uc3m | Universidad Carlos III de Madrid

Doble Grado en Ingeniería Informática y Administración y Dirección de Empresas

2019-2020

Trabajo Fin de Grado

"Procesamiento del lenguaje natural con BERT: Análisis de sentimientos en tuits"

Iago Collarte González

Tutor/es

Miguel Ángel Patricio Guisado

Antonio Berlanga de Jesús

Colmenarejo, 16 de julio de 2020

ÍNDICE

Índice 5
Índice de figuras 6
1.Introducción 1
1.1 Motivación 1
1.2 Objetivos 2
1.3 Medios utilizados 3
1.4 Marco regulador 3
1.5 Entorno socioeconómico 3
1.6 Estructura 4
2.Estado del arte 6
2.1 Algoritmos que resuelven el mismo problema 6
2.2 ¿Por qué se usa BERT? 12
2.3 Tecnologías y Frameworks con los que se ha trabajado 22
*3.*Experimentación 26
3.1 Descripción del *dataset* 26
3.2 Procesado de los datos 30
3.3 Algoritmos utilizados 31
3.4 Experimentación 43
4.Conclusiones y futuros trabajos 55
4.1 Conclusiones 55
4.2 Futuros trabajos 56
5.Bibliografía 57
6.Anexos 61
6.1 Planificación 61
6.2 Presupuesto 62

ÍNDICE DE FIGURAS

Tabla 1 Aplicaciones del análisis de sentimientos (Traducción) (D'Andrea, Ferri, Grifoni, & Guzzo, 2015)..... 1
Figura 1 Arquitectura del modelo Skip-gram (Mikolov et al., 2013)..... 6
Figura 2 Arquitectura (simple) del modelo CBOW. (Kulshrestha, 2019) 7
Figura 3 Probabilidades de coocurrencia para las palabras hielo y vapor para un corpus con seis mil millones de palabras. (Pennington, Socher & Manning, 2014)..... 8
Figura 4 Diagrama simplificado de las transformaciones aplicadas a cada token antes de pasar a LSTM. (Mihail, 2018)..... 9
Figura 5 Fragmento de modelo equivalente a ELMo. (Kim, Jernite, Sontag, & Rush, 2015) 9
Figura 6 Fases de ULMFiT (Howard & Ruder, 2018) 11
Figura 7 Arquitectura del modelo de transformador (Vaswani et al., 2017)..... 13
Figura 8 Detalle del “encoder block” (Vaswani et al., 2017)..... 13
Figura 9 Diagrama de Multi-head attention (Vaswani et al., 2017)..... 14
Figura 10 Detalle del “decoder block” (Vaswani et al., 2017)..... 14
Figura 11 Normalización por lotes y normalización por capas (Zhang & Zhang, 2019)..... 15
Figura 12 Comparativa de bidireccionalidad entre BERT, GPT y ELMo. BERT es profundamente bidireccional, ELMo es bidireccional con poca profundidad y GPT es unidireccional (Devlin & Chang, 2018). 16
Figura 13 Contexto para la palabra “bank” donde la izquierda y la derecha cambian el significado (Rizvi, 2019). 16
Figura 14 Arquitectura de BERT ilustrada (Alammar, 2018) 18
Figura 15 Representación del input de BERT (Devlin, Chang, Lee, & Toutanova, 2018) 18
Figura 16 Diagrama del pre-training de BERT (Devlin, Chang, Lee, & Toutanova, 2018). 19
Figura 17 Resultados para el benchmark GLUE (Devlin, Chang, Lee, & Toutanova, 2018)..... 20
Figura 18 Resultados en SQuAD v1.1 (Devlin, Chang, Lee, & Toutanova, 2018). 21
Figura 19 Resultados en SQuAD v2.0 (Devlin, Chang, Lee, & Toutanova, 2018). 21
Figura 20 Resultados obtenidos en SWAG (Devlin, Chang, Lee, & Toutanova, 2018). 21
Figura 21 Diagrama de bert-as-service (Xiao, 2019) 23
Figura 22 Arquitectura de bert-as-service (Xiao, 2019). 24
Figura 23 Salida de código que muestra el contenido del dataset 26
Figura 24 Gráfico de frecuencias de los sentimientos. (elaboración propia) 28
Figura 25 Comando para lanzar el cliente de bert-as-service..... 31
Figura 26 Variación en el rendimiento en función del valor de pooling_layer (Xiao, 2019)..... 32
Figura 27 Fórmula de probabilidad teorema de Bayes (Estadística de la probabilidad: Teorema de bayes) 33
Figura 28 Fórmula para toma de decisión BernoulliNB de scikit-learn (scikit-learn)..... 34
Figura 29 Fórmula para el cálculo de la probabilidad usado por GaussianNB (scikit-learn)..... 35
Figura 30 Código que importaba el tokenizer del modelo multilingüe de BERT en la prueba realizada 43
Figura 31 Configuración del modelo de Bert basado en la librería Transformers. 43
Figura 32 Obtención de dirección IP asociada al Notebook de Google Colab y fallo del intento de comunicación. 44
Figura 33 Salida cuando se lanza bert-serving-server con éxito 44

Figura 34 Ilustración matriz de confusión (Lahby Mohamed) 45
Figura 35 Comparativa de resultados para el modelo pre entrenado de Google (izquierda) y el que ha pasado por el ajuste (derecha)..... 46
Figura 36 Comando para lanzar un modelo ajustado en bert-as-service 46
Figura 37 Función para la división del texto en muestra lotes 47
Figura 38 Llamada a bert-as-service para que se realice el encoding..... 47
Figura 39 Salida de classification report y matriz de confusión..... 47
Figura 40 Resultados para BernouilliNB (izquierda) y GaussianNB (derecha) 48
Figura 41 Resultados para random forest con 15 árboles 49
Figura 42 Resultados regresión logística. 49
Figura 43 Resultados KNeighbordClssifier 50
Figura 44 Prueba sobre el nuevo dataset de reviews de películas 51
Figura 45 Resultados neutral contra el resto 51
Figura 46 Resultados none contra el resto 51
Figura 47 Resultados negativo contra el resto..... 52
Figura 48 Resultados positivo contra el resto..... 52
Figura 49 Análisis utilizando sólo none y neutral..... 53
Figura 50 Comparativa entre negativo y positivo. 53
Figura 51 Diagrama de Gantt del trabajo..... 61

1. INTRODUCCIÓN

1.1 Motivación

"El lenguaje natural (LN) es el medio que utilizamos de manera cotidiana para establecer nuestra comunicación con las demás personas" (Cortez Vasquez, Vega Huerta, & Pariona Quispe, 2009). La comprensión de este lenguaje en haciendo uso de máquinas (ordenadores) y entre otros campos la inteligencia artificial se conoce como procesamiento del lenguaje natural (PLN). Uno de los campos dentro del procesamiento del lenguaje natural es el análisis de sentimientos, el objetivo del análisis de sentimientos consiste en la obtención de información a partir de la computación de los sentimientos, y opiniones que se pueden encontrar en textos.

Dentro del análisis de sentimientos podemos diferenciar entre distintos tipos de análisis: el primero sería la polaridad, este análisis se centra en determinar si se trata de algo positivo, negativo o neutral; dentro de este análisis se puede añadir una mayor granularidad al incluir muy positivo o negativo. La intención también es sujeto de análisis, en este caso el análisis tratará de determinar, por ejemplo, el interés de una persona. Finalmente se puede hacer análisis de la emoción, en este caso se trata de determinar la emoción puntual transmitida, ya sea alegría o tristeza, por ejemplo. Todos estos tipos de análisis pueden tener gran interés desde un punto de vista empresarial, político o social.

A pesar de todo, el análisis de sentimientos sigue teniendo que hacer frente a diversas dificultades, entre ellas se encuentran la definición de neutral, el tono del lenguaje natural o el sarcasmo.

El análisis de sentimientos presenta diferentes posibles aplicaciones, entre las que destacan la reputación de las marcas, el comercio en línea, sistemas de transporte inteligente…

Tabla 1 Aplicaciones del análisis de sentimientos (Traducción) (D'Andrea, Ferri, Grifoni, & Guzzo, 2015)

Aplicaciones del Análisis de Sentimientos
Negocios
Voz del consumidor
Reputación de marcas
Publicidad *online:* Publicidad contextual centrada en *bloggers* Publicidad *online* centrada en insatisfacción
Comercio en línea
Política

Brújulas de voto
Clarificación de posiciones de políticos
Acciones públicas
Monitorización de eventos
Blogs de asuntos legales
Proposiciones de ley
Sistemas de transporte inteligente

En 2018 se publica el estudio *"BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*" (Devlin, Chang, Lee, & Toutanova, 2018) abriendo un nuevo horizonte de posibilidades en el campo del procesamiento del lenguaje natural, al introducir al modelado del lenguaje el entrenamiento bidireccional de transformadores.

BERT, abre a su vez con sus diferentes modelos disponibles la posibilidad de aplicar sus avances en diferentes idiomas, esto permite aplicar la tecnología publicada por los investigadores de Google al español. En conjunción con su capacidad para el análisis de distintos idiomas, las posibilidades que ofrece BERT con su codificación al leer secuencias enteras, a diferencia de los modelos bidireccionales tradicionales, pudiendo conocer mejor el contexto de la palabra.

Finalmente, Twitter presenta un entorno ideal para el análisis de sentimientos al poner a disposición del público gran cantidad de mensajes no estructurados, donde se expresan opiniones y experiencias de las personas. Hay que tener presente que en Twitter están presentes gran parte de los problemas que soporta el análisis de sentimientos, sarcasmo, ironía y emojis hacen que el análisis de sentimientos en la red social sea más complicado.

1.2 Objetivos

Por lo que respecta a los objetivos que se han perseguido con el trabajo realizado, estos se pueden dividir en función de si están centrados en el análisis de sentimientos o en comprender las capacidades de BERT.

En primer lugar y por lo que respecta al análisis de sentimientos, el objetivo es realizarlo sobre tuits en español, además el análisis deberá ser multi etiqueta sobre el *dataset* que ha sido proporcionado.

Dentro del análisis de sentimientos se divide en dos partes, analizar con éxito en español y hacerlo sobre tuits que presentan sus particularidades.

Por otro lado, se encuentran los objetivos que respectan a BERT, en primer lugar, se quiere analizar la viabilidad de BERT para aplicarlo al procesamiento del lenguaje natural en español, y en segundo lugar la facilidad o dificultad que entraña hacer uso de la herramienta, analizando sus capacidades y distintas formas de trabajar con la misma.

1.3 Medios utilizados

Para la realización del trabajo se utiliza un ordenador en el que se ha instalado Anaconda. Para que fuese posible la utilización de BERT ha sido necesario instalar en el ordenador un cliente VPN, un cliente para FTP y también Putty para poder establecer una conexión con el servidor que estaba alojando BERT.

Por lo que respecta a conseguir que funcione BERT, ha sido necesario hacer uso de un servidor de la universidad Carlos III de Madrid que cuenta con una tarjeta gráfica. Además, para algunas pruebas realizadas con BERT ha sido necesario hacer uso de Google Colab y de Google Cloud Storage.

1.4 Marco regulador

En este apartado el objetivo es observar las regulaciones que pueden tener un impacto tanto en el trabajo realizado como en futuros trabajos que se quieran realizar con el fin de avanzar sobre este.

El trabajo realizado se enmarca principalmente bajo el marco del Reglamento General de Protección de Datos (RGPD) así como las licencias del *software* utilizado. Inicialmente, todo el *software* puede ser utilizado de manera gratuita por lo que no supone ningún problema.

El otro aspecto que se puede ver afectado por regulaciones, más en concreto por el RGPD mencionado con anterioridad es el *dataset* que se ha utilizado. En este caso, el *dataset* consiste en tuits que están disponibles al público y ha sido obtenido de Analytics Vidhya abierto a cualquiera que se registre, dentro del *dataset* no hay información que pueda identificar a ningún individuo, ya sea su nombre o su dirección IP, por lo que no se encuentra restringida por el RGPD.

1.5 Entorno socioeconómico

Por lo que respecta al entorno socioeconómico, el procesamiento del lenguaje natural y el análisis de sentimientos se encuentran en el centro de atención tanto para el mundo

empresarial como para el mundo de la investigación. La mayor parte de las empresas empiezan a poner su foco en la inteligencia artificial y en concreto en procesamiento del lenguaje natural y análisis de sentimientos, especialmente en mundos como el marketing.

El análisis de la información y conseguir convertir esta en conocimiento actuable que ayuda a las empresas en la toma de decisiones genera valor en las compañías, independientemente de su tamaño, por lo que las tecnologías exploradas y sus posibles aplicaciones podrán encontrarse en el mundo empresarial en un futuro cercano sino en la actualidad.

También se debe tener en cuenta que en la actualidad BERT ya está teniendo un gran impacto en el mundo empresarial, BERT ya está siendo utilizado por Google en su motor de búsqueda, y esto no sólo afecta a los resultados que podemos encontrar en las búsquedas, sino que afecta a cómo las empresas aplican técnicas de optimización de motor de búsqueda para que sus empresas tengan mayor presencia.

Por último, se ha podido comprobar durante la pandemia de COVID-19 la importancia de las herramientas de análisis y de la inteligencia artificial, de modo que permita tener mayor conocimiento del tráfico de personas, reconocimiento facial, indicar si se ha tenido contacto con personas contagiadas… esta situación dramática y excepcional ha devuelto a la inteligencia artificial y las posibilidades que esta ofrece al centro de atención de la sociedad.

Por lo que respecta al aspecto económico de replicar lo realizado en este trabajo en un mundo empresarial en un momento de crisis económica en la que la mayoría de los países afrontan una recesión, se puede hacer con bajos costes económicos, principalmente estarán relacionados con alquilar un servidor con una tarjeta gráfica. Por tanto, poder aplicar las técnicas del trabajo en el mundo real es posible hasta en la compleja situación actual.

1.6 Estructura

A lo largo de los siguientes capítulos se hablará del estado del arte, los datos utilizados, descripciones de los algoritmos utilizados, la experimentación realizada y finalmente unas conclusiones.

Dentro del estado del arte se hablará de los algoritmos que resuelven el mismo tipo de problemas, se justificará la selección de los algoritmos que se han utilizado y por último las tecnologías y *frameworks* con los que se realiza el trabajo.

La exploración del *dataset* se centrará en dos aspectos distintos: en primer lugar, se procederá a la descripción del *dataset* que se ha tenido que utilizar y sus características y, en segundo lugar, cómo se ha procesado el *dataset* para poder realizar el análisis.

En el siguiente capítulo se analizarán los algoritmos utilizados, se entrará también en los diferentes algoritmos utilizados para la clasificación. También se describirá la experimentación que se ha realizado en el análisis sobre el dataset, los resultados y las conclusiones que se han podido extraer.

Por último, en las conclusiones se hablará de los objetivos marcados y si se ha conseguido alcanzar dichos objetivos para, finalmente, hablar de posibles trabajos futuros para poder ampliar sobre el trabajo realizado.

2. ESTADO DEL ARTE

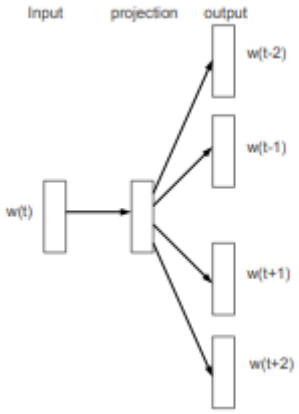
2.1 Algoritmos que resuelven el mismo problema

En este apartado se hablará de todas las posibles alternativas a BERT para la codificación de los tuits. Por tanto, se procederá a analizar los distintos frameworks de procesamiento del lenguaje natural disponibles en la actualidad, desde Word2Vec pasando por GloVe, ELMo, ULMFiT y OPENAI GPT.

Word2Vec: desarrollado en 2013 por Tomas Mikolov en Google, obtiene representaciones numéricas de palabras (*word embeddings*), se trata de una combinación de dos métodos distintos: *Continious Bag of Words* (CBOW) y *Skip-gram model*.

Skip-gram model se basa en que las representaciones distribuidas de palabras en un vector ayudan a los algoritmos de aprendizaje a lograr mejores resultados (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013). El modelo Skip-gram ofrece un método eficiente para el aprendizaje de representaciones vectoriales de palabras de alta calidad de grandes cantidades de textos desestructurados. (Mikolov et al., 2013).

Figura 1 Arquitectura del modelo Skip-gram (Mikolov et al., 2013)

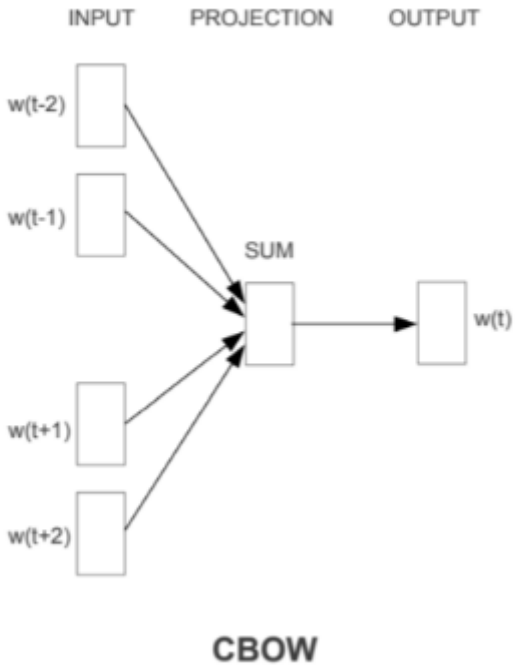


El objetivo del entrenamiento del modelo Skip-gram es encontrar representaciones de palabras que sean útiles para predecir las palabras que rodean en una oración o documento (Mikolov et al., 2013)

Continious bag of words: utiliza el vocabulario del documento o *dataset*, para cada palabra se cuenta el número de veces que aparece. El método trata de predecir una palabra a partir de su contexto.

