texto con BERT Tensorflow (2022a). En la que se puede cambiar facilmente el conjunto de datos a estudio, así como el modelo *BER*T a utilizar. Esto último es posible debido a un extenso diccionario que permite elegir un modelo entre una larga lista(*BERT*, *ELECTRA*, *ALBERT*, etc.).

Para nuestro estudio hemos elegido los 3 siguientes modelos como referencia (véase la tabla 5.2).
En la tabla 5.3 podremos ver los resultados finales del trabajo en los que se comparan nuestro modelo y los 3 modelos *BERT*.

Cuadro 5.1: **Resultados del estudio de ablación**

AIRLINES	AUC	ACC	P^-	P^+	R^-	R^+	F_1^-	F_1^+	T_train	T_total
Predictor total	0.91	0.90	0.91	0.82	0.97	0.62	0.94	0.71	20 segundos	103 segundos(2 minutos)
Predictor nodos+aristas en común	0.86	0.86	0.90	0.68	0.92	0.60	0.91	0.63	17 segundos	98 segundos(2 minutos)
Predictor total(d1)	0.91	0.89	0.90	0.82	0.96	0.59	0.93	0.68	14 segundos	132 segundos(2 minutos)
Predictor nodos+aristas en común(d1)	0.84	0.82	0.90	0.60	0.88	0.60	0.89	0.57	15 segundos	43 segundos(3.5 minutos)
SST2	AUC	ACC	P^-	P^+	R^-	R^+	F_1^-	F_1^+	T_train	T_total
Predictor total	0.94	0.86	0.92	0.82	0.79	0.93	0.85	0.87	269 segundos(4.5 minutos)	994 segundos(16.5 minutos)
Predictor nodos+aristas en común	0.93	0.84	0.94	0.78	0.72	0.95	0.82	0.86	89 segundos(1.5 minutos)	173 segundos(3 minutos)
Predictor total(d1)	0.94	0.88	0.90	0.86	0.85	0.87	0.88	0.88	113 segundos(2 minutos)	256 segundos(4 minutos)
Predictor nodos+aristas en común(d1)	0.93	0.84	0.95	0.77	0.72	0.96	0.82	0.86	73 segundos(1.25 minutos)	146 segundos(2.5 minutos)
Amazon	AUC	ACC	P^-	P^+	R^-	R^+	F_1^-	F_1^+	T_train	T_total
Predictor total	0.91	0.92	0.79	0.94	0.66	0.97	0.71	0.95	6849 segundos(1 hora 54 min)	7497 segundos(2 horas 5 min)
Predictor nodos+aristas en común	0.83	0.89	0.88	0.89	0.33	0.99	0.48	0.94	1806 segundos(30 min)	2404 segundos(40 min)
Predictor total(d1)	0.91	0.92	0.81	0.93	0.62	0.97	0.70	0.95	6097 segundos(1 hora 42 min)	6492 segundos(1 hora 48 min)
Predictor nodos+aristas en común(d1)	0.83	0.89	0.85	0.89	0.33	0.99	0.48	0.94	1509 segundos(25 min)	1947 segundos(32 min)
T4SA	AUC	ACC	P^-	P^+	R^-	R^+	F_1^-	F_1^+	T_train	T_total
Predictor total	0.97	0.93	0.92	0.93	0.86	0.97	0.89	0.95	2274 segundos(38 minutos)	3116 segundos(52 min)
Predictor nodos+aristas en común	0.94	0.86	0.87	0.85	0.66	0.95	0.75	0.90	1202 segundos(20 min)	1890 segundos(32 min)
Predictor total(d1)	0.96	0.92	0.92	0.92	0.82	0.97	0.87	0.94	1695 segundos(28 minutos)	2478 segundos(41 min)
Predictor nodos+aristas en común(d1)	0.92	0.83	0.84	0.84	0.62	0.94	0.71	0.88	1018 segundos(17 minutos)	1691 segundos(28 min)
Sentiment	AUC	ACC	P^-	P^+	R^-	R^+	F_1^-	F_1^+	T_train	T_total
Predictor total	0.73	0.71	0.70	0.72	0.73	0.68	0.71	0.70	6344 segundos(1 hora 46 minutos)	7943 segundos(2 horas 12 min)
Predictor nodos+aristas en común	0.67	0.62	0.68	0.59	0.45	0.79	0.54	0.67	4776 segundos(1 hora 20 min)	6126 segundos(1 hora 42 min)
Predictor total(d1)	0.73	0.71	0.72	0.71	0.70	0.72	0.71	0.71	5399 segundos(1 hora 30 min)	7476 segundos(2 horas 5 min)
Predictor nodos+aristas en común(d1)	0.65	0.60	0.67	0.57	0.38	0.81	0.49	0.67	3124 segundos(52 min)	5025 segundos(1 hora 24 min)

Cuadro 5.2: **Modelos BERT elegidos**

Modelo	capas	neuronas por cada	cabezas de atención
BERT L2 H128 A2	2	128	2
BERT L4 H128 A2	4	128	2
BERT L2 H256 A4	2	256	4