De manera similar a Skip-gram se utiliza cada par de palabras para enseñar al modelo que suceden al mismo tiempo, sin embargo, en lugar de sumar los errores se añaden las *input words* para la misma palabra objetivo (Kulshrestha, 2019).

Figura 2 Arquitectura (simple) del modelo CBOW. (Kulshrestha, 2019)



El objetivo que se persigue al utilizar Word2Vec es la agrupación de vectores de palabras similares en el espacio vectorial. Esto es que se detectan similitudes matemáticamente. Word2Vec crea vectores que son representaciones numéricas distribuidas de apariciones de palabras, apariciones como el contexto de palabras individuales (Nicholson, n.d.)

Si se le da suficiente información a Word2Vec este puede realizar predicciones bastante precisas del significado de una palabra que permiten establecer relaciones con otras palabras (Nicholson, n.d.).

La similitud entre palabras se mide mediante similitud coseno, donde la similitud se representa como el valor del coseno en función de los grados, si no existe ninguna similitud entonces será el coseno de noventa grados y en caso de ser idénticas las palabras sería el coseno de cero grados (Nicholson, n.d.).

Con el fin de hacer que Word2Vec sea más eficiente a nivel de computación es necesario que se apliquen además *Hierarchical Softmax* y *Skip-gram Negative Sampling*. *Hierarchical Softmax* utiliza un árbol binario para representar todas las palabras del

vocabulario. Se considera que si hay V hojas entonces tiene que haber V-1 unidades internas. Para cada unidad hoja tiene que haber un camino único desde la raíz hasta la hoja, este camino se utiliza para calcular la probabilidad de la palabra representada por la hoja. En *hierarchical softmax* no existe un vector representación de palabras como salida, en su lugar, para cada una de las V-1 unidades internas existe un vector de salida y la probabilidad de la palabra se define como una función de ese vector (Rong, 2014).

Por lo que respecta a *Skip-gram negative sampling*, para hacer frente al problema que supone tener un gran número de vectores de salida por cada iteración, se opta por actualizar solamente una muestra de los vectores. La palabra (muestra positiva) debe mantenerse en la muestra y actualizarse, debiéndose introducir una serie de muestras negativas. Una distribución probabilística es necesaria para determinar el funcionamiento del proceso de muestreo, esta distribución se conocerá como distribución de ruido. En wor2vec, en lugar de producir una distribución multinomial posterior bien definida, los autores sostienen que el objetivo de entrenamiento simplificado utilizado es capacidad de producir *word embeddings* de alta calidad (Rong, 2014).

GloVe: "las estadísticas de ocurrencias de palabras en un corpus son la principal fuente de información disponible para todos los modelos de aprendizaje no supervisado de representación de palabras, […], la cuestión sigue siendo cómo se genera significado a partir de las estadísticas y cómo el vector de palabras generado representa ese significado." (Pennington, Socher & Manning, 2014). El desarrollo de GloVe tiene como objetivo dar claridad sobre este problema al ser capturadas de manera directa por el modelo las estadísticas globales del corpus.

Figura 3 Probabilidades de coocurrencia para las palabras hielo y vapor para un corpus con seis mil millones de palabras. (Pennington, Socher & Manning, 2014)

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

GloVe se construye entorno a la idea de que se pueden extraer relaciones semánticas de la matriz de coocurrencia. GloVe utiliza tanto estadísticas globales como locales para poder obtener una función de pérdida que utilice ambas estadísticas.

Para poder pasar de una medida a vectores de palabras es necesario tener una ecuación predecir los ratios de coocurrencia utilizando los vectores de palabras. Otro problema al

que se debe hacer frente es que se debe pasar de un vector a un escalar y por último se debe pasar de las tres entidades que se utilizan en GloVe (las dos palabras para las que se quiere determinar la coocurrencia y otra tercera palabra "palabra sonda") a solamente dos para simplificar la función de pérdida (Pennington, Socher & Manning, 2014).

ELMo (E*mbedding from Language Models*): los *embeddings* aprendidos de modelos de lenguaje neuronal a gran escala puede resultar efectivo para modelos de transferencia del aprendizaje semi supervisado. Si se añaden los vectores de ELMo a una tarea de procesamiento del lenguaje natural se podrán apreciar mejoras en los resultados (Mihail, 2018).

La arquitectura de ELMo se inicia con el entrenamiento de un modelo de lenguaje basado en una red neuronal, en el caso de ELMo la base es un LSTM bidireccional de dos capas, entre ambas capas se añade una conexión residual, esta capa permitirá obtener mejores resultados en el entrenamiento. Posteriormente para cada letra de cada palabra se busca un *carácter embedding*, los *embeddings* se pasan por una capa convolucional con una serie de filtros y por una capa de *max-pool*; finalmente, la representación se pasa una *highway network* de dos capas antes de ser transferido a las 2 capas mencionadas con anterioridad (Mihail, 2018).

Figura 4 Diagrama simplificado de las transformaciones aplicadas a cada token antes de pasar a LSTM. (Mihail, 2018)

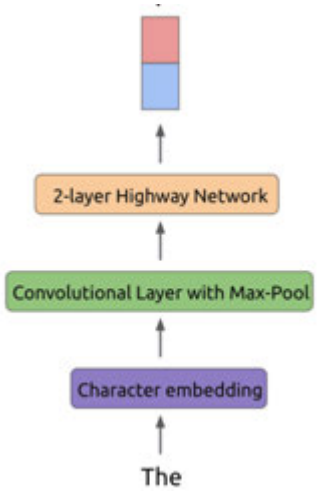
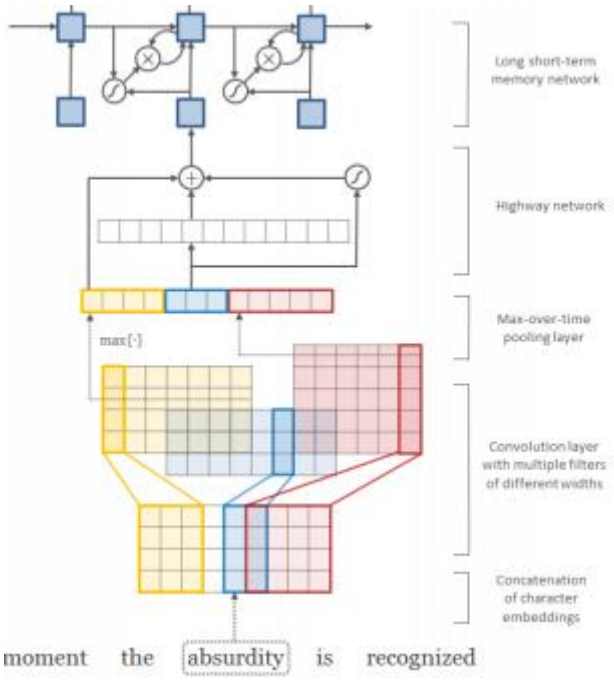


Figura 5 Fragmento de modelo equivalente a ELMo. (Kim, Jernite, Sontag, & Rush, 2015)



Como se puede ver por lo explicado con anterioridad, ELMo no introduce innovaciones en el modelo de arquitectura, todo lo que se usa en ELMo había aparecido con anterioridad en la literatura, sin embargo, la innovación introducida por ELMo es cómo usa el modelo una vez ha sido entrenado (Mihail, 2018).

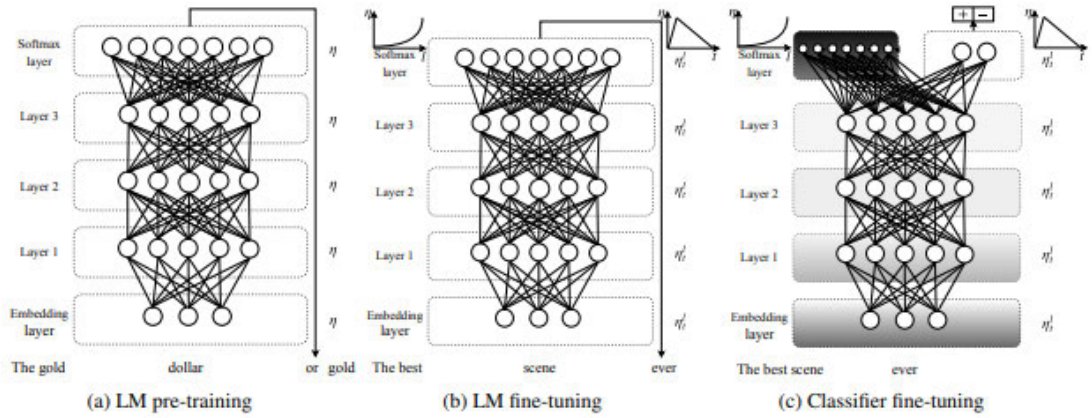
Las ventajas de las transformaciones que realiza ELMo son varias, en primer lugar, al utilizar *character embeddings* puede apreciar características morfológicas que si se aplica a nivel de palabra no es posible; a su vez el trabajo a nivel de caracter permite trabajar con palabras, aunque estas puedan estar fuera del vocabulario. Los filtros convolucionales permiten seleccionar características de los n-gramas que mejoran las representaciones que se realizan (Mihail, 2018).

Por lo que respecta al entrenamiento y al uso de ELMo, se puede mencionar que se ha entrenado con el *dataset* de *1B Word Benchmark*. Por otro lado, las capas de LSTM tienen 4096 unidades y la transformación de la entrada tiene 2048 filtros convolucionales (Mihail, 2018).

ULMFiT: el desconocimiento de cómo entrenar a los modelos de lenguaje suponía un obstáculo para una adopción generalizada del ajuste (fine-tune) de los modelos. Los modelos de lenguaje tienden al sobreajuste cuando se trabaja con un dataset con dimensiones reducidas y sufriendo olvido catastrófico cuando se ajusta con un clasificador. Los modelos de procesamiento del lenguaje natural tienden a ser poco profundos y necesitan diferentes técnicas para el ajuste (Howard & Ruder, 2018).

Ante estos problemas, Jeremy Howard y Sebastian Ruder (2018) proponen un nuevo método Universal Language Model Fine-tuning (ULMFiT), que trata de permitir una transferencia de aprendizaje inductiva robusta para las tareas de procesamiento del lenguaje natural.

Figura 6 Fases de ULMFiT (Howard & Ruder, 2018)



En la primera fase de ULMFiT se pasa el modelo de lenguaje por un corpus general para que en las diferentes capas capturen las características generales del lenguaje. En una segunda fase, se ajusta el modelo con información específica de la tarea aplicando un ajuste con discriminación (como las capas van a capturar diferentes tipos de información, se aplican diferentes ratios de aprendizaje para las distintas capas) y STLR de modo que en esta fase se capturan las características específicas del caso. Por último, el clasificador se ajusta, en este caso se aplica deshielo gradual (se ajusta primero la última capa, luego se "deshiela" la siguiente capa y se ajustan ambas y así sucesivamente), también se aplica un ajuste con discriminación y STLR, el objetivo es preservar las representaciones en las capas más bajas y que se ajusten las capas más altas (Howard & Ruder, 2018).

GPT: haciendo uso de transformadores (introducidos por Google en 2017), se establece el procedimiento de entrenamiento en dos etapas, de manera similar a ULMFiT, en la

primera etapa se hace uso de un corpus de texto de gran tamaño, seguido por una capa de ajuste donde se ajusta el modelo a la tarea para la que se pretende usar.

Mientras que el entrenamiento de hace sin supervisión el ajuste se hace de manera supervisada para lograr adaptar los parámetros (Radford, Narasimhan, Salimans, & Sutskever, 2018).

2.2 ¿Por qué se usa BERT?

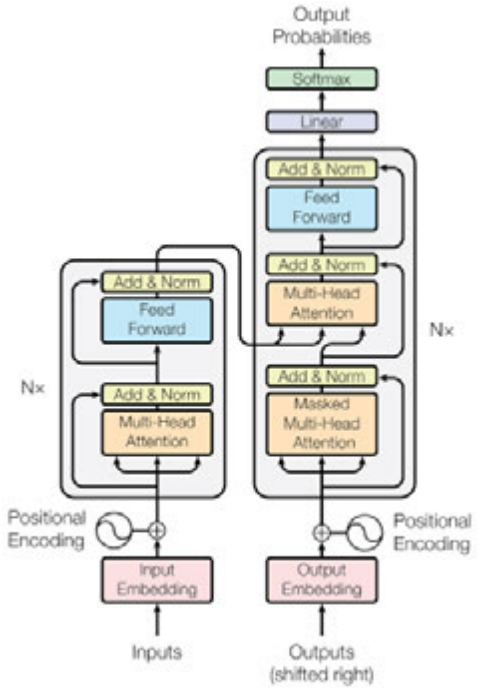
Si bien ya se ha dicho con anterioridad que el objetivo del trabajo es explorar las posibilidades que ofrece BERT para el análisis de sentimientos, en este caso para tuits en castellano, cabe explicar por qué se elige BERT para el trabajo y sus particularidades.

Como ya se ha mencionado con anterioridad BERT se publica en 2018, e introduce el entrenamiento bidireccional de transformadores. Para poder hablar de BERT es necesario hacerlo antes de los transformadores y del *paper* "*Attention is All you Need*" (Vaswani et al., 2017) publicado por Google en 2017, en el que se presenta la arquitectura de los transformadores.

Los flujos de trabajo de LSTM, con una introducción de la información de entrada en serie o secuencialmente, y son necesarios *inputs* del estado anterior para operar en el estado actual. Este flujo secuencial descrito no hace uso de las GPUs actuales diseñadas principalmente para computación en paralelo.

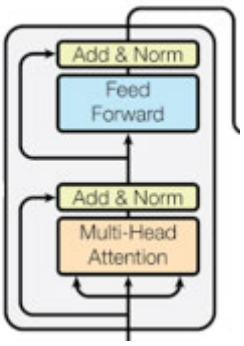
Para lograr la paralelización de la entrada de datos secuencial se introducen la arquitectura de los transformadores. La red de los transformadores tiene una arquitectura "*encoder-decoder*" similar a la de las redes neuronales recurrentes, la diferencia es que la secuencia de entrada se puede pasar en paralelo. Con un transformador no se pasa el *input* de manera simultánea y se obtienen los *embeddings* de manera simultánea.

Figura 7 Arquitectura del modelo de transformador (Vaswani et al., 2017)



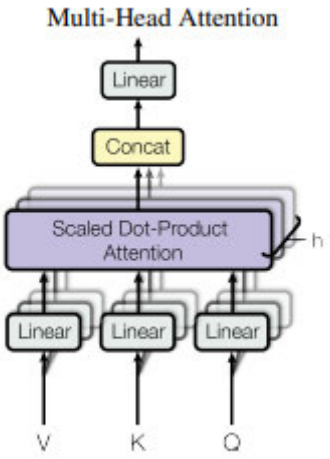
Para entender cómo se consigue esta paralelización es necesario estudiar la arquitectura de los transformadores explicada por CodeEmporium (CodeEmporium, 2020) y tomando como referencia el artículo original (Vaswani et al., 2017), para ello tomaremos como referencia que la entrada sería una oración. El "*input embedding"* nos da un vector que se corresponde con la palabra de entrada, una vez ha pasado por el "*input embedding"*. La siguiente fase es el "*positional encoder*", lo que hace el "*positional encoder*" es dar información sobre el contexto, la posición, de la palabra en la entrada, para ello se aplica una función al vector y se obtiene un nuevo *embedding* que ahora contiene información del contexto. Una vez se ha dado la información del contexto, el vector entra en el "*encoder block*", en este bloque pasa por una capa de "*multi-head attention*" y una capa de "*feed forward*".

Figura 8 Detalle del "encoder block" (Vaswani et al., 2017)



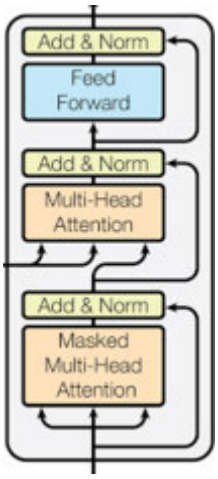
La atención responde a en qué parte de la entrada nos centramos, lo que hace la capa de atención es dar para cada palabra de la entrada un vector de atención propia en el que se captura la relación contextual entre las palabras de la oración. Como se tiene mayor interés en la relación con otras palabras de la oración, se obtienen múltiples vectores de atención y se calcula una media ponderada, de ahí que sea "*multi-head attention*".

Figura 9 Diagrama de Multi-head attention (Vaswani et al., 2017)



La segunda parte importante en el "*encoder block*" es la capa de "*feed forward*" es una red neuronal de *feed forward* que se aplica a cada uno de los vectores de atención de manera individual, como cada una de las redes de atención es independiente de las demás se puede paralelizar el proceso de *feed forward*, además transforma los vectores de atención a una forma que puede ser procesada por el "*decoder block*".

Figura 10 Detalle del "decoder block" (Vaswani et al., 2017)



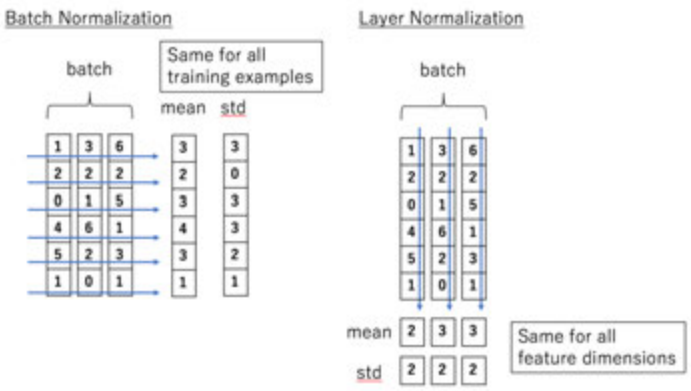
En el otro bloque, el "*decoder block*", durante la fase de entrenamiento tendrá como entrada el objetivo final. Para que el "*decoder* block" pueda entender este input, se debe

pasar antes de entrar al bloque por el "*input embedding*" y luego se le añade un vector de posición para tener información contextual del objetivo, ese vector ya puede entrar al "*decoder block*". El primer bloque dentro del *decoder* es una capa de *attention*, que genera los vectores de atención propios para cada palabra de la entrada (asumiendo que el objetivo final es otra oración). En este caso a diferencia del *encoder* este bloque tiene *"masked*" en el nombre, esto se debe a que no se utilizan todas las palabras de la oración objetivo, algunas de estas palabras se ocultan con una "máscara" de forma que la capa de atención no las puede utilizar.

Los vectores de atención propia junto con los vectores del *encoder* pasan al siguiente bloque de atención, que se conoce como "*encoder-decoder*" en el artículo original, este bloque de atención determina en qué grado está relacionado cada vector de atención con otro y aquí es donde se produce principalmente el mapeo. La salida del segundo bloque de atención es un vector de atención para cada palabra de cada una de las entradas (la del *encoder* y el input de la anterior capa de atención del decoder) cada vector representa la relación con otras palabras en ambas entradas. Posteriormente los vectores generados pasan a otro bloque de *feed forward*, donde se vuelve a dar un formato más manejable para futuras capas a los vectores.

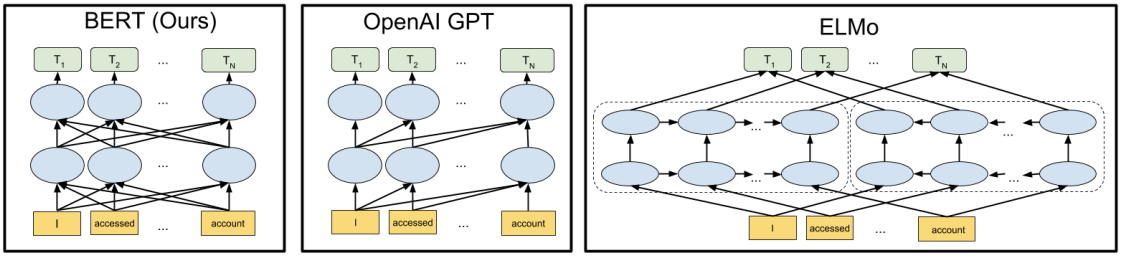
La capa lineal que se puede ver en la Figura 7, es otra capa de *feed forward*, se utiliza para adaptar las dimensiones al número de neuronas y, finalmente, Softmax lo transforma en una probabilidad, siendo la salida final es la palabra que se corresponde con la máxima probabilidad. Tras cada una de las capas se aplica normalización, normalmente se aplica normalización por lotes, esto permite una optimización más fácil utilizando ratios de aprendizaje más grandes, también es posible utilizar normalización por capas lo que permite normalizar por aparición y no por muestra.

Figura 11 Normalización por lotes y normalización por capas (Zhang & Zhang, 2019)



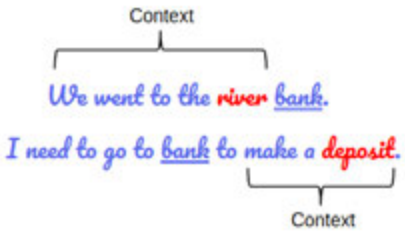
Es importante entender los transformadores puesto que BERT está basado en la arquitectura de los transformadores. Otra de las características de BERT es su entrenamiento previo, este entrenamiento se realiza sobre texto sin etiquetar que incluye la totalidad de la Wikipedia y Book Corpus, entre ambos suponen un corpus total de entorno a tres mil millones de palabras. Que BERT se entrene con un corpus de tal tamaño es parte del éxito que tiene BERT, con un entrenamiento de esas características el modelo va a capturar gran cantidad de detalles y se logra un entendimiento del funcionamiento del lenguaje (Devlin, Chang, Lee, & Toutanova, 2018).

Figura 12 Comparativa de bidireccionalidad entre BERT, GPT y ELMo. BERT es profundamente bidireccional, ELMo es bidireccional con poca profundidad y GPT es unidireccional (Devlin & Chang, 2018).



Otro aspecto importante de BERT es la bidireccionalidad, lo que quiere decirse con bidireccional es que BERT tiene en cuenta el entorno del token tanto a la derecha como a la izquierda durante la fase de entrenamiento, esto permite que se trabaje con más información a la hora de generar representaciones (Rizvi, 2019).

Figura 13 Contexto para la palabra "bank" donde la izquierda y la derecha cambian el significado (Rizvi, 2019).



El último aspecto importante por destacar de BERT es que permite con un pequeño ajuste añadiendo una última capa adicional de salida obtener buenos resultados, esto se da para una variedad de tareas de procesamiento del lenguaje natural.

Por lo explicado en el epígrafe anterior podría decirse que hay una evolución en la forma de tratar los problemas de procesamiento del lenguaje natural que finalmente acaba conduciendo a BERT que se resume a continuación, tomando como referencia la de Mohd Sanad Zaki Rizvi (2019), Jacob Devlin y Ming-Wei Chang (2018).

Tomando como punto de partida Word2Vec y GloVe, donde los *word embeddings* utilizados permitían capturar la relación entre diferentes palabras, sin embargo, los modelos utilizados eran poco profundos y esto limitaba su entrenamiento al limitar la información que podía ser capturada por el modelo, además como se explicó en el apartado anterior, estos modelos no tienen en cuenta el contexto de la palabra, solamente tienen en cuenta la palabra de la que se obtiene el *embedding*.

La siguiente innovación fue ELMo, con ELMo se pasa a trabajar con capas de arquitecturas LSTM bidireccional, con este cambio una palabra podía tener *embeddings* distintos como consecuencia de su contexto. A raíz de este cambio empieza a cobrar importancia el concepto de *pre-training*.

La mayor importancia del *pre-training* se hace evidente con la llegada de ULMFiT, el *framework* permitía entrenar modelos y hacer ajustes de modelos para obtener grandes resultados, con ULMFiT se establece la forma de transferencia del aprendizaje para procesamiento del lenguaje natural, y esto se consigue al hacer pre-training con un gran corpus y pasar después por un proceso de ajuste o *fine-tuning*.

GPT trabajó sobre los métodos introducidos por ELMo y ULMFiT en materia de *pre-training* y ajuste. La innovación de GPT fue la sustitución de LSTM por la utilización de transformadores, permitiendo que el modelo de GPT se pudiese entrenar para una variedad de tareas de procesamiento del lenguaje natural.

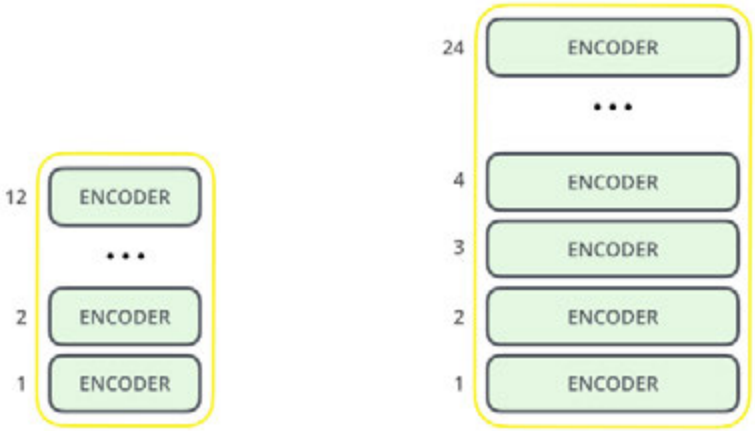
Finalmente, BERT toma como base los transformadores cuyo funcionamiento había quedado probado con el éxito del GPT para continuar avanzando en el procesamiento del lenguaje natural.

Sobre el trabajo previo y con las particularidades mencionadas sobre BERT, queda describir su arquitectura y funcionamiento para poder, finalmente, hablar de los resultados que se han obtenido con BERT.

Por lo que respecta a la arquitectura y como se ha dicho ya con anterioridad, esta se basa en la arquitectura de los transformadores, sin embargo, BERT únicamente utiliza varias capas de *encoders*, principalmente se destacan dos arquitecturas BERT base y BERT large, BERT base tiene doce *encoders*, "*hidden size*" de setecientos sesenta y ocho y doce "*self attention heads*" con ciento diez millones de parámetros, mientras que BERT large tiene veinticuatro encoders, "*hidden size*" de mil veinticuatro y dieciséis "*self attention heads*" con trescientos cuarenta millones de parámetros (Devlin, Chang, Lee, &

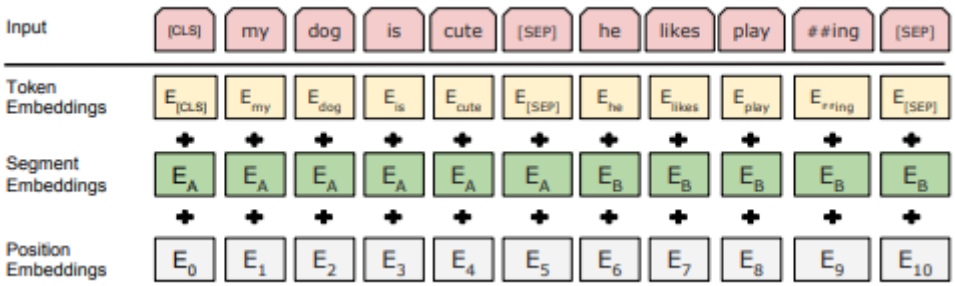
Toutanova, 2018). El modelo de BERT base tiene el mismo tamaño que GPT de OPENAI para facilitar las comparaciones, aunque BERT utiliza *self-attention* bidireccional mientras que GPT tiene "*constrained self*-attention" y solo tiene en cuenta el contexto a la izquierda.

Figura 14 Arquitectura de BERT ilustrada (Alammar, 2018)



Por lo que respecta al funcionamiento de BERT, con la finalidad de que BERT pueda dar soporte a una variedad de tareas de procesamiento del lenguaje natural, la representación de las entradas permite que se represente una única oración o una pareja de oraciones en una secuencia de tokens. Para la representación de la entrada BERT hace uso de WordPiece *embeddings* que tiene un vocabulario de 30000 tokens. El primer token de cada secuencia debe ser un token especial de clasificación ([CLS]). Para las parejas de oraciones, se compactan en un paquete para convertirlas en una única secuencia y son separadas con el token especial ([SEP]), posteriormente se averiguan los *embeddings*. Para cada token su representación de entrada se consigue sumando los correspondientes *embeddings* de token, segmento y de posición (Devlin, Chang, Lee, & Toutanova, 2018).

Figura 15 Representación del input de BERT (Devlin, Chang, Lee, & Toutanova, 2018)

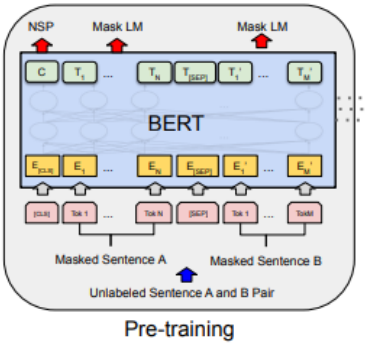


En el proceso de *pre-training* de BERT se utilizan dos tareas no supervisadas distintas: "*Masked LM*" y "*Next Sentence Prediction (NSP)*".

Masked LM: como consecuencia del entrenamiento, la palabra pueda "verse a sí misma" o que el modelo pueda predecir de manera trivial la palabra objetivo en un contexto multicapa. Para dar solución a este problema, un porcentaje del *input* es enmascarado de manera aleatoria (en el artículo original un 15% de cada secuencia), posteriormente se deben predecir esos tokens enmascarados. Como consecuencia se obtiene un modelo bidireccional pre entrenado, sin embargo, como consecuencia se genera una asimetría entre el proceso de *pre-training* y el de *fine-tuning* puesto que el token [MASK] que oculta esos tokens de entrada, no va a aparecer en el proceso de ajuste. Para mitigar este efecto, en el proceso de *pre-training*, el generador de datos de entrenamiento oculta con un [MASK] los tokens en el ochenta por ciento de los casos en los que se trata de utilizar [MASK], en otro diez por ciento se utiliza un token aleatorio y en el diez por ciento restantes, no se modifica el i-ésimo token. Finalmente, el *input* token es utilizado para predecir el valor del token original a través de la pérdida de entropía cruzada (Devlin, Chang, Lee, & Toutanova, 2018).

Next Sentence Prediction (NSP): muchas de las tareas para las que se puede utilizar BERT como preguntas y respuestas (QA por sus siglas en inglés) o inferencia del lenguaje natural (NLI por sus siglas en inglés) se basan en entender la relación entre dos oraciones. En el proceso de *pre-training* se entrena también una tarea de predicción de la siguiente oración. En el cincuenta por ciento de los casos, cuando se eligen pares de oraciones, la segunda oración B (separada en la secuencia por el token [SEP]) es realmente la oración siguiente y se indica con la etiqueta "*IsNext*" mientras que en el resto de los casos se indica con un "*NotNext*" que no es la siguiente oración en realidad (Devlin, Chang, Lee, & Toutanova, 2018).

Figura 16 Diagrama del pre-training de BERT (Devlin, Chang, Lee, & Toutanova, 2018).



El otro proceso importante en BERT es el de *fine-tuning* o ajuste, este es relativamente simple según Devlin et al. (2018), y esto se debe a que el mecanismo de *self-attention* de los transformadores permite a BERT adaptarse a una gran cantidad de tareas. Para cada tarea distinta sólo hay que dar a BERT los *inputs* y los *outputs* propios de la tarea y ajustar los parámetros *end-to-end*.

Algunos resultados de BERT comparado con otras opciones se muestran a continuación, entre las pruebas se encuentra GLUE (*General Language Understanding Evaluation*) que es una colección de diversas tareas de comprensión del lenguaje natural. SQuAD v1.1 que se trata del *dataset* para preguntas y respuestas de Stanford, dada una pregunta y un párrafo de la Wikipedia donde se encuentra la respuesta, la tarea consiste en encontrar la parte del párrafo en la que se está la respuesta. SQuAD v2.0 que amplía sobre el anteriormente mencionado SQuAD v1.1 donde se amplía la prueba con la posibilidad de que no exista una respuesta corta a la pregunta en el párrafo que se proporciona. Finalmente, SWAG (*Situations With Adversarial Generations*) un *dataset* con ciento trece mil pares de oraciones que evalúan el "sentido común", dada una oración se debe dar con la continuación más probable de cuatro posibles.

Figura 17 Resultados para el benchmark GLUE (Devlin, Chang, Lee, & Toutanova, 2018).

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$BERT_{BASE}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$BERT_{LARGE}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Para todos estos *benchmarcks*, BERT consiguió superar a las alternativas que se han mencionado con anterioridad, logrando excelentes resultados.

Figura 18 Resultados en SQuAD v1.1 (Devlin, Chang, Lee, & Toutanova, 2018).

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Figura 19 Resultados en SQuAD v2.0 (Devlin, Chang, Lee, & Toutanova, 2018).

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	86.3	89.0	86.9	89.5
#1 Single - MIR-MRC (F-Net)	-	-	74.8	78.0
#2 Single - nlnet	-	-	74.2	77.1
Published				
unet (Ensemble)	-	-	71.4	74.9
SLQA+ (Single)	-		71.4	74.4
Ours				
$\text{BERT}_{\text{LARGE}}$ (Single)	78.7	81.9	80.0	83.1

Figura 20 Resultados obtenidos en SWAG (Devlin, Chang, Lee, & Toutanova, 2018).

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
OpenAI GPT	-	78.0
$\text{BERT}_{\text{BASE}}$	81.6	-
$\text{BERT}_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

A la vista de estos resultados, resulta razonable intentar hacer uso de BERT y sus capacidades para la tarea propuesta de análisis de sentimientos, a pesar de que esta presenta una serie de complejidades que se explicarán con más detalle al analizar el dataset.

2.3 Tecnologías y Frameworks con los que se ha trabajado

Para la realización del trabajo se han utilizado: Google Colab, Google Cloud Platform, Google Drive, Python, Anaconda para Windows, bert-as-service, scikit-learn, PuTTy, FileZilla, un servidor con una GPU y Linux instalado. También se han utilizado PyTorch y HuggingFace a la hora de evaluar cómo utilizar BERT y cuál era la solución óptima para la realización del trabajo.

Google Colab: es un servicio en la nube gratuito que pone a disposición de los usuarios GPUs de las que se puede hacer uso sin realizar un pago extra. Colab permite programar en una estructura de cuaderno en Python. Pone a disposición del usuario librerías como Keras o TensorFlow, además permite ejecutar comandos *Shell*.

Ha sido utilizado de dos maneras distintas, inicialmente se trató de realizar en Google Colab una implementación de BERT que debido a las limitaciones que nos podemos encontrar con la herramienta y a las características de la tarea, un análisis de sentimientos con cuatro etiquetas distintas, no se pudo llevar a cabo. Sin embargo, sí que se pudo utilizar Google Colab para ajustar un modelo de BERT con un corpus en español para tratar de ajustarlo a la tarea.

Google Cloud Platform: aquí Google ha agrupado productos y herramientas que hacen uso de los recursos físicos que se encuentran en los centros de datos de Google. Dentro de estos productos se tiene acceso a máquinas virtuales, GPUs y TPUs y otra gran cantidad de recursos.