Como resumen de los resultados, vemos que los modelos BERT dan unos resultados mejores que los nuestros. Sin embargo, aunque si que es verdad que en el conjunto de datos de sentiment140 la diferencia de rendimiento entre nuestros modelos y los modelos BERT es grande y si podría merecer la pena su uso, hay que darse cuenta de que en el resto de modelos la diferencia de rendimiento es más modesta y no justifica la enorme diferencia temporal. Una vez entendido eso, nos vamos a detener en los resultados de cada conjunto de datos.
Empezando por el conjunto de datos *Twitter US Airlines*, este es el conjunto más pequeño. por lo que es razonable que los tiempos de ejecución sean mucho menores que en los otros cuatro que veremos a continuación. Para comparar las métricas entre los 4 modelos que se ponen a prueba, hemos marcado para métrica en negrita el modelo que mejores resultados obtiene. Siguiendo esto podemos ver que aunque en algunas métricas hay empate entre dos o más modelos, el último modelo *BERT L2 H256 A4* es el que mejores resultados arroja en las métricas de rendimiento. Pero también vemos que es el que más tiempo consume. Por comparar nuestro modelo con el mejor modelo de los *BERT*, vemos que este último mejora al nuestro en un 5 % en el área bajo la curva, un 4 % en *accuracy*, 5 % en la precisión negativa, 3 % en la positiva, 0 % en el recall negativo, 35 % en el positivo, 3 % en el F1 negativo y 22 % en el positivo. Por otro lado, en el apartado temporal, vemos que el modelo *BERT* es un 1630 % más lento que el nuestro. Observando los datos de rendimiento podemos que, en la mayoría de los casos, nuestro modelo se acerca al mejor modelo *BERT*, aunque sí es verdad que este último logra balancear mejor las clases positiva y negativa. Sin embargo, aunque vemos que la diferencia temporal es muy grande, si vemos los resultados en números brutos, vemos que 38 minutos es un número aceptable. incluso si no tomásemos el mejor *BERT* y tomásemos el segundo mejor, el tiempo total serían 30 minutos que sería aún más asequible. Por lo que en conclusión, vemos que en este conjunto de datos, nuestro modelo se acerca en la mayoría de métricas de rendimiento, y que en términos temporales es mucho más rápido, aunque aún sería razonable utilizar un modelo *BERT* viendo que el tiempo de ejecución no se dispara en exceso.

Siguiendo con el conjunto SST2, vemos que el mejor modelo *BERT* vuelve a ser *BERT L2 H256 A4*. Comparando de nuevo las métricas de este modelo con el nuestro, vemos que consigue aumentar en un 3 % en el área bajo la curva, un 3 % en *accuracy*, 1 % en la precisión negativa, 6 % en la positiva, 7 % en el recall negativo, 5 % en el positivo, 3 % en el tanto en F1 negativo como en el positivo. Por otro lado, en el apartado temporal, el modelo *BERT* es un 5936 % más lento. En este caso, vemos que nuestro modelo, a diferencia de en el conjunto de datos *Airlines*, consigue balancear mejor las clases positivas y negativas. Y vemos que aquí los resultados temporales ya se empiezan a disparar más, vemos que incluso en el modelo *BERT* más sencillo, *BERT L2 H128 A2*, la diferencia temporal respecto al nuestro es de un 2527 % más. Una diferencia que superaba a la observada en el conjunto *Airli-*