En el trabajo se ha utilizado para el almacenamiento del modelo de BERT que ha sido ajustado, esto se puede hacer de manera gratuita al dar Google un préstamo de trescientos dólares que se pueden usar en sus servicios (aunque con restricciones).

Google Drive: se trata del servicio de almacenamiento en la nube de Google. En este caso se ha utilizado para la transferencia del modelo de BERT que se entrenó, de Google Drive el modelo pasó al almacenamiento local antes de poder ser subido al servidor para hacer uso del modelo.

Python: lenguaje de programación de alto nivel, interpretado y orientado a objetos con una semántica dinámica (What is Python? Executive Summary, nd.). Las posibilidades que ofrece para distintos niveles de usuarios hacen que pueda resultar accesible, pudiendo ejecutarlo tanto en línea de comandos como en entornos de desarrollo o incluso en Google Colab (Downey, nd.).

Se ha utilizado Python en diferentes entornos, en primer lugar, se ha utilizado en Google Colab cuando se han explorado las posibilidades para hacer funcionar BERT. Posteriormente, se utiliza en Spyder para llevar a cabo las pruebas que guardaban mayor relación con el análisis de sentimientos.

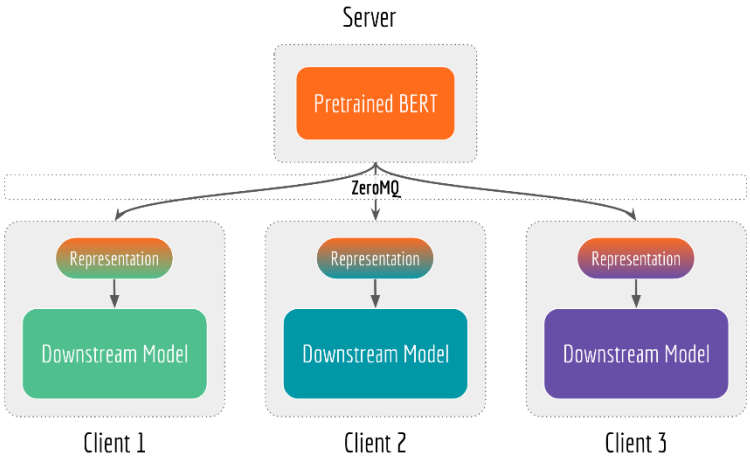
Algunas de las librerías utilizadas incluyen: bert-as-service, para la que se realizará una descripción más detallada más adelante, pandas para poder gestionar *dataframes* en la ingesta de los datos del *dataset*, sklearn la librería de *machine learning* elegida en este caso que también recibirá un análisis con más detalle, numpy, nltk y BeautifulSoup.

Anaconda: la *suite* de código abierto pone a disposición del usuario aplicaciones y librerías que hacen posible el trabajo fácil y rápido con Python, además, dispone de un gran número de librerías con las que es posible de trabajar. A su vez permite la creación de entornos con los que poder trabajar en la máquina local sin afectar a todo el equipo.

Se ha utilizado como ya se ha mencionado anteriormente entre otras cosas para programar, utilizando el IDE Spyder que se instala de manera simultánea al instalar Anaconda, además se ha utilizado para la descarga de librerías y la creación de entornos virtuales donde haciendo uso de pip se ha podido instalar el cliente de bert-as-service, por ejemplo.

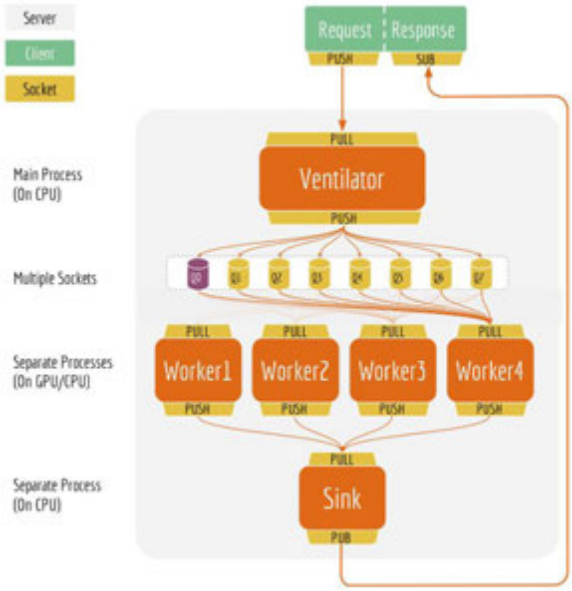
Bert-as-service: publicado por Han Xiao del exingeniero del Tencent AI Lab (en noviembre de 2018), convierte a BERT en un servicio de codificación de oraciones. El objetivo es dar el vector de representación de la entrada que se pueda obtener de una manera simple.

Figura 21 Diagrama de bert-as-service (Xiao, 2019)



Para lograr que funcione bert-as-service fue necesario tener disponible un servidor con una tarjeta gráfica, en este caso el servidor era de la Universidad Carlos III de Madrid. Para lograr la escalabilidad del sistema bert-as-service trabaja con un *pipeline* de *ventilator-worker-sink* con *sockets push/pull* y *publish/subscribe*. El *ventilator* actúa como un mecanismo de gestión de los lotes y mantiene el equilibrio de la carga.

Figura 22 Arquitectura de bert-as-service (Xiao, 2019).



Scikit-learn (sklearn): librería de Python, pone a disposición de quien la usa gran cantidad de algoritmos de clasificación, de regresión… además es compatible con otras librerías como NumPy (que ha sido necesario en este caso) (Scikit-learn, herramienta básica para el data science en Python. 2018).

En este caso se ha utilizado sklearn tanto para dividir el *dataset* en datos para test y de entrenamiento y por sus algoritmos de clasificación que han sido utilizados después de utilizar BERT para el *encoding*.

PuTTy: cliente que permite establecer conexiones mediante distintos protocolos.

Ha sido utilizado para poder conectarse con el servidor de manera remota y poder hacer uso de bert-as-service.

FileZilla: cliente FTP, ha permitido subir modelos de BERT al servidor, tanto modelos pre entrenados de Google como el modelo con el que se ha realizado la prueba da ajuste sobre un modelo de BERT.

Por último, mencionar el servidor al que la universidad ha permitido el acceso, para poder operar con bert-as-service, para ello es necesario tener a tu disposición una GPU con la que se lance el servicio. Las otras opciones para tratar de lanzar el servicio no resultaron viables como, por ejemplo, Google Colab, donde no era posible lograr comunicarse con el servicio una vez se había lanzado.

3. EXPERIMENTACIÓN

3.1 Descripción del *dataset*

Uno de los objetivos es conocer las capacidades de BERT a la hora de realizar análisis de sentimientos sobre tuits en español, esto supone que se necesita tener un *dataset* compuesto por tuits en español con su correspondiente etiqueta para su sentimiento. El *dataset* disponible se puede encontrar en:

http://www.sepln.org/workshops/tass/2017/#datasets

El *dataset* utilizado se hace para el de la primera tarea propuesta, análisis de sentimientos a nivel de tuit, esto supone que tenemos que ceñirnos a un *dataset* limitado puesto que no tenemos más tuits en español en *datasets* de análisis de sentimientos disponibles a la hora de realizar el trabajo.

El *dataset* se encuentra en formato XML y la información sigue la siguiente estructura una vez lo tenemos en un *dataframe*:

Figura 23 Salida de código que muestra el contenido del dataset

	tweetid	user	content	date	lang	sentiment
0	768213876278165504	OnceBukowski	-Me caes muy bien \r\n-Tienes que jugar más pa...	2016-08-23 22:30:35	es	NONE
1	768213567418036224	anahorxn	@myendlesshazza a. que puto mal escribo\r\n\r\...	2016-08-23 22:29:21	es	N
2	768212591105703936	martitarey13	@estherct209 jajajaja la tuya y la d mucha gen...	2016-08-23 22:25:29	es	N
3	768221670255493120	endlessmilerr	Quiero mogollón a @AlbaBenito99 pero sobretodo...	2016-08-23 23:01:33	es	P
4	768221021300264964	JunoWTFL	Vale he visto la tia bebiendose su regla y me ...	2016-08-23 22:58:58	es	N
5	768220253730009091	Alis_8496	@Yulian_Poe @guillermoterry1 Ah. mucho más por...	2016-08-23 22:55:55	es	P
6	768224728049999872	caval100	Se ha terminado #Rio2016 Lamentablemente no ar...	2016-08-23 23:13:42	es	N
7	768231706746912769	mgcsunshine	11. siiii fue super gracioso teniamos que habe...	2016-08-23 23:41:26	es	P
8	768231229439311872	__LOST_PRINCESS	@toNi_end seria mejor que dejasen de emitir es...	2016-08-23 23:39:32	es	N
9	768231166965145600	ConLDeLucifer_	@jonoro96 te mandaria a comprarte un burro, pe...	2016-08-23 23:39:17	es	N

En total el dataset cuenta con 1008 entradas, esto supone que es un dataset relativamente pequeño y esto podría afectar al análisis realizado al disponer de una muestra tan limitada.

A partir de esto se puede analizar el contenido de cada una de las partes del *dataset*:

Tweetid: identificador del tuit, no aporta información al análisis y no va a ser utilizado.

User: usuario que ha publicado el tuit, tampoco forma parte del análisis.

Content: el contenido del tuit es la primera parte del análisis que va a ser utilizada como entrada para bert-as-service y la primera parte sobre la que será necesario trabajar.

Date: indica la fecha en la que el tuit fue publicado, no se utiliza en el análisis.

Lang: indica el idioma en el que está el tuit, no se ha utilizado en el análisis.

Sentiment: almacena el sentimiento asociado al tuit, hay cuatro etiquetas de sentimiento posible: None que indica que no hay sentimiento, N indica sentimiento negativo en el tuit, P indica que el tuit tiene sentimiento positivo y Neu indica que el tuit es neutral.

Por lo tanto, corresponde explicar con algo más de profundidad el campo de sentimiento y especialmente el que contiene el texto del tuit.

El contenido de los tuits supone el elemento más complicado del *dataset*, para entender esta afirmación es necesario prestar atención a algunos ejemplos de los tuits del *dataset,* en este caso el primero.

"*-Me caes muy bien*

-Tienes que jugar más partidas al lol con Russel y conmigo

-Por qué tan Otako, deja de ser otako

-Haber si me muero"

En este ejemplo podemos apreciar varios elementos que demuestran la complejidad del análisis de este tipo de contenido. Se puede mencionar que será necesario limpiar el texto de guiones, tabulaciones y caracteres que no son letras, espacios o signos de puntuación. El primer elemento que se puede destacar es "lol" mientras que tradicionalmente en inglés sirve como abreviatura de "*laughing out loud*" en este caso un análisis basado en este concepto llevaría a error, en este caso el "lol" es el videojuego "League of Legends" al que también se le suele hacer referencia como "LoL". A continuación, se encuentra el uso de la expresión "otako", la expresión correcta sería "otaku", y aun utilizando la palabra correcta esta no se encuentra en el diccionario de la RAE y no tiene un uso común o extendido, por lo tanto, es difícil que el modelo pueda analizar correctamente su presencia.

Para terminar con el ejemplo anterior se debe analizar la última oración "Haber si me muero", en esta nos encontramos con una utilización incorrecta del verbo "haber" que debería ser "a ver", la utilización de haber hace que el modelo pre entrenado pueda encontrar grandes dificultades para poder entender esta oración. Por el conjunto de estas dificultades, el análisis de sentimientos para este tuit puede presentar una extrema complejidad.

Si procedemos al segundo tuit:

"*@myendlesshazza a. que puto mal escribo*

b. me sigo surrando help

3. ha quedado raro el "cómetelo" ahí JAJAJAJA"

Nos volvemos a encontrar con una serie de elementos que pueden hacer difícil la interpretación del contenido del tuit, en primer lugar, el usuario al que va dirigido y en segundo lugar "surrando".

Si tomamos los tuits cuarto y quinto de la muestra:

"*Quiero mogollón a @AlbaBenito99 pero sobretodo por lo rápido que contesta a los wasaps*"

"*Vale he visto la tia bebiendose su regla y me hs dado muchs grima*"

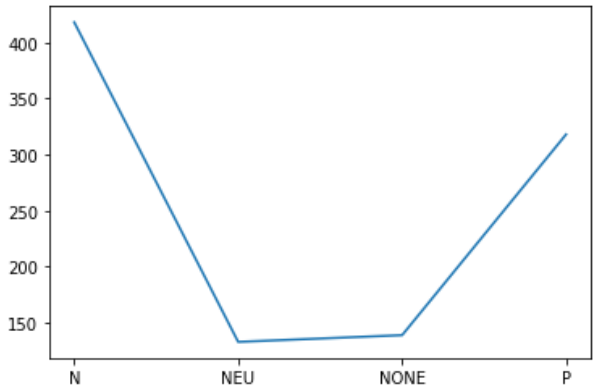
Podemos volver a apreciar problemas mencionados con anterioridad, en el caso del cuarto "wasaps" y en el quinto las faltas a la hora de escribir.

Con estos ejemplos es evidente que el propio contenido de lo que se va a analizar y la naturaleza de la muestra elegida para la generación del *dataset* van a hacer que el análisis sea más complicado.

Por lo que se entiende generalmente, el límite que tenía establecido Twitter de número de caracteres era de ciento cuarenta, cuando se ha analizado el tamaño de los *strings* en contenido estos llegan a ciento cuarenta y cinco, esto probablemente se debe a caracteres especiales que aparecen representados en el *string,* pero invisibles para el límite de Twitter. El *string* más corto tiene 18 caracteres, en ningún caso el largo de estos tuits debería suponer un problema a la hora de realizar el análisis.

Por lo que respecta a los sentimientos y sus frecuencias podemos observarlas en el siguiente gráfico:

Figura 24 Gráfico de frecuencias de los sentimientos. (elaboración propia)



Como se puede observar tenemos una diferencia muy grande entre las frecuencias de positivo y negativo y las de neutral y *none*. Esta diferencia que podemos apreciar entre las frecuencias puede afectar al análisis realizado sobre el *dataset*, el algoritmo de clasificación que se pueda aplicar va a tener defectos como consecuencia de la muestra de datos con una tendencia a considerar los tuits positivos y negativos.

Un segundo defecto que nos encontramos en lo referido a los sentimientos que tenemos definidos en el dataset es la presencia de *none* y de neutral, resulta complicado diferenciar ambas etiquetas. El análisis de sentimientos se encuentra de por sí se tiende a ignorar la neutralidad, en este caso se introducen dos etiquetas que podrían identificarse con el concepto de neutralidad y se obliga a no sólo a diferenciar bien el concepto de neutral con una muestra reducida, sino que se introduce otra etiqueta de características similares y con una muestra similar que dificulta el análisis.

Otro problema que se puede encontrar en el dataset como consecuencia del origen de la información, en este caso Twitter, es la forma de expresarse propia de las personas en esa red social. Una ocurrencia común en Twitter es la presencia de sarcasmo e ironía. En el caso de los humanos que tienen que leer estos textos, la presencia de sarcasmo puede hacer difícil la comprensión del texto, si tenemos esto en cuenta, el análisis de sentimientos en textos con presencia de ironía o sarcasmo con relativa frecuencia como es el caso de Twitter.

Finalmente, un elemento para tener en cuenta en el texto que puede tener un tuit es los emojis. Existen dos opciones: eliminarlos a la hora de tratar el texto, de forma que se puede perder contenido o modificar el significado al desaparecer la aportación. Sin embargo, también debe tenerse en cuenta que pueden profundizar en el sarcasmo e inducir a confesión. Por otro lado, es posible sustituir el emoji por un texto que represente la imagen, en este caso, no se pierde información, sin embargo, podemos caer en lo mencionado con anterioridad, agravar el efecto del sarcasmo en un tuit dificultando aún más el análisis.

3.2 Procesado de los datos

Una parte fundamental para cualquier tarea de procesamiento del lenguaje natural es el procesamiento de los datos, si no se realiza un buen trabajo procesando los datos del *dataset* el impacto en los resultados podría ser negativo.

Para poder conseguir la información se convierte el XML que se obtiene de la fuente a csv, este se carga en un *dataframe* de pandas, una vez se tiene en este formato se pueden empezar a realizar modificaciones al contenido de las columnas de este y preparar el texto.

El procesado de datos realizado sigue el siguiente orden:

1º Se ha utilizado la función *get_text*() de la librería *BeautifulSoup*, esta librería tiene funciones que permiten extraer texto de páginas web. Al aplicar la función *get_text*() nos queda únicamente texto dentro de contenido. Con *BeautifulSoup* también habría sido posible pasar todo el texto a minúsculas, pero como consecuencia del modelo de BERT que se utiliza para la tarea no es necesario puesto que el modelo se ha entrenado con texto capitalizado.

2º Eliminación de los emojis. Se aplica una función que sustituye los emojis presentes en el texto por un texto que explica lo que son, esto como se ha explicado con anterioridad puede suponer que se agraven los efectos del sarcasmo en algunos casos, pero en la mayoría de las ocasiones el emoji aportará más información de la que se tendría en caso de eliminarlo.

3º Se eliminan los signos de puntuación del texto, para ello se escribe una función que elimina todos los signos de puntuación puesto que para el análisis que se va a realizar no aportan información y podría considerarse que aportan ruido cuando sean procesados por BERT. Este proceso elimina además los signos de puntuación que se puedan haber introducido con la función de eliminación de emojis.

4º Tokenización. Para este proceso se utiliza la librería nltk y su función *sent_tokenize*(), esta nos permite indicar que el idioma utilizado es el español. Con este proceso lo que se hace es convertir la secuencia en palabras individuales que más serán el *input* de pasos siguientes.

5º Eliminación de palabras vacías, una vez se ha dividido el texto en palabras individuales se puede proceder a la eliminación de palabras vacías del texto puesto que estas palabras no van a aportar significado al análisis y por lo tanto solamente introducen ruido cuando

pasen a BERT. Para la eliminación de palabras vacías se vuelve a utilizar la librería nltk, en este caso se descarga el corpus de palabras vacías que tiene esta librería para el español, una vez se tiene el corpus se comparan las palabras contra el mismo y se elimina aquellas palabras que se encuentren dentro del corpus.

6º Eliminar saltos de línea, retornos de carro, tabulaciones… y adaptar el formato para sklearn y bert-as-service. Finalmente, una vez se han terminado todos los procesos anteriores nos encontramos con que es necesario eliminar "\n", "\r" y otros similares del texto puesto que no van a aportar información cuando se procese. Posteriormente también es necesario convertir el contenido de las columnas del *dataframe* a listas de forma que puedan ser procesados con facilidad por las funciones de sklearn y por bert-as-service.

Adicionalmente se tiene que realizar una modificación a las etiquetas para sentimientos que vienen por defecto, para que bert-as-service pueda trabajar con ellas sin problema se deben convertir los valores originales que eran *strings* a *integers*.

Otras opciones que se probaron para comprobar los resultados que ofrecían ante las características del texto con el que se trabajaba son la lematización y eliminar la primera palabra de cada tuit, normalmente se trata de un nombre de usuario, pero en ambos casos los resultados obtenidos eran peores y se descartaron.

3.3 Algoritmos utilizados

Si bien este apartado tiene como objetivo principal la descripción de los diferentes algoritmos de clasificación que han sido empleados una vez se ha codificado el texto haciendo uso de bert-as-service, se hará también mención de cómo BERT ha sido utilizado a la hora de intentar obtener esas codificaciones, aunque estás se explicarán con más detalle en el apartado siguiente que explicará las pruebas realizadas.

Entrando en detalle sobre la utilización de BERT y cómo se ejecuta bert-as-service, una vez se tiene el modelo de BERT con el que se desea trabajar en el servidor, es necesario instalar la librería de bert-serving-client, cuando se tiene instalada la librería se puede lanzar el servicio con el siguiente comando:

Figura 25 Comando para lanzar el cliente de bert-as-service

```
bert-serving-start -model ~/multi_cased_L-12_H-768_A-12 -num_worker=2 -max_seq_len=50
```

En la figura se puede observar cómo se establecen 3 parámetros:

Model: indica la ubicación del modelo en el servidor, no tiene ningún impacto en el rendimiento más allá del que pueda tener el propio modelo que se tiene almacenado, como en este caso el objetivo era trabajar en español, el modelo es el multilingüe pre entrenado del que se ha hablado antes.

Num_worker: establece el número de *workers* que va a trabajar en la GPU/CPU, cada uno de ellos trabajará en un proceso distinto. Este parámetro puede afectar a la velocidad con la que se obtiene una respuesta del servidor, sin embargo, no tiene un impacto sobre el resultado del análisis, solamente afecta a la velocidad de ejecución. Lo recomendado por Han Xiao, creador de bert-as-service es un número igual o menor al número de GPUs/CPUs disponibles, por tanto, en este caso el número de trabajadores recomendado es de 1 (aunque es posible trabajar con 2).

Max_seq_len: es un parámetro que afecta al servidor, controla el tamaño de secuencia máximo que admite el modelo de BERT. Si se envía una secuencia que excede este máximo entonces se truncará. Este parámetro afecta tanto al rendimiento como al resultado final del análisis en caso de que se tengan que truncar las secuencias enviadas. Ha sido posible trabajar con un tamaño máximo de secuencia de 150, esto supone que en ningún caso se tienen que truncar las secuencias enviada, a cambio de esto el tiempo de respuesta del servidor es algo más lento al tardar más en procesar estas secuencias más largas.

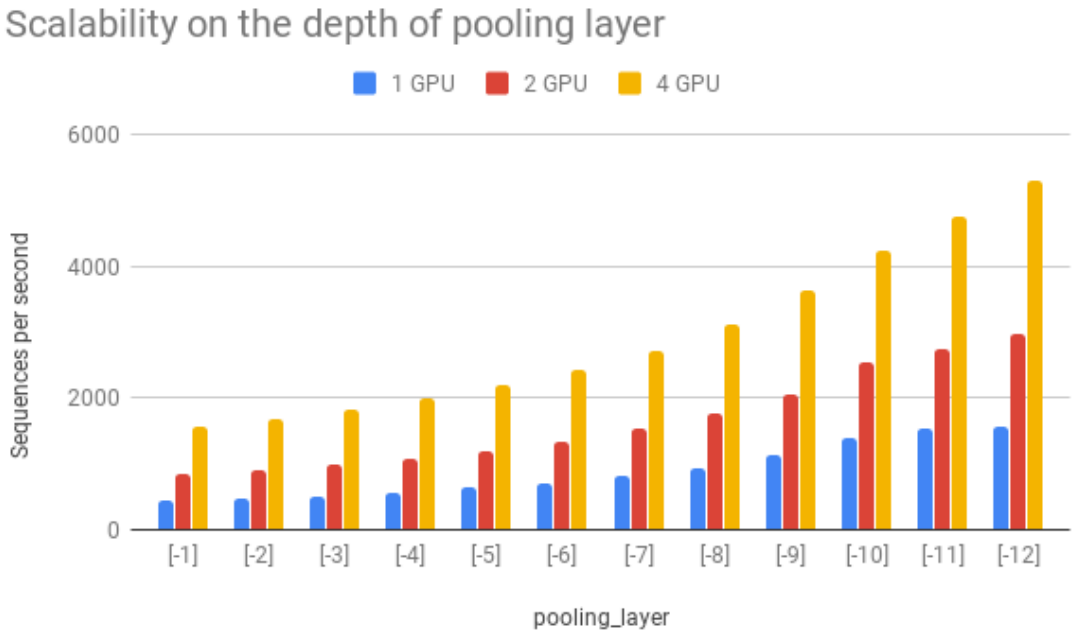
Otros parámetros que es posible modificar son:

Client_batch_size: este parámetro del lado del cliente afecta a la función *encode*, permite obtener un mejor rendimiento cuando se envían las secuencias por lotes en lugar de hacerlo todo a la vez.

Max_batch_size: parámetro del servidor que determina el número máximo de muestras por lote por trabajador. Si llega al servidor un lote con un tamaño mayor que el establecido el servidor deberá dividirlo en lotes más pequeños (o iguales) que el máximo establecido.

Pooling_layer: determina en qué capa de *encoder* se va a operar, un -1 supone trabajar en la capa más cercana al *output*, mientras que un -12 (en BERT base) supone trabajar lo más cerca posible de la capa de *embedding*. Este parámetro afecta a la velocidad, a menor número de *encoders* mayor velocidad.

Figura 26 Variación en el rendimiento en función del valor de pooling_layer (Xiao, 2019)



Una vez explicado BERT y cómo se ha hecho uso de los modelos de Google, a continuación, se hablará de los distintos algoritmos utilizados para la clasificación de la salida de BERT, algunos algoritmos han sido utilizados para analizar cómo se adaptaban al problema, pero ante los resultados no se profundizó en el análisis y no se recogerán en este apartado.

Los algoritmos utilizados proceden todos de la librería scikit-learn, dentro de esta librería existe una gran variedad de algoritmos que permiten hacer una clasificación multi etiqueta, entre ellos se encuentran algunos como *Naive Bayes*, *Random Forest*…

En este caso el objetivo principal era la realización de un análisis entre todas las etiquetas de manera simultánea, por lo que aquellos clasificadores que aplican uno contra el resto o que hacen clasificaciones uno contra uno, no han sido utilizados en este caso.

Naive Bayes: son algoritmos de clasificación basados en el teorema de Bayes, en los modelos "*Naive*" se asume la independencia entre las distintas variables usadas en la predicción, esto quiere decir que una determinada característica en un conjunto de datos no está relacionada con cualquier otra característica que pueda estar presente. Se trata del cálculo de una probabilidad *a posteriori* basada en unas probabilidades conocidas (anteriores) (Roman, 2019). La fórmula para el cálculo de una probabilidad en Naive Bayes es la siguiente:

Figura 27 Fórmula de probabilidad teorema de Bayes (Estadística de la probabilidad: Teorema de bayes)

$$P(A_i/B) = \frac{P(A_i) \cdot P(B/A_i)}{P(B)}$$

Donde:
$P(A_i)$ = Probabilidad a priori
$P(B/A_i)$ = Probabilidad condicional
$P(B)$ = Probabilidad Total
$P(A_i/B)$ = Probabilidad a posteriori

Dentro de Scikit-learn encontramos dos opciones para aplicar Naive Bayes, uno es un modelo Bernoulli y el otro es Gaussiano, la utilización de uno u otro depende del problema que se esté tratando. Bernoulli se utilizará si una característica representa, por ejemplo, la presencia de un término, mientras que el gaussiano se utilizará cuando se quiera representar la probabilidad de un término. A continuación, se explican ambas implementaciones de scikit-learn (scikit-learn).

BernoulliNB: implementa algoritmos de entrenamiento y clasificación para datos distribuidos como una distribución de Bernoulli multivariada. Esta clase requiere de una representación de los valores con forma de vectores booleanos de aparición, si se utiliza otro tipo de dato entonces puede que convierta en binarios los datos en función del parámetro binarize.

La fórmula que utiliza este Naive Bayes de scikit-learn para las decisiones es:

Figura 28 Fórmula para toma de decisión BernoulliNB de scikit-learn (scikit-learn)

$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i)$$

Esta fórmula penaliza de manera explícita la no ocurrencia de una característica que sea indicador de una clase $\boldsymbol{y}$. Bernoulli puede ser usado para análisis de sentimientos, su rendimiento puede ser mejor que el de otros Naive Bayes en determinados *datasets*, especialmente para documentos más cortos (scikit-learn).

Los parámetros que tiene son:

- Alpha: parámetro para suavizado de la función (en este caso Laplace). En el caso de estudio, un mayor suavizado hará que se tienda a clasificar en las 2 etiquetas más abundantes, en este caso positivo y negativo.
- Binarize: determina a partir de qué valor se "binariza", si se le da valor None, se asume que la muestra ya son vectores binarios. La selección de este valor

dependerá de las características de los datos, en este caso el valor por defecto es el que obtiene los mejores resultados.

- Fit_prior: determina si se deben aprender las probabilidades previas de las clases.
- Class_prior: es un array con las probabilidades previas de las clases.

GaussianNB: implementa un clasificador de Bayes ingenuo gaussiano, la probabilidad de las características se asume gaussiana. La fórmula para el cálculo de la probabilidad es la siguiente:

Figura 29 Fórmula para el cálculo de la probabilidad usado por GaussianNB (scikit-learn)

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Los parámetros σ_y y μ_y se estiman utilizando máxima verosimilitud.

Los parámetros de GaussianNB son los siguientes:

- Priors: probabilidades previas de las clases, en caso de que no se ajusten en base a los datos.
- Var_smoothing: parte de la varianza más grande de todas las características que se añade a las varianzas para mayor estabilidad en el cálculo.

Random Forest: para entender el *Random Forest* es necesario dar primero una pequeña explicación de los árboles de decisión. Los árboles de decisión es la base sobre la que se construye el *Random Forest* (Breiman, 2001), una forma de interpretar el árbol de decisión es como una serie de preguntas respecto a los datos que eventualmente llevarán a una predicción de una clase (como se trata de un clasificador para el problema que tratamos). Los detalles técnicos del árbol dependerán de cómo se generan esas preguntas que se realizan respecto a los datos (Koehrsen, 2018). En el caso del algoritmo CART, el árbol se construye determinando las preguntas que reducen al mínimo el índice Gini de impuridad. Como consecuencia de esa forma de generar nodos, cada nodo contendrá una gran cantidad de muestras de una sola clase encontrando características que dividen los datos en clases (Loh, 2011). El índice Gini de impuridad es la probabilidad de que una muestra aleatoria seleccionada de un nodo se etiquetase de manera incorrecta, finalmente cuando se llega a las capas más profundas el índice tiende a cero.

Un *Random Forest* es por tanto un modelo que comprende varios árboles de decisión, en lugar de calcular una media de las predicciones realizadas por los árboles, se aplican dos conceptos distintos:

Muestreo aleatorio de observaciones del entrenamiento, cuando se entrena un árbol en un *Random Forest*, este aprende de una serie de muestras aleatorias de datos, las muestras se seleccionan con *bootstrapping* lo que supone que una muestra puede ser utilizada en varias ocasiones para un mismo árbol. El objetivo es que en el conjunto de todos los árboles se minimice la varianza sin que sea a costa de la generación de un sesgo (Koehrsen, 2018).

Partes aleatorias de características para dividir nodos, sólo una parte aleatoria de las características es utilizada para dividir cada nodo de cada árbol de decisión.

Por lo que respecta a los *Random Forest* de scikit-learn, siguiendo la documentación de la librería (2007-2019), se construyen teniendo en cuenta ambos conceptos mencionados con anterioridad. Esta introducción de aleatoriedad hace que disminuya la varianza en el *Random Forest*, a diferencia de un único árbol que tiende a tener una alta varianza y a sobre ajustar. La introducción de la aleatoriedad hace que los errores de los árboles del "bosque" tengan errores de predicción poco relacionados, esto puede hacer que incluso algunos errores se cancelen entre sí. Esta aleatoriedad reduce la varianza a cambio de un pequeño incremento en el sesgo. Esto concuerda con lo explicado hasta ahora sobre los *Random Forest*. En lo que difiere respecto a la publicación original de Breiman (2001) en la que los árboles votan la clase más popular, en la implementación de scikit-learn, combina las predicciones calculando una media.

Los parámetros que se pueden establecer a la hora de llamar a *Random Forest* son los siguientes conforme a la documentación de scikit-learn (2011):

- N_estimators: el número de árboles en el "bosque". Esto tiene un impacto en el rendimiento (a mayor número de árboles más lenta la ejecución) y en los resultados, cuanto mayor sea el número de árboles mayor tendencia va a tener el clasificador a elegir positivo o negativo como consecuencia del *dataset* con el que se trabaja.
- Criterion: selecciona la función que se utiliza para medir la calidad de una división. Se puede usar el índice de impuridad de Gini mencionado antes o entropía, la entropía requiere de más recursos a nivel computacional. Gini tendrá

un impacto en las clasificaciones erróneas mientras que entropía se tiende a usar en análisis exploratorios.

En el caso de este estudio, la utilización de entropía arroja mejores resultados, sin embargo, conforme aumenta el número de árboles, Gini logra menores errores de clasificación para los casos con menor número de apariciones en la muestra.

- Max_depth: determina la profundidad máxima de los árboles, si no se establece no se pone límite a esta. Con árboles de menor profundidad, la tendencia a clasificar en base a positivo y negativo se ve incrementada al suponer estos la mayor parte de las muestras.
- Min_samples: el número mínimo de muestras necesarias para dividir un nodo interno. Si se establece un número alto, como consecuencia de las características del dataset, se dejará de clasificar como none y neutro puesto que su número de muestras es muy bajo comparado con positivo y negativo.
- Min_samples_leaf: número de muestras mínimo necesario para que un nodo sea una hoja. El comportamiento con el *dataset* es similar al descrito para min_samples.
- Min_weight_fraction_leaf: la fracción mínima de peso requerida sobre la suma total de pesos necesaria para ser un nodo hoja. De manera análoga a los dos parámetros anteriores, un incremento en el peso mínimo supone que se tiende a las dos etiquetas con mayor número de apariciones.
- Max_features: el número de características que se consideran a la hora de seleccionar la mejor división. Se puede seleccionar un número natural, un decimal, en dicho caso se considerará el entero resultante de multiplicar el valor de max_features por n_features para cada división; auto tiene el mismo efecto que sqrt, en ambos se trabaja con la raíz de n_features, log2 hará que max_features sea el logaritmo en base 2 de n_features y finalmente si se introduce None será igual a n_features. En este caso un mayor número de características consideradas hace que mejore la precisión para aquellas etiquetas con menor representación, sin embargo, esta ser verá reducida ligeramente para las que tienen mayor número de ocurrencias.

- Max_leaf_nodes: máximo de nodos hoja, los árboles se generan en base a el mejor primero. Introducir un límite, perjudicará a aquellas etiquetas con un menor número de ocurrencias.
- Min_impurity_decrease: la reducción de la impureza mínima para que se produzca una división, esto quiere decir que cuando una división provoque una reducción de impureza igual o mayor se produce la división. Valores altos perjudican de nuevo a aquellas etiquetas con menor número de apariciones.
- Bootstrap: determina si se utilizan muestras para la construcción de los árboles, sino se utiliza la totalidad del *dataset* se utiliza para construir cada árbol. Por lo explicado con anterioridad referido a varianza y sesgo se deja el valor por defecto que utiliza Bootstrap.
- Oob_score: utilización de muestras "*out of the bag*" para estimar la precisión de generalización. Debido a la limitación al número de árboles para evitar la tendencia a seleccionar las etiquetas que suponen la mayor parte de la muestra no se tienen suficientes árboles para hacer uso de este parámetro.
- N_jobs: número de tareas que se van a ejecutar en paralelo. Afecta al rendimiento, pero no a la clasificación.
- Random_state: controla la aleatoriedad del bootstrapping y del muestreo de las características cuando se busca la mejor división para cada nodo.
- Verbose: controla la verbosidad en el ajuste y predicción.
- Warm_start: permite utilizar la solución de la anterior llamada a fit, añadiendo más estimadores, si no se usa se genera un "bosque" completamente nuevo.
- Class_weight: permite dar un peso a cada una de las clases. Utilizando este parámetro se puede asignar a las etiquetas para neutral y none un peso que compense su falta de apariciones en comparación con las restantes, sin embargo, hacer esto puede suponer una pérdida de precisión tanto para positivo como para negativo por lo que los pesos deben seleccionarse cuidadosamente.
- Ccp_alpha: permita establecer un valor para la poda en función del mínimo coste-complejidad. El subárbol con el mayor coste-complejidad por debajo de ccp_alpha es podado.