Cuadro 5.3: **Comparación de nuestro modelo con *BERT***

AIRLINES	AUC	ACC	P^-	P^+	R^-	R^+	F_1^-	F_1^+	T_train	T_total
Predictor total(d1)	0.91	0.89	0.90	0.82	0.96	0.59	0.93	0.68	14 segundos	132 segundos(2 minutos)
BERT L2_H128_A2	0.95	0.92	0.94	0.83	0.96	0.74	0.95	0.78	934 segundos(16 minutos)	982 segundos(16 minutos)
BERT L4_H128_A2	0.96	0.93	0.95	0.84	0.96	0.79	0.95	0.82	1711 segundos(29 minutos)	1790 segundos(30 minutos)
BERT L2_H256_A4	0.96	0.93	0.95	0.85	0.96	0.80	0.96	0.83	2191 segundos(37 minutos)	2284 segundos(38 minutos)
SST2	AUC	ACC	P^-	P^+	R^-	R^+	F_1^-	F_1^+	T_train	T_total
Predictor total(d1)	0.94	0.88	0.90	0.86	0.85	0.87	0.88	0.88	113 segundos(2 minutos)	256 segundos(4 minutos)
BERT L2_H128_A2	0.95	0.88	0.88	0.88	0.88	0.89	0.88	0.88	6669 segundos(1 hora 51 min)	6724 segundos(1 hora 52 minutos)
BERT L4_H128_A2	0.96	0.89	0.88	0.91	0.91	0.87	0.89	0.89	11888 segundos(3 horas 18 min)	11993 segundos(3 horas 20 min)
BERT L2_H256_A4	0.97	0.91	0.91	0.91	0.91	0.91	0.91	0.91	15366 segundos(4 horas 16 min)	15452 segundos(4 horas 18 minutos)
Amazon	AUC	ACC	P^-	P^+	R^-	R^+	F_1^-	F_1^+	T_train	T_total
Predictor total(d1)	0.91	0.92	0.81	0.93	0.62	0.97	0.70	0.95	6097 segundos(1 hora 42 min)	6492 segundos(1 hora 48 min)
BERT L2_H128_A2	0.97	0.94	0.85	0.96	0.76	0.98	0.80	0.97	39769 segundos(11 horas 3 min)	41228 segundos(11 horas 27 min)
BERT L4_H128_A2	0.97	0.95	0.86	0.96	0.78	0.98	0.82	0.97	73546 segundos(20 horas 26 min)	76087 segundos(21 horas 8 min)
BERT L2_H256_A4	0.97	0.95	0.87	0.96	0.80	0.98	0.83	0.97	95160 segundos(26 horas 26 min)	98388 segundos(27 horas 20 min)
T4SA	AUC	ACC	P^-	P^+	R^-	R^+	F_1^-	F_1^+	T_train	T_total
Predictor total(d1)	0.96	0.92	0.92	0.92	0.82	0.97	0.87	0.94	1695 segundos(28 minutos)	2478 segundos(41 min)
BERT L2_H128_A2	0.99	0.99	0.98	0.99	0.98	0.99	0.98	0.99	41822 segundos(11 horas 37 min)	43607 segundos(12 horas 7 min)
BERT L4_H128_A2	1.00(0.996)	0.99	0.99	0.99	0.98	0.99	0.98	0.99	76772 segundos(21 horas 20 min)	79487 segundos(22 horas 5 min)
BERT L2_H256_A4	1.00(0.997)	0.99	0.99	1.00(0.996)	0.99	1.00(0.997)	0.99	1.00(0.995)	100561 segundos(27 horas 56 min)	102960 segundos(28 horas 36 min)
Sentiment	AUC	ACC	P^-	P^+	R^-	R^+	F_1^-	F_1^+	T_train	T_total
Predictor total(d1)	0.73	0.71	0.72	0.71	0.70	0.72	0.71	0.71	5399 segundos(1 hora 30 min)	7476 segundos(2 horas 5 min)
BERT L2_H128_A2	0.90	0.82	0.81	0.83	0.83	0.81	0.82	0.82	74796 segundos(20 horas 46 min)	77806 segundos(21 horas 37 min)
BERT L4_H128_A2	0.91	0.83	0.82	0.83	0.83	0.83	0.82	0.83	136479 segundos(37 horas 55 min)	141506 segundos(39 horas 18 min)
BERT L2_H256_A4	0.92	0.84	0.83	0.85	0.85	0.83	0.84	0.83	181100 segundos(50 horas 18 min)	187307 segundos(52 horas 2 min)

nes. Podemos ver ya un indicio de que a medida que el conjunto de datos aumenta de tamaño, la diferencia temporal puede empezar a ser inasumible, y que mientras nuestro modelo se mantenga cercano en las métricas de rendimiento, podría ser mejor que los modelos *BERT*.

Ahora analizaremos el conjunto de datos Amazon, en este caso vemos que en cuanto a métricas de rendimiento, el modelo *BERT L4 H256 A4* es el que mejor resultados da, mejorando a nuestro modelo en un 7 % en el área bajo la curva, un 3 % en *accuracy*, 7 % en la precisión negativa, 3 % en la positiva, 29 % en el recall negativo, 1 % en el positivo, 17 % en el F1 negativo y 2 % en el positivo. Por otro lado, en el apartado temporal, vemos que el modelo *BERT* es un 1415 % más lento que el nuestro. Observando los resultados vemos que, salvo en recall y f1 negativo, nuestro modelo se acerca bastante al mejor modelo *BERT*, mientras que, como siempre, en el apartado temporal, el nuestro sale ganando por bastante, la diferencia temporal ya no es tan grande como era en el conjunto *SST2*. También podemos ver que para este conjunto de datos los 3 modelos *BERT* están bastante parejos entre sí, por lo que en este caso podría merecer la pena utilizar el modelo *BERT L2 H128 A2* que sería un 535 % más lento que el nuestro. Aunque aun así la diferencia seguiría siendo grande.

Respecto al conjunto de datos *T4SA*, aunque este es de un tamaño parecido al conjunto *Amazon*, los resultados son bastante distintos, sobre todo en el apartado temporal. Como siempre, primero compararemos el mejor modelo *BERT*, que es *BERT L4 H256 A4*, siendo que este mejora a nuestro modelo en un 4 % en el área bajo la curva, un 8 % en *accuracy*, 8 % en la precisión negativa, 9 % en la positiva, 20 % en el recall negativo, 3 % en el positivo, 14 % en el F1 negativo y 6 % en el positivo. Por otro lado, en el apartado temporal, vemos que el modelo *BERT* es un 4054 % más lento que el nuestro. Aquí vemos que los resultados temporales se disparan hasta las 11 horas-27 horas dependiendo del modelo *BERT* escogido contra apenas 41 minutos que tarda en nuestro. Por lo que para este conjunto de datos nuestro modelo hace valer sus ventajas claramente.