- Max_samples: si se ha habilitado Bootstrap, establece el número de muestras que se seleccionan para entrenar cada estimador.

LogisticRegression: a pesar del nombre, se trata de un modelo de clasificación linear más que de una regresión (scikit-learn), también se conoce como regresión logit, clasificación por máxima entropía o clasificación log-linear. Las probabilidades describen las posibles de una prueba descritas usando una función logística (scikit-learn). Es importante tener en cuenta que sólo se puede tomar como multiclase cuando se utiliza multi_class = "multinomial" sino es un uno contra el resto.

Los parámetros que permite especificar scikit-learn son:

- Penalty: especifica la norma utilizada para la penalización, algunos *solvers* solo tienen soporte para l2, 'elasticnet' solo se puede usar para saga. Si se introduce none (que no es soportado por 'liblinear') no se aplica regularización.
- Dual: formulación dual o *primal*. Dual sólo se implementa con penalización l2 con un *solver* 'liblinear'.
- Tol: tolerancia para los criterios de parada.
- C: inversa de la fuerza de la regularización, debe ser positiva, los valores más pequeños indican una regularización más fuerte.
- Fit_intercept: indica si se debe añadir una constante a la función de decisión.
- Intercept_scaling: sólo funciona con el *solver* 'liblinear' con fit_intercept siendo True. Convierte al intercepto en el producto: intercept_scaling*synthetic_feature_weight.
- Class_weight: peso asociado a las clases, funciona de manera similar al de *Random Forest* explicado anteriormente, nos permite dar el mismo tratamiento a las etiquetas de forma que se trata de mitigar la diferencia apariciones en el *dataset* a costa de menor precisión en positivo y negativo.
- Random_state: usado para los *solvers* 'sag', 'saga' o 'liblinear' para mezclar los datos.
- Solver: indica el algoritmo que se debe utilizar para la optimización del problema.
    - Para *datasets* pequeños se recomienda la utilización de 'liblinear' mientras que para grandes se recomienda 'saga'

    - Para clasificación con múltiples clases no se recomienda 'liblinear' sino 'newton-cg', 'sag', 'saga' y 'lbfgs'.

- Max_iter: máximo número de iteraciones que el *solver* debe realizar para converger.
- Multi_class: como ya se ha indicado con anterioridad deberá tener valor multinomial para el caso que se está tratando.
- Verbose: verbosidad para *solvers* 'liblinear' y 'lbfgs'.
- Warm_start y n_jobs funcionan de manera similar a como lo hacían para *Random Forest*.
- L1-ratio: parámetro de mezcla para l1, solamente se usa si se establece penalización 'elasticnet'. Indica el grado en el que se entremezclan l1 y l2.

No se han especificado efectos individuales para gran parte de los parámetros al estar la mayoría de ellos interconectados, dependiendo del *solver* o *penalty* seleccionados y por tanto no pudiendo aislar los efectos. Sí que se destaca que no es posible usar 'liblinear' al tratarse de un problema con múltiples clases.

LabelPropagation: da nombre a varias formas de algoritmos de inferencia semi supervisados. Con scikit-learn se tiene acceso a dos modelos de propagación, *label propagation* y *label spreading*, ambos funcionan de manera similar construyendo un grafo sobre todos los elementos del *dataset* (scikit-learn).

Label propagation y *label spreading* difieren en las modificaciones que se realizan a la matriz de similitud del grafo y el efecto *clamping* en la distribución de etiquetas. El *clamping* permite al algoritmo cambiar el peso medido en entrenamiento y *testing* para las etiquetas en un cierto grado. Con *label propagation* se produce un *clamping* duro, lo que supone que se hace con un factor $\alpha = 0$, este se puede relajar a valores como $\alpha = 0.2$ lo que supone que se retiene un ochenta por ciento del valor original de la distribución de etiquetas, pero el algoritmo puede cambiar la confianza de la distribución en un veinte por ciento. *Label propagation* hace uso de una matriz de similitud construida a partir de los datos sin modificaciones, *label spreading* por otro lado, minimiza una función de pérdidas que tiene propiedades de regularización por lo que suele ser más robusta en caso de que haya ruido en la muestra (scikit-learn).

Los parámetros que se pueden modificar para *label propagation* son según la documentación:

- Kernel: se especifica o bien la función kernel o su identificador. Sólo admite 'knn' y 'rbf' como entradas. La función que se quiera usar que no sea esas dos anteriores deberá aceptar una entrada de forma (n_samples, n_features) y devolver una matriz de pesos con forma (n_samples, n_samples). El funcionamiento de 'rbf' depende de gamma y el de 'knn' de n_neighbors por lo que se hablará de los efectos de cada kernel al hablar de estos valores.
- Gamma: especifica el valor de gamma para la función de 'rbf'. Debe estar entre $\gamma = 0.2$ y $\gamma = 1$ para evitar que se clasifique únicamente en una etiqueta, cuanto más cercano a 0 mayor tendencia a las etiquetas con más apariciones. Los valores por encima de 0.8 empiezan a dar demasiado peso a neutral y por encima empieza a clasificar todo como neutral.
- N_neighbors: indica el número de *neighbors* que se va a utilizar para 'knn', de manera similar al número de árboles en el *random forest*, conforme incrementa este número mayor tendencia a la clasificación en positivo y negativo.
- Max_iter: número máximo de iteraciones permitido.
- Tol: tolerancia de convergencia, establece el valor para el cual se considera que un sistema está en un estado estable.
- N_jobs: de nuevo el número de tareas a ejecutar en paralelo.

Nearest Neighbor classification: la clasificación por vecinos más cercanos se basa en aprendizaje no generalizado. No se trata de construir un modelo interno, se almacenan instancias de datos, la clasificación se construye a partir del "voto de la mayoría" de los vecinos más cercanos (scikit-learn).

KNeighborClassification: es la técnica más utilizada, la elección del número de vecinos dependerá de los datos, un mayor número de vecinos elimina ruido, pero difumina las fronteras (scikit-learn).

Los parámetros que se pueden modificar son:

- N_neighbors: establece el número de vecinos, si se utiliza un número muy alto la clasificación tiende a clasificar en las clases que aparecen más en la muestra, esto se debe a que habrá más vecinos cerca de los datos más frecuentes en el *dataset*.

- Weights: función de peso usada en la predicción, permite usar un peso uniforme donde todos los puntos tiene el mismo peso, permite utilizar la distancia, donde a menor distancia mayor influencia. Por último, se puede usar una función de peso definida por el usuario. La utilización de un peso uniforme beneficia a las clases menos comunes mientras que la distancia beneficia a las clases más comunes.
- Algorithm: indica el algoritmo utilizado para computar los vecinos más cercanos. Puede ser utilizando balltree, un KDTree, fuerza bruta o automático donde se tratará de buscar el algoritmo más apropiado en base a los valores que se dan al método fit.
- Leaf_size: se utiliza en caso de usar alguno de los algoritmos basados en árboles.
- P: parámetro de potencia para la métrica de Minkowski, afecta al rendimiento y consumo de memoria.
- Metric: la métrica utilizada para la distancia, por defecto se utiliza Minkowski con p de dos, así es equivalente a una distancia euclídea.
- Metric_params: argumentos adicionales para la función para la métrica.

También se puede especificar el número de tareas utilizadas.

RadiusNeighborsClassifier: se utiliza para casos donde el muestreo de datos no es uniforme, se utiliza un radio fijo de forma que los puntos con menos vecinos utilizan menos menor número de vecinos en la clasificación. Este método puede tener problemas con espacios con múltiples dimensiones (scikit-learn).

Los parámetros que se pueden modificar son, según la documentación:

- Radius: el radio utilizado para determinar el número de vecinos empleados en la clasificación. Debido a las características del *dataset* cuando el radio consigue encontrar vecinos en todos los casos la clasificación es en su totalidad como negativo.
- Outlier_label: etiqueta las muestras que se consideren *outliers*, se puede hacer de manera manual, se puede asignar la etiqueta de más frecuente de y a los *outliers* o se puede no etiquetarlos (esta es la opción por defecto).

El resto de los parámetros son idénticos a los de KNeighborsClassifier.

Con esto ya se han explorado todos los algoritmos que han sido utilizados y se puede proceder a la experimentación realizada para alcanzar los objetivos.

3.4 Experimentación

Las primeras pruebas que se han realizado estaban centradas en la búsqueda de la implementación de BERT que mejor se adaptase a las necesidades del problema, una vez se tuviese un modelo de BERT funcionando se podría proceder a la experimentación relacionada con el análisis de sentimientos.

Inicialmente, por tanto, conviene hablar de las pruebas realizadas sobre a la utilización de BERT al ser este uno de los objetivos principales del trabajo.

En primer lugar, se intentó implementar BERT en un *notebook* de Google Colab, en este caso el código estaba basado en el de Chris McCormick y Nick Ryan (2019). En este caso el modelo que se descarga es BERT Base multilingual cased, este modelo está entrenado en varios idiomas y entre ellos se encuentra el español, este modelo se trata del más pequeño de los dos posibles de BERT que tiene un número menor de *encoders*.

Figura 30 Código que importaba el tokenizer del modelo multilingüe de BERT en la prueba realizada

```
from transformers import BertTokenizer

# Load the BERT tokenizer.
print('Loading BERT tokenizer...')
tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased', do_lower_case=False)
```

Cuando se trató de ajustar el número de etiquetas y el modelo de BERT utilizado, empezaron a aparecer dificultades relacionadas con CUDA y con Google Colab que hacían imposible el ajuste del modelo y por lo tanto hacían inviable esta opción. A pesar de ponerse en contacto con los creadores del *notebook* no se pudieron resolver estos problemas encontrados.

Figura 31 Configuración del modelo de Bert basado en la librería Transformers.

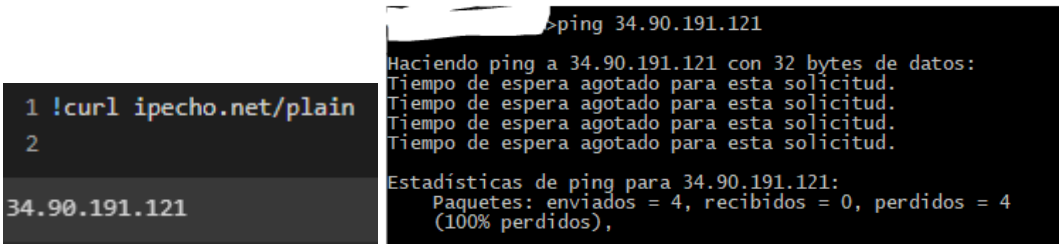
```
from transformers import BertForSequenceClassification, AdamW, BertConfig

# Load BertForSequenceClassification, the pretrained BERT model with a single
# linear classification layer on top.
model = BertForSequenceClassification.from_pretrained(
    "bert-base-multilingual-cased", # Use the 12-layer BERT model, with a cased vocab.
    num_labels = 4, # The number of output labels--2 for binary classification.
                    # You can increase this for multi-class tasks.
    output_attentions = False, # Whether the model returns attentions weights.
    output_hidden_states = False, # Whether the model returns all hidden-states.
)

# Tell pytorch to run this model on the GPU.
model.cuda()
```

Con este código se pretendía realizar un ajuste directamente sobre el modelo de BERT, sin embargo, por las limitaciones que tiene Google Colab, no es posible hacer uso de un modelo multietiqueta, sin embargo, con un modelo binario sí que es posible hacer uso de esta opción de entrenamiento que está abierto a más modelos de BERT y es por tanto digno de mención para problemas puramente binarios como una alternativa al uso de bert-as-service en el que se tiene más control sobre el modelo.

Como consecuencia de este error empezó la búsqueda de una alternativa encontrada en bert-as-service, al hacer uso de bert-as-service, siempre que se tuviese acceso a un modelo de BERT, el cual se puede descargar, sería posible lanzarlo con una arquitectura cliente-servidor y hacer peticiones desde una máquina local al servidor.

El primer intento de hacer uso de bert-as-service se hace en Google Colab, la posibilidad de utilizar comandos *Shell* permite cambiar las versiones de Python y TensorFlow para poder lanzar el servicio, una vez lanzado el servicio se intenta comunicar con Google Colab, pero no es posible comunicarse con el servicio iniciado en Colab

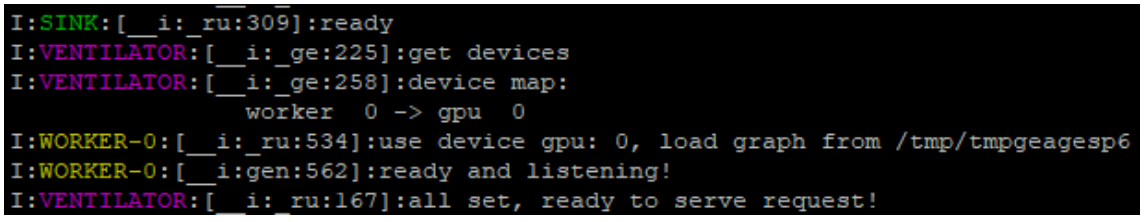
Figura 32 Obtención de dirección IP asociada al Notebook de Google Colab y fallo del intento de comunicación.



Ante esta situación se solicita poder hacer uso de un servidor de la universidad que también tiene una GPU, el acceso es concedido y se accede al mismo por medio de una VPN y ssh (PuTTy).

Una vez se tiene acceso al servidor, se procede a la instalación de bert-serving-client, la subida del modelo de BERT multilingüe y se trata de lanzar el servicio. Esta vez se tiene éxito a la hora de lanzar el servidor.

Figura 33 Salida cuando se lanza bert-serving-server con éxito

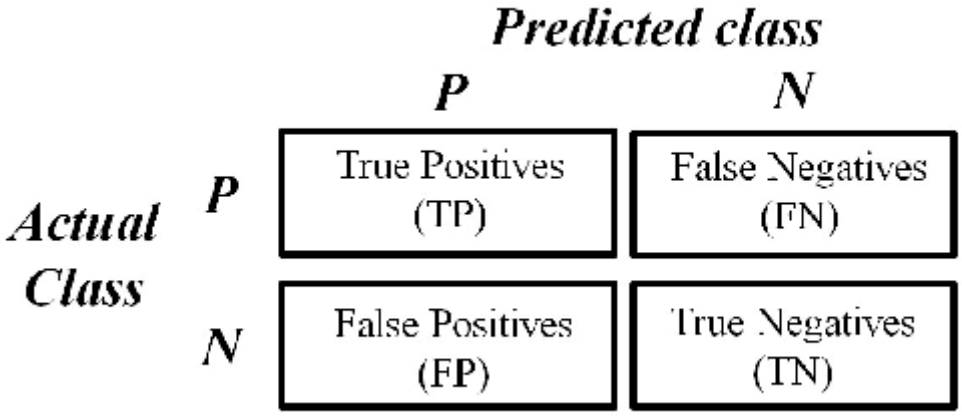


Antes de entrar en las pruebas realizadas es importante mencionar que las clases se representan con etiquetas numéricas como se ha mencionado al hablar del tratamiento de los datos, como referencia para el resto de la experimentación las etiquetas son: NEUTRAL = 1, NONE = 2, NEGATIVO = 3, POSITIVO = 4.

Las métricas utilizadas para el análisis son la matriz de confusión y las salidas de *metrics.classification_report* de sklearn, dentro de estas salidas se encuentra la *precision*, *recall*, *f1-score*, *support*, *accuracy*, *macro avg* y *weighted avg*.

La matriz de confusión permite analizar la precisión de la predicción, pudiendo observar los positivos reales, cuando para una etiqueta coinciden ambos ejes, es decir, para la etiqueta 2 la posición [2,2] de la matriz. Permite ver cómo se está confundiendo el clasificador, es decir, si para una etiqueta que signifique positivo tiende a clasificar como negativo o neutral. En el caso de una matriz con sólo dos clases posibles se podrán ver los verdaderos negativos en la posición equivalente a un positivo de la otra clase que no se está analizando, falsos negativos, cuando la clase analizada se identifica como la otra posible o falsos positivos, cuando la otra clase a analizar es identificada como la que se está analizando.

Figura 34 Ilustración matriz de confusión (Lahby Mohamed)



Precision, la métrica de *classification_score,* indica cuántas clasificaciones correctas se han realizado de la clase. *Recall* da a conocer cuántos elementos de la clase se han encontrado del total de elementos de esa clase. F1-score calcula la media armónica entre *precision* y *recall*. Support es el número total de ocurrencias de la clase en la muestra con la que se trabaja. *Accuracy* es la precisión medida como aciertos totales entre el tamaño total de la muestra. Las medidas *macro averaged* es la media de esas medidas para todos

los valores dividido por el número de valores, es decir, si hay cuatro clases entre cuatro. Finalmente, las medidas *weighted* son aquellas en las que los valores se ponderan utilizando el número de ocurrencias de cada una de las clases y se divide por el tamaño de la muestra, son medias ponderadas.

Una vez se ha conseguido lanzar bert-as-service con el modelo multilingual, se plantea la posibilidad de hacer un ajuste a un modelo de BERT, esto se hace con un *notebook* de Antyukhov Denis Olegovich, sin embargo, los resultados obtenidos son peores que los del modelo multilingüe de Google base y por lo tanto no se explora en más profundidad esta opción, no deja de ser una opción interesante a explorar y en la que profundizar, sin embargo, no entraba dentro de los objetivos explorar en profundidad esta opción.

Figura 35 Comparativa de resultados para el modelo pre entrenado de Google (izquierda) y el que ha pasado por el ajuste (derecha).

	precision	recall	f1-score	support
1	0.22	0.23	0.22	43
2	0.23	0.19	0.21	37
3	0.54	0.50	0.52	134
4	0.42	0.48	0.45	89
accuracy			0.42	303
macro avg	0.35	0.35	0.35	303
weighted avg	0.42	0.42	0.42	303

```
[[10  4 19 10]
 [ 7  7 10 13]
 [20 11 67 36]
 [ 9  9 28 43]]
```

	precision	recall	f1-score	support
1	0.16	0.09	0.12	43
2	0.09	0.05	0.07	37
3	0.50	0.53	0.51	134
4	0.31	0.39	0.35	89
accuracy			0.37	303
macro avg	0.26	0.27	0.26	303
weighted avg	0.34	0.37	0.35	303

```
[[ 4  3 15 21]
 [ 6  2 15 14]
 [10 10 71 43]
 [ 5  7 42 35]]
```

En el proceso de entrenamiento se utiliza un corpus con un tamaño de cien millones de palabras, este corpus se obtiene de OPUS que contiene una gran colección de textos traducidos. Con este corpus se entrena al modelo utilizando los métodos clonados de github que se encuentran en (https://github.com/google-research/bert). Una vez se ha entrenado el modelo se almacenaba en Google Cloud Storage, de ahí se pasaba a Google Drive, posteriormente al equipo local y de ahí se sube al servidor donde se lanza como servicio con el siguiente comando:

Figura 36 Comando para lanzar un modelo ajustado en bert-as-service

```
bert-serving-start -model ~/Bert Model/uncased_L-12_H-768_A-12 -tuned_model_dir ~/bert-model -ckpt_name=model.ckpt-1000000 -num_worker=1 -max_seq_len=150
```

Puesto que bert-as-service permite lanzar un modelo ajustado indicando las localizaciones del modelo ajustado y del modelo original.

Una vez se ha conseguido lanzar el modelo multilingüe de BERT en el servidor se puede pasar a realizar análisis de sentimientos sobre el *dataset* del que se dispone.

El primer paso es el procesado del texto, este ya ha sido explicado en detalle con anterioridad por lo que no se va a volver a incidir en ello, una vez se ha procesado el texto se puede dividir el *dataset* en lote de entrenamiento y lote de prueba, para ello se utilizará la función de scikit-learn.

Figura 37 Función para la división del texto en muestra lotes

```
X_tr, X_val, y_tr, y_val = train_test_split(sentences, labels, test_size=0.15, random_state=123)
```

Una vez se ha dividido el *dataset* en muestras de entrenamiento y prueba, se puede pasar a BERT en el servidor para el proceso de *encoding*.

Figura 38 Llamada a bert-as-service para que se realice el encoding

```
X_tr_bert = bc.encode(X_tr.tolist())
X_val_bert = bc.encode(X_val.tolist())
```

Una vez se tienen los *encodings* es posible empezar a hacer pruebas con los distintos clasificadores de los que se ha hablado con anterioridad.

El primer paso es determinar qué clasificador de los anteriormente mencionados da los mejores resultados con la muestra de la que se dispone. Por tanto, para unas mismas muestras de entrenamiento y test se debe clasificar con cada uno de los clasificadores mencionados en el epígrafe anterior y comparar los resultados. En primer lugar, se puede descartar el RadiusNeighborsClassifier por lo que se ha comentado anteriormente, para los valores en los que se puede hacer uso del clasificador ya se clasifica todo como negativo (es una tendencia que se repite cuando se empiezan a modificar los parámetros de la mayoría de los clasificadores).

Figura 39 Salida de classification report y matriz de confusión

```
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        20
           2       0.00      0.00      0.00        20
           3       0.45      1.00      0.62        67
           4       0.50      0.02      0.04        45

    accuracy                           0.45       152
   macro avg       0.24      0.26      0.17       152
weighted avg       0.34      0.45      0.28       152

[[ 0  0 20  0]
 [ 0  0 19  1]
 [ 0  0 67  0]
 [ 0  0 44  1]]
```

Como se puede observar en los resultados, a pesar de tener un cien por cien de acierto dentro de la clase de negativo, estos resultados son únicamente para esta clase, cuando el modelo clasifica prácticamente todo como negativo es normal este resultado, sin embargo, nos interesa que se clasifiquen todos los sentimientos correctamente y por tanto se descarta este clasificador para futuras pruebas.

A continuación, se puede hablar de los dos clasificadores de *Naive Bayes*, sus resultados son los que se pueden ver a continuación utilizando las funciones con sus valores por defecto:

Figura 40 Resultados para BernouilliNB (izquierda) y GaussianNB (derecha)

	precision	recall	f1-score	support
1	0.27	0.15	0.19	20
2	0.28	0.35	0.31	20
3	0.64	0.66	0.65	67
4	0.49	0.51	0.50	45
accuracy			0.51	152
macro avg	0.42	0.42	0.41	152
weighted avg	0.50	0.51	0.50	152

```
[[ 3  4 10  3]
 [ 1  7  4  8]
 [ 4  6 44 13]
 [ 3  8 11 23]]
```

	precision	recall	f1-score	support
1	0.33	0.25	0.29	20
2	0.28	0.50	0.36	20
3	0.62	0.57	0.59	67
4	0.47	0.42	0.45	45
accuracy			0.47	152
macro avg	0.43	0.43	0.42	152
weighted avg	0.50	0.47	0.48	152

```
[[ 5  3  9  3]
 [ 2 10  2  6]
 [ 5 12 38 12]
 [ 3 11 12 19]]
```

Como podemos observar por los resultados, empieza a evidenciarse el efecto del desequilibrio entre los distintos sentimientos en la muestra, la tendencia a clasificar como negativo y positivo es muy grande como consecuencia de sus tamaños respecto a la muestra, además si se presta atención a la columna *support*, se puede ver que hay una diferencia entre los tamaños de muestras de la prueba entre las distintas clases.

Por lo que respecta a los resultados, como consecuencia del peso de positivo y negativo, los resultados para neutral y none son bajos, además podemos ver en la matriz de confusión que la confusión entre todas las clases y negativo y positivo tiene un peso importante, para neutral es un sesenta y cinco por ciento de los casos, para none un sesenta por ciento, mientras que en el caso de negativo y positivo, la clase alterna (negativo para positivo y positivo para negativo), supone en total el mismo número de confusiones que el conjunto del resto de clases (en Bernoulli), la menor precisión de Gaussian da un peso mayor a none.

Finalmente, limitándonos a analizar la precisión, una vez comprendido que el *dataset* supone un problema y que se deberán realizar otros análisis en el modelo. Se comprueba que Bernoulli tiene mejor precisión si se pondera por número de apariciones, pero una peor precisión global, esto es debido a que para none y neutral sus resultados son peores

que para Gaussian. Si tenemos en cuenta que el objetivo es clasificar todos los sentimientos no sólo los más comunes sería conveniente la utilización de GaussianNB.

Pasando ahora a *Random Forest*, en este caso se tienen muchos más parámetros para poder configurar el clasificador, del efecto de tener un alto número de árboles ya se ha hablado en el epígrafe anterior, en este caso sin introducir más que el número de árboles que se quiere utilizar, en este caso 15, se obtienen los siguientes resultados:

Figura 41 Resultados para random forest con 15 árboles

```
              precision    recall  f1-score   support

           1       0.29      0.10      0.15        20
           2       0.10      0.05      0.07        20
           3       0.52      0.72      0.60        67
           4       0.37      0.36      0.36        45

    accuracy                           0.44       152
   macro avg       0.32      0.31      0.30       152
weighted avg       0.39      0.44      0.40       152

[[ 2  2 10  6]
 [ 0  1  9 10]
 [ 5  3 48 11]
 [ 0  4 25 16]]
```

Se vuelve a hacer evidente el problema mencionado anteriormente de la distribución de clases en el *dataset*. Más allá de eso la precisión se limita a un cuarenta por ciento si se toma en cuenta el número de casos para ponderar y obteniendo malos resultados para las clases menos frecuentes. A pesar de que se puede utilizar class_weight para dar ponderaciones a las clases el desequilibrio en el *dataset* no permite lograr un resultado considerablemente mejor.

La siguiente prueba se realiza sobre la regresión logística, de nuevo se obtienen resultados similares, con una precisión alta para negativo, pero baja en todas las demás clases. Sin embargo, los resultados son peores que ambos *naive Bayes*.

Figura 42 Resultados regresión logística.

```
              precision    recall  f1-score   support

           1       0.21      0.15      0.18        20
           2       0.29      0.35      0.32        20
           3       0.66      0.69      0.67        67
           4       0.41      0.40      0.40        45

    accuracy                           0.49       152
   macro avg       0.39      0.40      0.39       152
weighted avg       0.48      0.49      0.48       152

[[ 3  4  8  5]
 [ 2  7  3  8]
 [ 1  7 46 13]
 [ 8  6 13 18]]
```

El siguiente clasificador sería *label propagation*, los resultados vuelven a empeorar, utilizando la configuración que mejores resultados ha dado, de nuevo los problemas del *dataset* lastran los resultados, por otro lado, también pueden verse afectados por las oraciones que se están analizando, los tuits presentan dificultades, pero la matriz de confusión parece indicar que se trata de problemas con la distribución.

Finalmente queda el clasificador KNeighborsClassifier, este se configura con 3 vecinos para que la clasificación no tienda más aún a negativo o positivo, en este caso se obtienen los peores resultados, en los que no se obtiene una buena clasificación ni en el caso negativo.

Figura 43 Resultados KNeighbordClssifier

	precision	recall	f1-score	support
1	0.20	0.15	0.17	20
2	0.20	0.10	0.13	20
3	0.46	0.49	0.48	67
4	0.30	0.38	0.34	45
accuracy			0.36	152
macro avg	0.29	0.28	0.28	152
weighted avg	0.35	0.36	0.35	152

```
[[ 3  0 14  3]
 [ 3  2  4 11]
 [ 4  5 33 25]
 [ 5  3 20 17]]
```

Para tratar de comprender mejor los resultados anteriores se realizan tres experimentos adicionales distintos, el primero hacer una clasificación uno contra el resto para todas las clases, también se realizan clasificaciones binarias para todos los pares y se prueba el modelo sobre un dataset distinto.

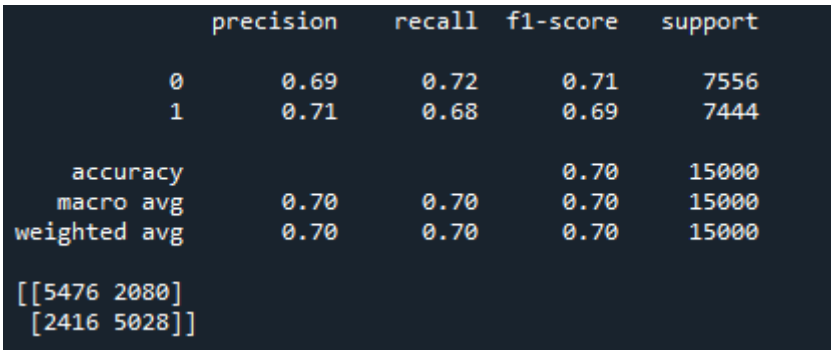
En primer lugar, cabe mencionar el experimento realizado sobre un dataset distinto, sin entrar en grandes detalles puesto que la finalidad era comprobar el funcionamiento del modelo que se estaba utilizando, esto incluye también cómo se estaba procesando la información.

El experimento utiliza dos dataset distintos (en inglés) de *reviews* de películas procedentes de Kaggle, se puede acceder a ellos a través de: https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-

reviews#IMDB%20Dataset.csv y a través de https://www.kaggle.com/c/word2vec-nlp-tutorial/data.

Una vez se han unido los dos *datasets* distintos se puede realizar la prueba sobre las setenta y cinco mil *reviews* que contiene. En este dataset se cuenta únicamente con dos clases, las *reviews* positivas codificadas con un "1" y las *reviews* negativas codificadas con un "0". Los resultados que arroja la prueba nos confirman que el modelo funciona pero que las limitaciones a la secuencia de entrada en bert-as-service penalizan los resultados, sin embargo, estos son buenos resultados que invitan a profundizar en el modelo con el *dataset* original y confirmar que se debe a la distribución de las clases en el mismo.

Figura 44 Prueba sobre el nuevo dataset de reviews de películas

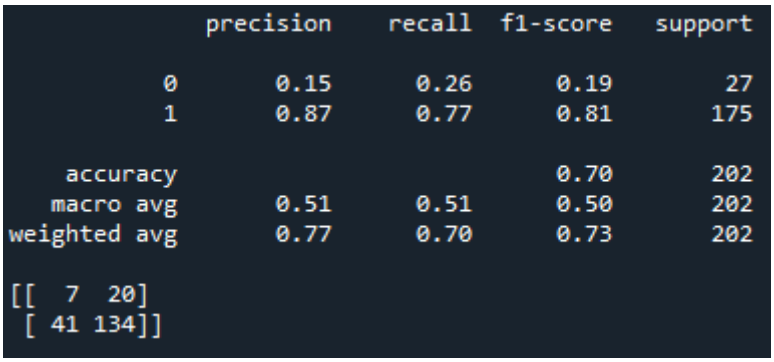


	precision	recall	f1-score	support
0	0.69	0.72	0.71	7556
1	0.71	0.68	0.69	7444
accuracy			0.70	15000
macro avg	0.70	0.70	0.70	15000
weighted avg	0.70	0.70	0.70	15000

```
[[5476 2080]
 [2416 5028]]
```

Por lo que respecta a los experimentos de clasificación uno contra el resto, se muestran a continuación, por orden de neutral, none, negativo y positivo.

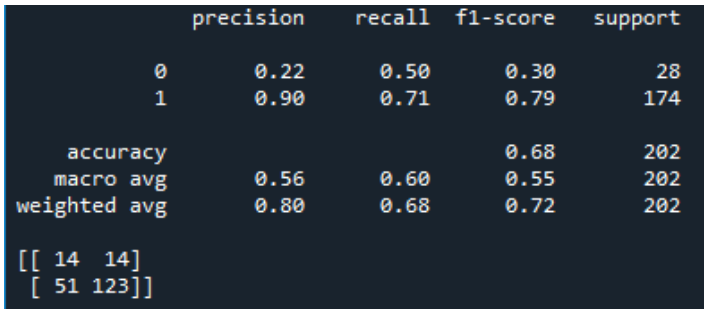
Figura 45 Resultados neutral contra el resto



	precision	recall	f1-score	support
0	0.15	0.26	0.19	27
1	0.87	0.77	0.81	175
accuracy			0.70	202
macro avg	0.51	0.51	0.50	202
weighted avg	0.77	0.70	0.73	202

```
[[  7  20]
 [ 41 134]]
```

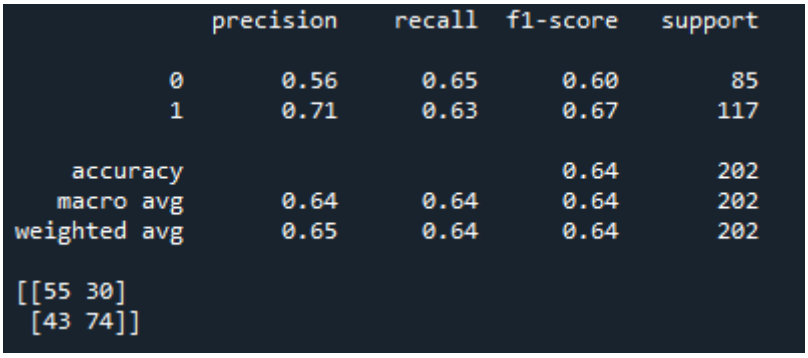
Figura 46 Resultados none contra el resto



	precision	recall	f1-score	support
0	0.22	0.50	0.30	28
1	0.90	0.71	0.79	174
accuracy			0.68	202
macro avg	0.56	0.60	0.55	202
weighted avg	0.80	0.68	0.72	202

```
[[ 14  14]
 [ 51 123]]
```

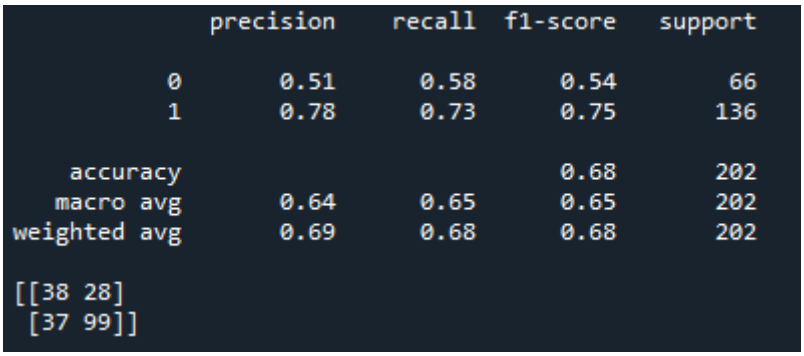
Figura 47 Resultados negativo contra el resto



	precision	recall	f1-score	support
0	0.56	0.65	0.60	85
1	0.71	0.63	0.67	117
accuracy			0.64	202
macro avg	0.64	0.64	0.64	202
weighted avg	0.65	0.64	0.64	202

```
[[55 30]
 [43 74]]
```

Figura 48 Resultados positivo contra el resto



	precision	recall	f1-score	support
0	0.51	0.58	0.54	66
1	0.78	0.73	0.75	136
accuracy			0.68	202
macro avg	0.64	0.65	0.65	202
weighted avg	0.69	0.68	0.68	202

```
[[38 28]
 [37 99]]
```

Como se puede ver en este caso los resultados mejoran en todos los casos con una precisión global sin ponderar de más del cincuenta por ciento en todos los casos, a pesar de ello el bajo número de muestras que se tienen de neutral y de none sigue dando como resultado una baja precisión a la hora de predecirlos en este caso.