Por último, tenemos el conjunto de datos *Sentiment140*, el más grande con diferencia. Por lo que es lógico que este conjunto sea el que más tiempo nos ha consumido. En este caso, podemos ver que nuestro predictor no se comporta tan bien como en conjuntos de datos más pequeños. Mientras que los modelos *BERT*, si bien también se resienten en cuanto a rendimiento, logran una mejora superior que la obtenida en otros conjuntos de datos. Más concretamente, observando el mejor modelo *BERT*, que es *BERT L2 H256 A4*, vemos una mejora de un 26 % en el área bajo la curva, un 18 % en *accuracy*, 15 % en la precisión negativa, 20 % en la positiva, 21 % en el recall negativo, 14 % en el positivo, 18 % en el F1 negativo y 17 % en el positivo. Respecto al apartado temporal, vemos que este modelo tarda un 2405 % más. Siendo la diferencia de 2 horas y 52 horas muy exagerada en este caso. Aunque es cierto que, viendo que hay una mejora importante en cuanto a rendimiento, y que los 3 modelos *BERT* no son muy diferentes entre sí, para este conjunto de datos podríamos optar por el modelo *BERT L2 H128 A2* que nos proporciona una mejora respecto a nuestro modelo de un 23 % en el área bajo la curva, un 15 % en *accuracy*, 13 % en la precisión negativa, 17 % en la positiva, 19 % en el recall negativo, 13 %

en el positivo y 15 % tanto en el F1 negativo como en el positivo. Mientrás que en el apartado temporal la diferencia se reduciría a tardar un 940 % más. Aunque aun así la comparación en bruto (2 horas contra 21 horas) sigue arrojando una diferencia muy grande, por lo que, aun en el peor conjunto de datos para nuestro modelo, este demuestra que escala bien y se hace valer contra modelos mucho más complejos, pero que escalan mal.

5.4. Análisis de interpretabilidad de nuestro modelo

Tras realizar el procedimiento explicado en esta misma fase en el capítulo de experimentación, ahora ya deberíamos tener almacenados el grafo de polaridad negativo, el positivo, así como el tweet más positivo y el más negativo, vamos a representarlos en el espacio para ver como se comparan los tweets más polarizados con los grafos de polaridad.
En nuestro caso, el tweet más positivo, una vez realizado el prepocesamiento, será el siguiente: *"good morning everyone beautiful day new england"*, y al emplear distancia 1, el grafo que representa nuestro tweet va a tener las siguientes aristas:

```
good – morning
morning – everyone
everyone – beautiful
beautiful – day
day – new
new – england
```

Mientras que el tweet más negativo, también tras ser preprocesado, será *"jogging really cool especially got fever"* y el grafo que lo representa contendrá las siguientes aristas:

```
jogging – really
really – cool
cool – especially
especially – got
got – fever
```

Primero, en la figura 5.1 tenemos el grafo de polaridad negativo representado. Por un lado, vemos que hay nodos más grandes que otros, lo mismo pasa con las aristas. El tamaño de estos nodos estará determinado por el valor calculado de *eigen vectors*, mientras que el tamaño de las aristas lo marcará el valor de *edge betweenness*. Por otro lado, vemos que además hay nodos rojos, verdes y negros. Los vértices rojos son aquellos que están en el grafo de polaridad correspondiente y además están en el tweet más negativo seleccionado, los verdes son aquellos que están en el grafo de polaridad y en el tweet más positivo. Mientras que los negros son aquellos nodos del grafo de polaridad que no están presentes en ningunos de ambos tweets. Además, vemos que aquellos nodos más importantes, se sitúan en el centro del grafo, y en nuestro caso, podemos ver que entre todas las palabras del tweet negativo y las del positivo, los dos nodos más grandes son dos rojos, es decir, pertenecen al tweet negativo. Al estar en el centro, también vemos que muchas aristas pasan por dichos

nodos(figura 5.2) por lo que así se justifica la importancía tanto en términos de *eigen vectors* como en términos de *edge betweenness*.

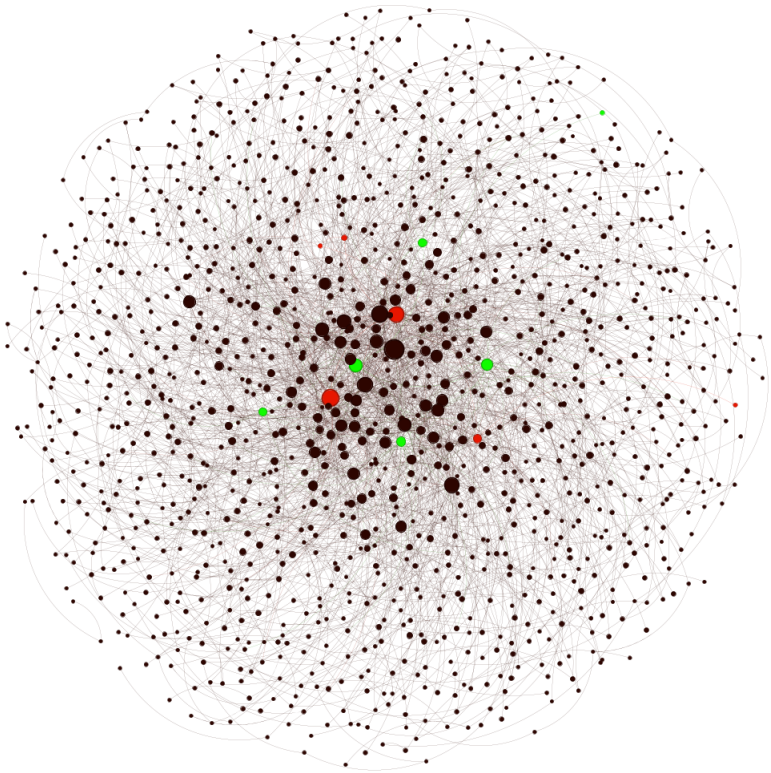

Figura 5.1: Representación del grafo de polaridad negativo

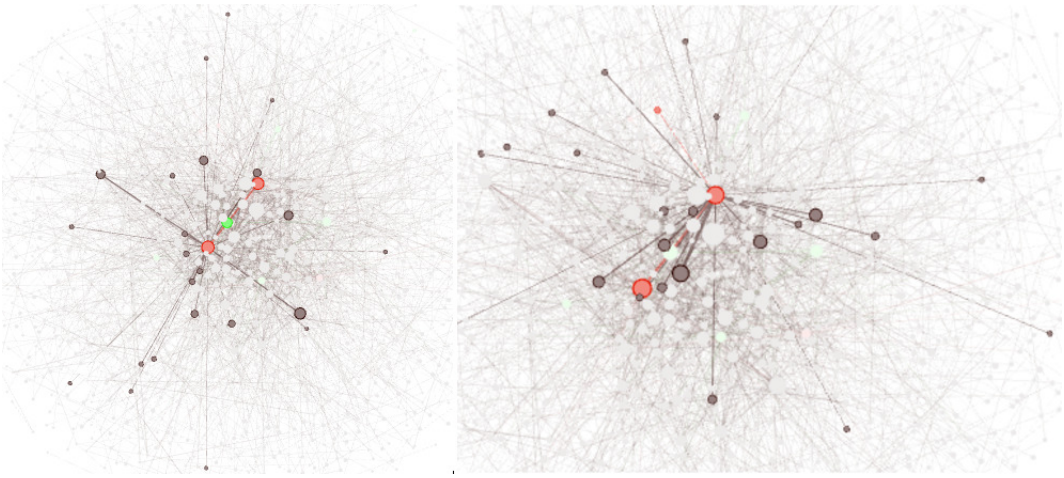

Figura 5.2: Aristas incidentes a los nodos negativos más importantes

Segundo, en la figura 5.3 tenemos el grafo de polaridad positivo representado. Siguiendo el mismo esquema que en el caso anterior, vemos que los nodos verdes se sitúan en su mayoría en el centro y con un gran tamaño. Lo que significa que el tweet seleccionado tiene palabras que son muy importantes en el grafo de polaridad positiva. Por lo que dicho tweet positivo queda muy bien representado en el grafo de polaridad. Además de nuevo, en la figura 5.4 podemos ver que los nodos positivos más importantes tienen muchas aristas incidentes, demostrando de nuevo una gran importancia. Por otro lado, vemos que los nodos rojos, los negativos, tienen muy poca importancia en el grafo tanto en términos de *eigen vectors* como en términos de *edge betweenness*

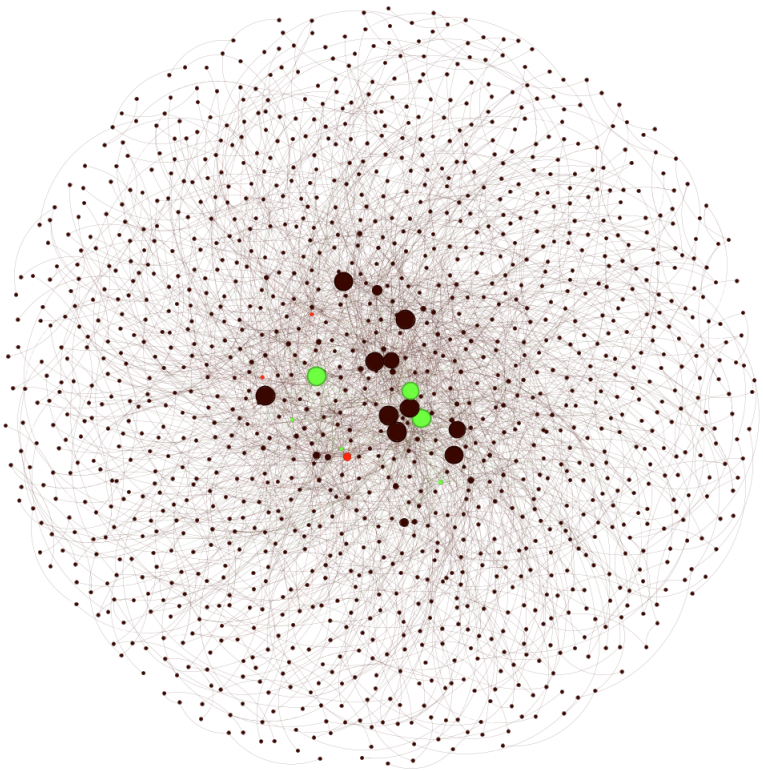

Figura 5.3: Representación del grafo de polaridad positivo

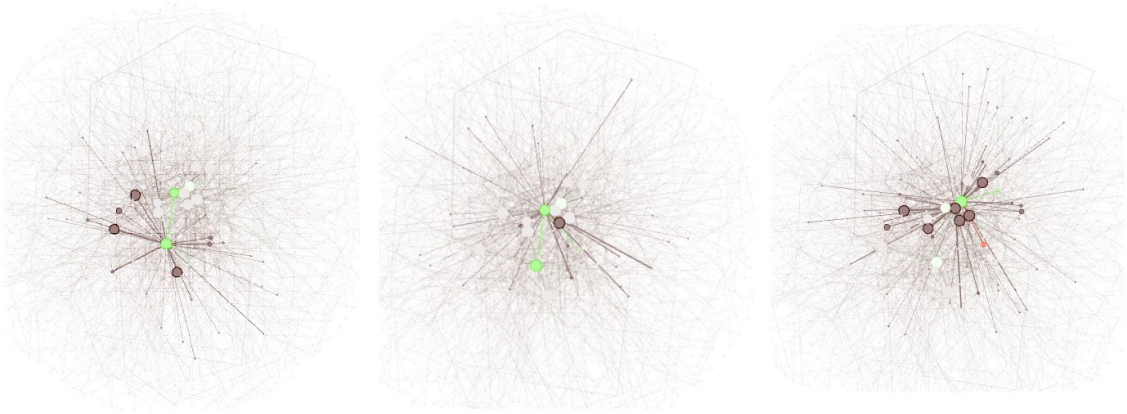

Figura 5.4: Aristas incidentes a los nodos positivos más importantes

Capítulo 6

Conclusiones y trabajo futuro

Para este último apartado, vamos a ver las conclusiones finales, así como el trabajo futuro que hemos pensado que podría ampliar este TFG de aquí en adelante.

6.1. Conclusiones

La conclusión principal es que se han cumplido los dos objetivos principales que planteamos en la introducción, que si recordamos estos eran que el modelo debía ser simple e interpretable.
Por un lado, en cuanto a la simplicidad, la idea era obtener un modelo sencillo, rápido, y que se acercase en términos de rendimiento a otros modelos más complejos. Nuestro modelo efectivamente es sencillo, ya que es una red neuronal MLP con una arquitectura bastante simple. Además, también hemos visto en la fase de experimentación que los resultados obtenidos con nuestro modelo son bastante buenos y que se acercan en mayor o menor medida a los obtenidos con modelos más sofisticados *BERT*. El único conjunto de datos para el cual nuestro modelo es sustancialmente peor es quizás el de Sentiment140, el más largo de todos. Pero para los otros cuatro, como decimos, nos acercamos bastante. Además, la diferencia temporal entre los modelos BERT y los nuestros son muy grandes. Por lo que en ese aspecto hemos obtenido grandes ganancias tal y como pretendíamos.
Por otro lado, vemos que también se ha cumplido el objetivo de que el modelo sea interpretable con facilidad. Esto es así, ya que, efectivamente, el modelo, además de tener una arquitectura sencilla, también tiene unos inputs y un output claro. Es sencillo de entender que, las entradas son un simple array *X TRAIN* y otro *Y TRAIN*. Esto contrasta con otros modelos con un proceso de entrada/salida más complejo, incluso en los que se combinan más de un modelo internamente. En nuestro caso, cada valor de *X TRAIN* se corresponde a un único textp, y consiste en 8 métricas de centralidad que son fáciles de entender, mientras que, por otro lado, el array *Y TRAIN* nos dice claramente la polaridad del texto. Por último, para terminar con esta parte de la interpretabilidad, también es importante mencionar que nuestra red MLP es facilmente modulable, pudiéndose cambiar la función de activación, el *solver*, la arquitectura, etc.

Ahora, también tenemos que hablar de las limitaciones que nos hemos encontrado en nuestro trabajo. Para ello, dividiremos esto en dos partes. Primero, las limitaciones encontradas con nuestros modelos, y, por otro lado, respecto a los modelos de

lenguaje *BERT*.

Primero, en nuestro modelo nos hubiera gustado probar con conjuntos de datos más grandes y con textos más grandes. Sin embargo, nuestro modelo ha demostrado que funciona mucho mejor para texto corto, y que además para conjuntos de datos especialmente grandes como sentiment140 no funciona también. Además, para un conjunto de datos de millones de datos, la computación de los grafos de polaridad, probablemente no sería soportada por un computador sencillo de andar por casa, por lo que romperíamos una de las premisas que establecíamos en los objetivos. Por lo que se podría decir en resumen que nuestro modelo funciona mejor para textos cortos y para conjuntos de datos iguales o inferiores a 1.000.000 muestras.
Segundo, en los modelos BERT nos hubiese gustado hacer pruebas con modelos más complejos, como *Electra* o *Albert*. Sin embargo, ya hemos visto que estos modelos, aunque funcionan muy bien, escalan muy mal en terminos temporales, y además cuanto más complejo sea más posibilidades hay de que un computador sencillo no lo aguante o tarde demasiado. También, con los tres modelos *BERT* utilizados nos hubiera gustado probar los 5 fold para los conjuntos de datos T4SA, Amazon y Sentiment140, sin embargo, ya hemos visto en los resultados de las pruebas que un simple fold puede tardar un día o más en computarse correctamente. Y eso, como dijimos antes, teniendo en cuenta que los modelos *BERT* solo utilizan 5 épocas para el entrenamiento, por lo que la escalabilidad de estos modelos es difícil de asumir para un computador cualquiera

6.2. Líneas de trabajo futuro

Para acabar este capítulo, vamos a hablar sobre el trabajo futuro que se cree que sería interesante para ser añadido a este TFG de aquí en adelante. Para esto, hemos pensado en dos ampliaciones del TFG.

6.2.1. Análisis de sentimiento a partir de texto e imágenes

Para empezar con la primera, recordemos que nuestro modelo trabaja con conjuntos de datos cuyos inputs son textuales. Y nos gustaría que nuestro modelo fuese también capaz de tratar con imágenes, interpretarlas y decidir también mediante imágenes la polaridad de un texto. Esto es así ya que creemos que limitarse a opiniones textuales hoy en día es insuficiente y que las imágenes pueden tener un valor expresivo muy grande(muchas veces se dice que una imagen vale más que mil palabras).
Un ejemplo de como funciona esto lo hemos extraído de la web T4SA Vadicamo et al. (2017), que es de donde también obtuvimos el conjunto de datos T4SA, en este caso los autores Lucia Vadicamo, Fabio Carrara, Andrea Cimino, Stefano Cresci, Felice Dell'Orletta, Fabrizio Falchi y Maurizio Tescon, proponen el siguiente esquema:
Como podemos ver en la figura 6.1, se parte de tweets híbridos que contienen tanto palabras como imágenes, en este caso con un mínimo de 5 palabras y una imagen, aunque eso sería variable. Por un lado, el texto sí estaría inicialmente etiquetado, pero las imágenes no. Sin embargo, el modelo debería ser capaz, a través del texto, de etiquetar las imágenes con una alta confianza y al final del todo tener un conjunto de

datos con texto e imágenes etiquetadas que luego será mucho más fácil de interpretar y predecir para futuras ejecuciones.

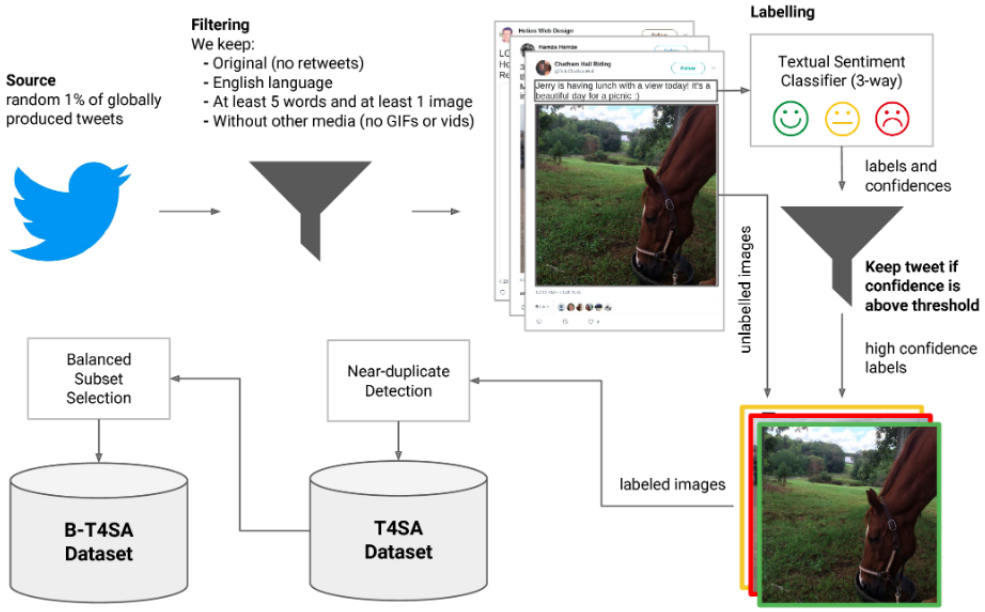

Figura 6.1: Esquema de tratamiento de tweets que incluyen imagenes Vadicamo et al. (2017)

6.2.2. Análisis de sentimiento mediante movimientos oculares

Por último, la otra sería también incluir información obtenida mediante *eyetracking*Wang et al. (2022) para saber la reacción de un usuario simplemente mediante el movimiento de sus ojos, hacia algo que queramos medir. Ya sea un producto, una imagen, un texto, etc. Esta parte ya se ha explicado en el apartado del estado del arte, por lo que hemos podido conocer ya su funcionamiento y hay que decir que creemos que tiene un gran potencial. Además, tal y como vemos en la figura 6.2, esta técnica se puede combinar con otras anteriormente vistas como pueden ser los modelos de incrustación, los modelos de atención o los modelos de interacción de la información basada en grafos.
En el caso de nuestro trabajo, la inclusión de estas técnicas de seguimiento ocular nos permitiría enriquecer mucho nuestro modelo, ya que además de predecir con mayor precisión, también nos permitiría encontrar patrones en la reacción de los usuarios ante un tweet positivo y la reacción ante uno negativo. La extracción de dichos patrones nos permitiría también predecir con más facilidad y ayudaría enormemente a la explicabilidad del modelo.
Por último, creemos que este es un campo que aún no se ha explorado, tanto como la clasificación de texto e imágenes que es más común. Y que puede tener un impacto importante en nuestro campo porque nos permite incorporar la psicología a

nuestro modelo, además de que también es útil en otros campos como puede ser las interfaces de usuario.

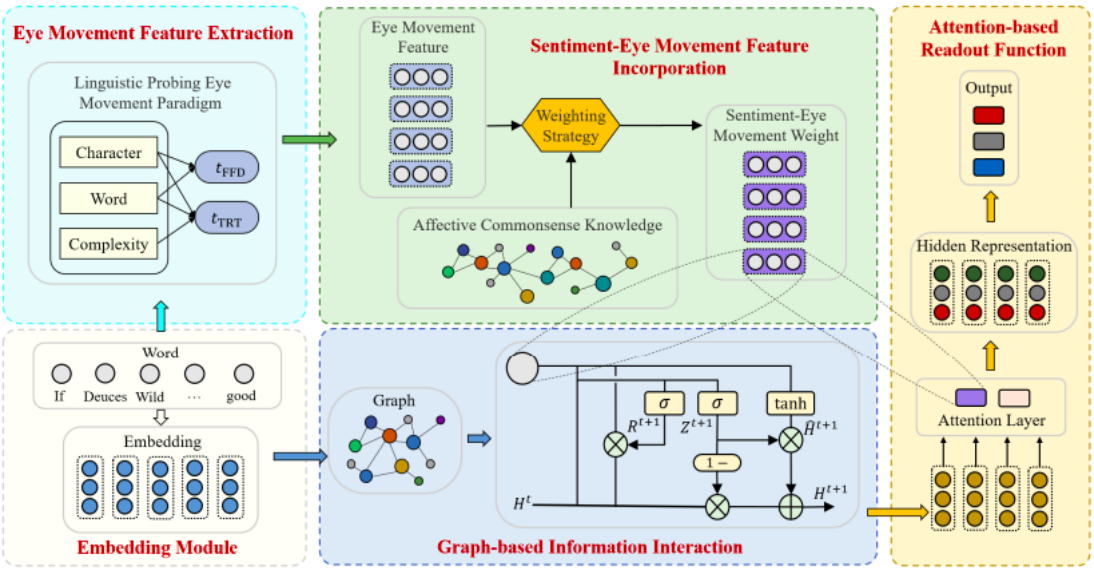

Figura 6.2: Esquema de tratamiento de tweets que incluyen imagenes Vadicamo et al. (2017)

Anexo

En este capítulo adjuntamos el código desarrollado para la realización del trabajo.
Repositorio github con el código

Referencias

admin@graphonline.ru (2015). Graphonline.

Basiri, M. E., Nemati, S., Abdar, M., Cambria, E., and Acharya, U. R. (2021). Abcdm: An attention-based bidirectional cnn-rnn deep model for sentiment analysis. *Future Generation Computer Systems*, 115:279–294.

Bastian, M., Heymann, S., and Jacomy, M. (2009). Gephi: An open source software for exploring and manipulating networks.

Crowdflower (2015). Twitter us airline sentiment.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Dey, R. and Salem, F. M. (2017). Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE.

everywhere, G. (2022). Algoritmos de centralidad de vector propio.

Everywhere, G. (2022). ¿qué es un knowledge graph?

Itelligent (2017). Análisis de sentimiento, ¿qué es, cómo funciona y para qué sirve?

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Lovera, F. A., Cardinale, Y. C., and Homsi, M. N. (2021). Sentiment analysis in twitter based on knowledge graph and deep learning classification. *Electronics*, 10(22):2739.

Lu, L. and Zhang, M. (2013). *Edge Betweenness Centrality*, pages 647–648. Springer New York, New York, NY.

Nigro, H. (2020). Una revisión a la minería de opiniones y los retos del pnl. *Revista Ingeniería, Matemáticas y Ciencias de la Información*, 7(13):105–110.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Stanford (2010). Sentiment140.

Stanford (2012). Amazon fine food reviews.

Staudemeyer, R. C. and Morris, E. R. (2019). Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*.

Tensorflow (2022a). Clasificar texto con bert.

Tensorflow (2022b). Herramientas de procesamiento de texto para tensorflow.

Turc, I., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962v2*.

Vadicamo, L., Carrara, F., Cimino, A., Cresci, S., Dell'Orletta, F., Falchi, F., and Tesconi, M. (2017). Cross-media learning for image sentiment analysis in the wild. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 308–317.

Wang, B., Liang, B., Du, J., Yang, M., and Xu, R. (2022). Semgraph: Incorporating sentiment knowledge and eye movement into graph model for sentiment analysis. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7521–7531.