Por tanto, como consecuencia de esta prueba podemos concluir que BERT y el clasificador (GaussianNB en este caso) están funcionando y el problema se encuentra en la distribución del dataset (aunque posiblemente la forma en la que la gente escribe tuits también dificulta el obtener mejores resultados).

Finalmente se han realizado comparaciones binarias entre todos los posibles pares de clases, sin embargo, en lugar de mostrar los resultados de todos, se mostrarán los dos

pares más interesantes, en este caso se trata de los pares neutral y none y positivo y negativo.

Neutral y none tienen tamaños de muestra similares, esto supone que no van a sufrir los efectos del *dataset* desequilibrado. Se vuelve a utilizar GaussianNB para esta prueba, el objetivo es determinar la capacidad de clasificación de la configuración y confirmar que los resultados de las primeras pruebas se debían a las características del *dataset*.

Figura 49 Análisis utilizando sólo none y neutral

```
              precision    recall  f1-score   support

           1       0.62      0.76      0.68        21
           2       0.83      0.71      0.76        34

    accuracy                           0.73        55
   macro avg       0.72      0.73      0.72        55
weighted avg       0.75      0.73      0.73        55

[[16  5]
 [10 24]]
```

Como se puede ver, la precisión alcanza niveles por encima del setenta por ciento cuando se calcula la media sin tener en cuenta la distribución de muestras. Podemos comprobar con esta prueba que el modelo utilizando BERT y posteriormente un clasificador es capaz de diferenciar entre none y neutral y clasificar correctamente, llegando a tener más de un setenta por ciento de acierto con una muestra relativamente pequeña.

Por lo que respecta a positivo y negativo volverá a haber un desequilibrio en la muestra al tener más casos negativos, sin embargo, servirá para comprobar la capacidad para diferenciar entre ambos que son los casos más comunes en Twitter.

Figura 50 Comparativa entre negativo y positivo.

```
              precision    recall  f1-score   support

           3       0.70      0.71      0.71        77
           4       0.68      0.66      0.67        71

    accuracy                           0.69       148
   macro avg       0.69      0.69      0.69       148
weighted avg       0.69      0.69      0.69       148

[[55 22]
 [24 47]]
```

Una vez más podemos observar una precisión cercana al setenta por ciento, la confusión entre las clases es baja.

Los resultados de las comparativas binarias demuestran que el clasificador de Naive Bayes gaussiano aplicado al *encoding* de BERT funciona, es posible realizar análisis de sentimientos sobre tuits en español, sin embargo, los resultados se ven lastrados por la distribución del *dataset*.

Una vez realizadas las comparaciones binarias que ayudan a entender con mayor profundidad el problema, podemos empezar a extraer conclusiones del trabajo realizado.

4. CONCLUSIONES Y FUTUROS TRABAJOS

4.1 Conclusiones

De manera similar a como se hizo con los objetivos las conclusiones deben ir divididas en dos líneas distintas, aquellas relacionadas con BERT y aquellas relacionadas con el análisis de sentimientos.

En primer lugar, centrándonos en el análisis de sentimientos, se deben extraer varias conclusiones, la primera es que la pobre calidad del *dataset* disponible ha sido un impedimento a lo largo de todo el trabajo realizado, la dificultad para encontrar un dataset de contenidos similares hace que haya sido una necesidad trabajar con él.

Una vez se ha entendido que el *dataset* supone un problema, el análisis de sentimientos, especialmente si se centra el análisis en el resultado de los análisis binarios con muestras similares se obtiene resultados positivos, en estos casos, las características del *dataset* con el que se trabaja tienen un menor impacto.

Por tanto, es posible decir que se ha podido realizar análisis de sentimientos en español, y que además se ha hecho sobre tuits, lo cual añade una capa de dificultad por la forma en la que están escritos, en la que abundan las faltas de ortografía y el sarcasmo. Por lo tanto, a pesar de que no se puede decir que se ha conseguido por completo el objetivo al no haber conseguido resultados a la altura de lo que se espera con BERT al analizar todos los posibles sentimientos a la vez, ha sido exitoso en los resultados cuando se ha conseguido eliminar el efecto negativo de la distribución del *dataset*.

Por lo que respecta a BERT, ha sido posible utilizarlo para el análisis de sentimientos en español, el modelo pre entrenado de Google, que puede trabajar con más de 100 idiomas ha hecho esto posible, por tanto, se puede aplicar este modelo al español. Además, en lo que respecta a la dificultad a la hora de poder hacer uso de BERT, teniendo en cuenta la posibilidad de hacer uso de bert-as-service, resulta relativamente simple, siempre que se pueda hacer uso de un servidor con una tarjeta gráfica.

Se puede decir por tanto que los objetivos en lo que respecta a BERT se han conseguido, se ha conseguido trabajar con el modelo a través de bert-as-service y se ha podido usar sobre texto en castellano.

4.2 Futuros trabajos

Los trabajos que elaboren sobre esta base pueden seguir tres líneas distintas que lograrían mejorar los resultados obtenidos.

En primer lugar, a lo largo de todo el trabajo, el *dataset* y la distribución de los datos en el mismo has sido un impedimento en la obtención de buenos resultados, a su vez, la falta de un dataset de características similares, compuesto por tuits en español, sin necesidad de que traten sobre un tema concreto, que tengan una clasificación por sentimiento para poder ser utilizados para entrenamiento ha hecho que sea imposible mejorar los datos con los que se trabajaba. Por tanto, una forma de mejorar tanto este como cualquier otro trabajo sobre el mismo tema sería la creación de un *dataset* de mejor calidad que el que ha tenido que ser utilizado para este trabajo.

Otra opción para ampliar sobre el trabajo está en el clasificador, en este trabajo se han utilizado únicamente clasificadores que ya existían en scikit-learn, una forma de mejorar los resultados sería la implementación de un clasificador. Una opción no explorada para el clasificador en este caso es la implementación de una red neuronal propia para realizar esta clasificación. Aprovechando librerías como TensorFlow, sería posible construir un clasificador propio que lograría con seguridad mejorar los resultados obtenidos por los clasificadores usados en este trabajo, sin embargo, este no era uno de los objetivos de este trabajo.

Por último, se puede avanzar en el trabajo profundizando sobre BERT o alternativas a BERT. Desde que BERT está disponible han aparecido nuevas alternativas que han logrado mejores resultados, entre ellos se puede mencionar RoBERTa, que trabaja sobre BERT, Transformer-XL, GPT-2, Ernie, XLNet, CTRL… todos ellos ofrecen una nueva base sobre la que trabajar que posiblemente mejoren los resultados obtenidos.

Otro enfoque en la ampliación del trabajo sobre BERT es continuar explorando las posibilidades que ofrecen PyTorch y HuggingFace a la hora de realizar un ajuste de un modelo de BERT, en este trabajo se ha intentado con escaso éxito y se ha tenido que trabajar con el modelo de BERT multilingüe base de Google, un ajuste sobre este modelo posiblemente arroje también grandes mejoras respecto a los resultados obtenidos.

Por tanto, a partir de este trabajo es posible avanzar en tres direcciones distintas, una sobre el *dataset*, otra sobre el clasificador y por último sobre BERT.

5. BIBLIOGRAFÍA

1. Alammar, J. (2018). The illustrated BERT, ELMo, and co. (how NLP cracked transfer learning). Retrieved from http://jalammar.github.io/illustrated-bert/
2. Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5-32. doi:10.1023/A:1010933404324
3. *Transformer neural networks - EXPLAINED! (attention is all you need)*. CodeEmporium (Director). (2020, -01-13).[Video/DVD] Retrieved from https://www.youtube.com/watch?v=TQQlZhbC5ps
4. Cortez Vasquez, A., Vega Huerta, H., & Pariona Quispe, J. (2009). Procesamiento de lenguaje natural. *Revista de investigación de sistemas e informática, 6*(2), 45.
5. D'Andrea, A., Ferri, F., Grifoni, P., & Guzzo, T. (2015). Approaches, tools and applications for sentiment analysis implementation. *International Journal of Computer Applications, 125*, 26-33. doi:10.5120/ijca2015905866
6. Devlin, J., & Chang, M. (2018, 2/Nov/). Open sourcing BERT: State-of-the-art pre-training for natural language processing. Retrieved from http://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html
7. Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. Retrieved from https://arxiv.org/abs/1810.04805v2
8. Downey, A. B.Think python: How to think like a computer scientist (2nd Edition ed.)
9. Estadística de la probabilidad: Teorema de bayes. (). Retrieved from http://karineydiana.blogspot.com/p/teorema-de-bayes.html
10. Howard, J., & Ruder, S. (2018). Universal language model fine-tuning for text classification. Retrieved from https://arxiv.org/abs/1801.06146v5
11. Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2015). Character-aware neural language models. Retrieved from https://arxiv.org/abs/1508.06615v4
12. Koehrsen, W. (2018). An implementation and explanation of the random forest in python. Retrieved from https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76

13. KULSHRESTHA, R. (2019). NLP 101: Word2Vec — skip-gram and CBOW. Retrieved from https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314

14. Lahby Mohamed.Figure 6: Confusion matrix illustration. Retrieved from https://www.researchgate.net/figure/confusion-matrix-illustration_fig4_331396279

15. Loh, W. (2011). *Classification and regression trees* John Wiley & Sons, Inc.

16. McCormick, C., & Ryan, N. (2019). BERT fine-tuning tutorial with PyTorch. Retrieved from http://mccormickml.com/2019/07/22/BERT-fine-tuning/

17. Medhat, W., Hassan, A., & Korashy, H. (2014). *Sentiment analysis algorithms and applications: A survey* doi:https://doi.org/10.1016/j.asej.2014.04.011

18. Mihail, E. (2018). Deep contextualized word representations with ELMo. Retrieved from https://www.mihaileric.com/posts/deep-contextualized-word-representations-elmo/

19. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 26* (pp. 3111–3119) Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

20. Nicholson, C.A beginner's guide to Word2Vec and neural word embeddings. Retrieved from http://pathmind.com/wiki/word2vec

21. Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. doi:10.3115/v1/D14-1162

22. Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. Retrieved from https://arxiv.org/abs/1802.05365v2

23. Radford, A., Narasimhan, K., Salimans, T. & Sutskever, I. (2018). Improving language understanding
by generative pre-training. Retrieved from https://openai.com/blog/language-unsupervised/

24. Rizvi, M. S. Z. (2019). Demystifying BERT: The groundbreaking NLP framework. Retrieved from https://medium.com/analytics-vidhya/demystifying-bert-the-groundbreaking-nlp-framework-8e3142b3d366

25. Roman, V. (2019). Algoritmos naive bayes: Fudamentos e implementación. Retrieved from https://medium.com/datos-y-ciencia/algoritmos-naive-bayes-fudamentos-e-implementaci%C3%B3n-4bcb24b307f

26. Rong, X. (2014). Word2vec parameter learning explained. Retrieved from https://arxiv.org/abs/1411.2738v4

27. scikit-learn. (a). Naive bayes. Retrieved from https://scikit-learn.org/stable/modules/naive_bayes.html#bernoulli-naive-bayes

28. scikit-learn. (b). Sklearn.naive_bayes.BernoulliNB. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html#sklearn.naive_bayes.BernoulliNB

29. scikit-learn. (c). Sklearn.naive_bayes.GaussianNB. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB

30. scikit-learn. (2011a). Linear models. Retrieved from https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

31. scikit-learn. (2011b). Nearest neighbors. Retrieved from https://scikit-learn.org/stable/modules/neighbors.html#classification

32. scikit-learn. (2011c). Semi-supervised. Retrieved from https://scikit-learn.org/stable/modules/label_propagation.html#label-propagation

33. scikit-learn. (2011d). Sklearn.ensemble.RandomForestClassifier. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier

34. scikit-learn. (2011e). Sklearn.linear_model.LogisticRegression. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression

35. scikit-learn. (2011f). Sklearn.neighbors.KNeighborsClassifier. Retrieved from https://scikit-

learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier

36. scikit-learn. (2011g). Sklearn.neighbors.RadiusNeighborsClassifier. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.RadiusNeighborsClassifier.html#sklearn.neighbors.RadiusNeighborsClassifier
37. scikit-learn. (2011h). Sklearn.semi_supervised.LabelPropagation. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.semi_supervised.LabelPropagation.html#sklearn.semi_supervised.LabelPropagation
38. Scikit-learn, herramienta básica para el data science en python. (2018, -08-06T07:00:40+00:00). Retrieved from https://www.master-data-scientist.com/scikit-learn-data-science/
39. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention is all you need. Retrieved from https://arxiv.org/abs/1706.03762v5
40. What is python? executive summary. Retrieved from https://www.python.org/doc/essays/blurb/
41. Xiao, H. (2019a). Bert-as-service. Retrieved from https://github.com/hanxiao/bert-as-service#speed-wrt-num_client
42. Xiao, H. (2019b). Serving google BERT in production using tensorflow and ZeroMQ . Retrieved from https://hanxiao.io/2019/01/02/Serving-Google-BERT-in-Production-using-Tensorflow-and-ZeroMQ/
43. Zhang, Q., & Zhang, Q. (2019). An overview of normalization methods in deep learning. Retrieved from https://www.zhqiang.org/normalization/

6. ANEXOS

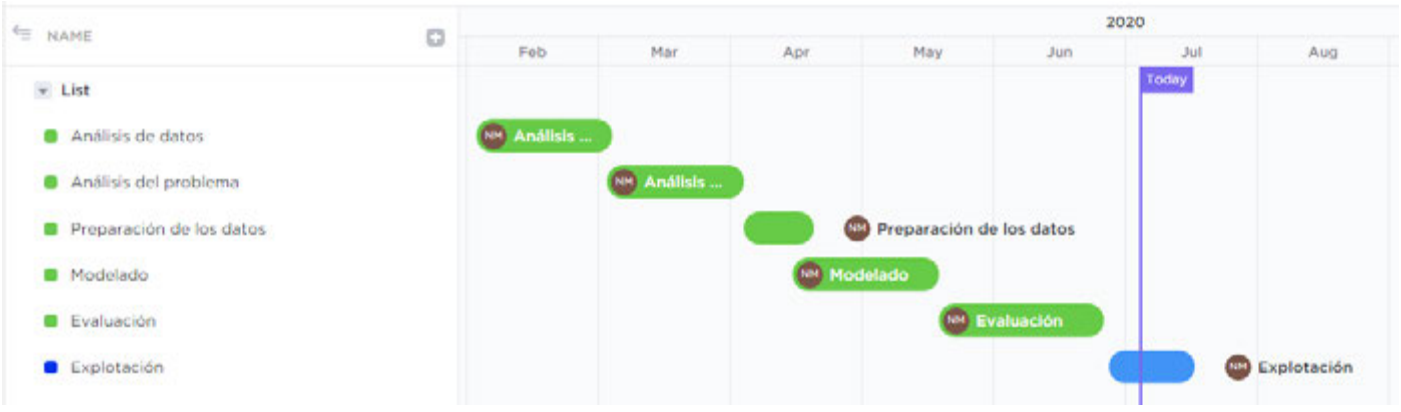
6.1 Planificación

Se ha seguido la metodología CRISP-DM (Chapman, P, Clinton, J. Khabaza, T. Reinartz, T. Rüdiger, W. The CRISP-DM Process Piatetski-Shapiro G., Frawley W.J: Knowledge discovery in databases. Ed. AAAI/MIT Press, 1991).

Comprende 6 fases distintas:

- Análisis del problema: se convierten los objetivos y requisitos en objetivos técnicos y planificación.
- Análisis de datos: identifica la calidad de los datos y se elaboran las primeras hipótesis.
- Preparación de los datos: para que puedan ser tratados posteriormente por las técnicas de modelado.
- Modelado: se seleccionan las técnicas más adecuadas para el proyecto.
- Evaluación: se evalúa el modelado en función de los criterios técnicos del problema.
- Explotación: presentar resultados logrando un incremento del conocimiento. Mantenimiento de la aplicación y difusión de los resultados

Figura 51 Diagrama de Gantt del trabajo



Se han omitido dependencias debido a que el trabajo también ha requerido de la colaboración con terceros que han permitido el paso entre fases.

6.2 Presupuesto

Para la realización de un proyecto similar serán necesarias 130 jornadas laborales de un analista de datos, esto supone un salario anual medio de entre 30000€ y 50000€ asumiendo un perfil junior.

El coste del servidor con TPU para las tareas de entrenamiento y del almacenamiento del modelo y los equipos informáticos (la amortización correspondiente al periodo de tiempo). Dentro del almacenamiento se incluyen los costes de realizar las transacciones de y a la base de datos.

Además, se deberán tener en cuenta los costes de electricidad e internet, basados en costes medios para las facturas.

Concepto	Cantidad	Unidad	Coste unitario	Coste Total
Analista de datos	1040	Horas	19,61€	20.394,40€
Servidor	3936	Horas	32,00€	125.952,00€
Almacenamiento	20	GB	0,05€	41,00€
Equipos informáticos	1	Ordenador	183,33€	223,33€
Electricidad	31,2	KW	0,90€	68,08€
Internet	5,5	Meses	40,00€	260,00€
Total				146.938,81€

Anexo final

DECLARACION DE ORIGINALIDAD

Yo, Iago Collarte González declaro que el TFG " Procesamiento del lenguaje natural con BERT: Análisis de sentimientos en tuits " es totalmente original mío, que no ha sido presentado en ninguna otra universidad como TFG y que todas las fuentes que han sido utilizadas han sido adecuadamente citadas y aparecen en las referencias bibliográficas.

Colmenarejo, a 16 de julio de 2020

Firma: