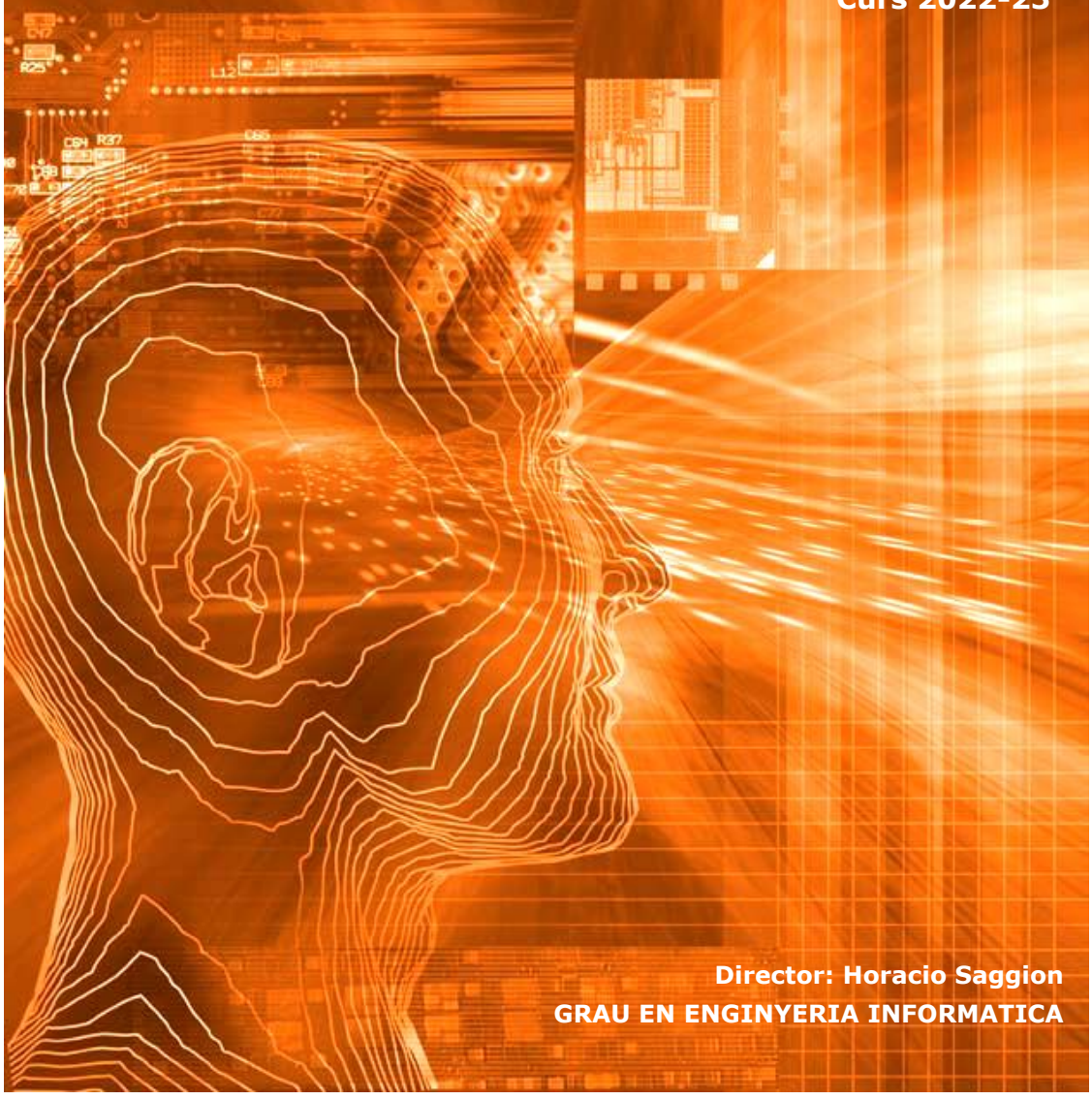
Detección de ironía en el texto en español

Casanovas Pérez, Èric

Curs 2022-23



Director: Horacio Saggion

GRAU EN ENGINYERIA INFORMATICA

Universitat Pompeu Fabra Barcelona
Escola d'Enginyeria

Treball de Fi de Grau

Agraïments

Vull agrair a la meva família i amics per tot el suport emocional que m'han donat, i en general a les oportunitats que se m'han brindat al poder treballar en un tema que m'interessa.

Resumen

Durante los últimos años, ha habido un notable desarrollo de las tecnologías de procesamiento del lenguaje natural para diversas tareas como la clasificación de textos, extracción de información y generación automática de lenguaje natural, por mencionar algunas. Sin embargo, el problema de detectar lenguaje figurado como la ironía no ha recibido tanta atención por parte de la comunidad científica. En este trabajo de fin de grado, proponemos el desarrollo de un modelo de procesamiento del lenguaje natural con el objetivo de detectar la presencia de ironía en textos en español. Para lograr esto, hemos adoptado una metodología de aprendizaje supervisado basada en redes neuronales transformadoras y la técnica conocida como fine-tuning. Específicamente, nos basamos en el modelo publicado por Google en 2018 llamado BERT. Los datos utilizados para entrenar el modelo forman parte del conjunto de datos proporcionado por los organizadores de IroSvA 2019, una de las tareas más recientes para detectar ironía en variantes del español que involucró a varias universidades. Además, analizamos la capacidad de nuestro sistema para detectar ironía en un conjunto de datos recién consolidado con la ayuda de dos expertos en lenguaje figurado. Los resultados obtenidos demuestran la efectividad de este modelo en la detección de ironía en español y pueden abrir el camino para futuras investigaciones en el campo.

Resum

Durant els últims anys, s'ha produït un notable desenvolupament de les tecnologies de processament del llenguatge natural per a diverses tasques com la classificació de textos, l'extracció d'informació i la generació automàtica de llenguatge natural, per citar-ne algunes. No obstant això, el problema de detectar llenguatge figurat com la ironia no ha rebut tanta atenció per part de la comunitat científica. En aquest treball de fi de grau, proposem el desenvolupament d'un model de processament del llenguatge natural amb l'objectiu de detectar la presència d'ironia a textos en espanyol. Per aconseguir-ho, hem adoptat una metodologia d'aprenentatge supervisat basada en xarxes neuronals transformadores i la tècnica ben coneguda com a fine-tuning. Específicament, ens basem en el model publicat per Google el 2018 anomenat BERT. Les dades utilitzades per entrenar el model formen part del conjunt de dades proporcionat pels organitzadors de l'IroSvA 2019, una de les tasques més recents per detectar ironia en variants de l'espanyol que va implicar diverses universitats. A més, analitzem la capacitat del nostre sistema per detectar ironia en un conjunt de dades recentment consolidat amb l'ajuda de dos experts en llenguatge figurat. Els resultats obtinguts demostren l'efectivitat d'aquest model en la detecció d'ironia en espanyol i poden obrir el camí per a futures investigacions en el camp.

Abstract

In recent years, there has been a notable development of natural language processing technologies for various tasks such as text classification, information extraction, and automatic natural language generation, to name a few. However, the problem of detecting figurative language like irony has not received as much attention from the scientific community. In this undergraduate thesis, we propose the development of a natural language processing model with the objective of detecting the presence of irony in Spanish text. To accomplish this, we have adopted a supervised learning methodology based on transformer neural networks and the well-known fine-tuning technique. Specifically, we build upon the model published by Google in 2018 called BERT. The data used to train the model is part of the dataset provided by the organizers of IroSvA 2019, one of the most recent tasks for detecting irony in Spanish variants that involved several universities. Additionally, we analyze the capability of our system to detect irony in a newly consolidated dataset with the assistance of two experts in figurative language. The obtained results demonstrate the effectiveness of this model in detecting irony in Spanish and can pave the way for future research in the field.

Índice

1. INTRODUCCIÓN

La ironía ha sido objeto de estudio a lo largo de la historia, y ha dado lugar a diversas definiciones e interpretaciones que difieren entre sí.

En concreto, la definición de ironía que dio el filósofo alemán Friedrich Schlegel en su obra "Athenaeum Fragments", la describe como la implicación de elementos contradictorios u opuestos para crear tensión creativa y transmitir una visión más profunda de la realidad [23].

Sobre todo, teniendo en cuenta que esta ironía no convella contexto muy profundo. Por ejemplo, un caso de supuesta ironía que no contemplaremos podría ser: "¡qué buen día hace hoy!", en el contexto de que las condiciones climáticas son desfavorables. Sin embargo, en una frase donde el contexto está presente, o es menos dependiente de un valor externo, sí que sería considerada como irónica: "Otro día de lluvias, si es que me encanta llegar empapado a la universidad,".

1.1 Motivación

La ironía, como expresión humana, es única y refleja nuestra habilidad para ver más allá de lo literal, capturando la complejidad de nuestras experiencias y transmitiendo mensajes profundos en forma de paradojas lingüísticas.

Es por ello, que poder detectar este concepto tan humano y ser capaz de detectar la ironía automáticamente, supondría un gran avance en muchos ámbitos.

Uno de ellos, la detección de sentimientos en el texto, y la capacidad de conocer la verdadera emoción del redactor del texto, lo cual puede resultar muy efectivo al momento de reconocer noticias falsas, sensacionalidad o subjetividad y muchas más, que contribuiría a mejorar la calidad y la veracidad de la información a la que estamos expuestos.

Además, la detección automática de ironía tiene aplicaciones en la generación de texto automático. Al ser capaz de reconocer y utilizar la ironía, podríamos conseguir ese punto extra de riqueza y humanidad en los textos automáticamente generados.

En resumidas cuentas, este avance en la capacidad automatizar un concepto tan inherente a nuestra naturaleza, abre nuevas posibilidades en campos como el análisis de sentimientos en el texto, la detección de calidad o de desinformación de un texto y la generación de texto automático, entre muchos otros.

1.2 Planteamiento del problema

Este trabajo de fin de grado, trata de implementar un modelo de detección automática de ironía en el texto en español con suficiente efectividad, pasando por todos los pasos que este proceso conlleva. Además, trata de exponer la efectividad de varios experimentos realizados en el proceso de mejora de este modelo.

1.3 Objetivos

Los objetivos que propone este trabajo de fin de grado varían entre los propuestos por mi supervisor y las ideas que he tenido durante el proyecto de cara a los problemas y/o las propuestas que se presentaban, pero para mayor claridad, se listan todos juntos.

- Encontrar, crear o concebir un modelo de detección automática de ironía en el texto en español.
- Entrenar y evaluar el modelo.
    - Crear un código en Python capaz de completar el proceso de entrenamiento del modelo.
    - Crear un código en Python capaz de evaluar el modelo y de generar resultados visuales.
- Programar la evaluación de los datos generada a través de "data augmentation".
- Encontrar una buena combinación de hiperparámetros con tal de mejorar la capacidad del modelo.
- Obtener buenos resultados en cuanto a calidad del modelo.
- Asistir en el proceso de creación de un nuevo dataset.
    - Preprocesar los textos iniciales y elegir los batches que los anotadores analizarán.
    - Generar documentos Excel con la finalidad de juntar la data de los batches, y de agilizar el proceso de anotación de estos.
    - Evaluar la similitud entre las anotaciones proporcionadas por los anotadores, con tal de conseguir un dataset unificado.
    - Generar el dataset final, en formato XML y crear un programa capaz de convertir los datos en un dataframe.

1.4 Cronología

El 28 de septiembre del 2022 tuve la primera reunión con mi supervisor, Horacio para hablar de mi tema de TFG, y concretar las bases y objetivos de este. Dada mi inexperiencia en el tema de DL ("deep learning") y PLN (procesamiento de lenguaje natural), decidimos que primero tomaría un tiempo aprendiendo las bases, leyendo sobre temas relacionados y haciendo pruebas antes de empezar a profundizar más en el temario.

Mientras tanto, empezamos con la creación del nuevo dataset aunque hubo un inconveniente y se tuvo que retrasar. El primer equipo de anotadores de manera imprevista se retiró y mi supervisor tuvo que buscar otros anotadores. Esto provocó 1 mes y medio de retraso aproximadamente.

En la siguiente figura, se muestra un GanntChart que enseña la duración aproximada de cada tarea realizada.

figura 1: GannChart sobre la cronología de las tareas del proyecto

Cabe mencionar que aun habiendo tareas que tengan corta duración, como es escribir la memoria, el tiempo dedicado puede ser mayor que a otras, como es el ejemplo de probar experimentos.

2. ESTADO DEL ARTE

En este apartado, trato de relatar y recopilar información sobre el trabajo anteriormente realizado para así poder construir mi modelo con una buena base.

2.1 Aprendizaje automático tradicional para detección de ironía.

Detectar la ironía es una tarea ambigua hasta para los humanos, con lo que para poder detectarla automáticamente, varios trabajos anteriores tratan de clasificar las variantes de esta para luego poder identificarla. Entre estos trabajos se encuentra el paper "*From humor recognition to irony detection: The figurative language of social media*" 2012, que trata de identificar componentes clave de la ironía y el humor para su procesamiento automático y que luego evalúa usando un modelo de "decision trees" [21].

Los componentes con los que trabaja son la ambigüedad, la polaridad, el contenido emocional y la sorpresa o el imprevisto, y obtiene resultados que confirman la utilidad de este tipo de información para caracterizar estos dispositivos (la ironía y el humor) [21].

También encontramos otros trabajos como el del paper "Modelling Irony in Twitter" 2014, que basan su modelo en características léxicas (la frecuencia, las diferencias orales/escritas, los sentimientos, la ambigüedad, la intensidad, la sinonimia y la estructura), más específicamente donde estas características están basadas en el concepto de la "inesperación" que podríamos definir como sorpresa o imprevisto, que confirman como clave de las afirmaciones irónicas [3] . En concreto, usan "random forest" y "decision trees" para clasificar la ironía [3].

Este equipo confirma obtener un modelo base que rinde mejor que un enfoque basado en "Bag of Words" entre dominios y que consigue un rendimiento de vanguardia, es decir, el del estado del arte en el momento de la publicación de esta investigación. Aun así, enfatizan que el aspecto de la ambigüedad es aún débil en la investigación y que precisa de mejoras [3].

De manera muy superficial, podemos ver como los modelos que usan Aprendizaje Automático tradicional, centran la investigación en entender, que es la ironía para poder detectarla, pasando por los distintos casos de esta o por los conceptos clave que la originan [21][3].

2.2 Tareas de detección de ironía

Con fin de desarrollar más el campo de la detección de ironía, varias tareas para esto han sido propuestas durante los últimos años, entre ellas IroSvA 2019 y SemEval 2018 Task 3.

2.2.1 IroSvA 2019

En 2019, una tarea para la detección de ironía en variantes del español fue propuesta. Esta propina la clasificación automática de mensajes cortos de twitter y de comentarios de diarios como irónicos o no irónicos [5]. Estos mensajes cortos se reparten en tres variantes del español, la cubana, la mexicana y la española (o de España) , y la idea era experimentar y tratar de obtener los mejores resultados. La diferencia entre las tareas de años anteriores es que las anotaciones venían con un contexto, lo cual puede propiciar en más resultados y dar más juego a más experimentación [5].

En esta tarea participaron 12 equipos con sus respectivas entregas [5] de las cuales, para este trabajo decidí centrarme en las dos con mejores resultados. En la tabla 1 se muestran los resultados de cada equipo de la tarea junto a sus resultados.

En cuanto a la evaluación de la tarea, dado que es un problema de clasificación binaria, se decidió evaluar usando la "precision", el "recall" y el "F1 score".

Ranking	Team	CU	ES	MX	AVG
1	ELiRF-UPV	0.6527	**0.7167**	**0.6803**	**0.6832**
2	CIMAT	**0.6596**	0.6449	0.6709	0.6585
*	BASELINE-LDSE	0.6335	0.6795	0.6608	0.6579
3	JZaragoza	0.6163	0.6605	0.6703	0.6490
*	BASELINE-W2V	0.6033	0.6823	0.6271	0.6376
4	ATC	0.5941	0.6512	0.6454	0.6302
5	CICLiku	0.5621	0.6875	0.641	0.6302
6	LabGeoCi	0.6396	0.6251	0.6121	0.6256
*	BASELINE-word n-grams	0.5684	0.6696	0.6196	0.6192
7	SCoMoDI	0.6338	0.6652	0.5574	0.6188
8	LASTUS-UPF_method1	0.6017	0.6606	0.5933	0.6185
9	VRAIN	0.5204	0.6842	0.6476	0.6174
10	LASTUS-UPF_method2	0.5737	0.6493	0.6218	0.6149
11	Aspie96	0.5388	0.5935	0.5747	0.5690
12	UO_run2	0.5930	0.5445	0.5353	0.5576
13	UFPelRules	0.5620	0.5088	0.5464	0.5391
14	UO	0.4996	0.5110	0.4890	0.4999
	BASELINE-majority	0.4000	0.4000	0.4000	0.4000

Tabla 1: Tabla de los resultados de IroSvA 2019, con el AVG F1 score según cada variante, extraída de [5]

Antes de mencionar el trabajo de los ganadores, cabe referenciar que la universidad Pompeu Fabra con el equipo LaSTUS también participó [5]. Este equipo presentó 2 entregas distintas, que fueron creadas a partir de la arquitectura "simple bidirectional LSTM" [2].

El equipo CIMAT, obtuvo el mejor resultado en la variante cubana con un 0,6595 de average F1 score usando un modelo basado en explotar las características tradicionales y profundas (exploiting Traditional and Deep Features) [5][19]y el equipo de ELiRF-UPV que obtuvo los mejores resultados en las variantes española y mexicana con un 0,7167 y un 0,6803 de average F1 score, aplicando un modelo basado en codificadores por transformadores (Transformer encoders)[5][12] .

El equipo de CIMAT, se decidió por una estrategia que combina varias técnicas e hizo pruebas con diversas combinaciones para obtener más diversidad de resultados[6|21]. Ellos concatenaron tres componentes para construir su modelo.

El primero consiste en una BoW (Bag Of Words), que se centra en extraer y usar características basadas en una secuencia de palabras, aunque ellos se refieren al término como BoT (Bag of Terms) como la generalización que puede contener otro tipo de características como el n-grams. Para construir este espacio de características usan el "Term Frequency Inverse Document Frequency (TF-IDF)" [12].

En la segunda usan embeddings, en concreto embeddings pre entrenados de 300 dimensiones de Google [18] y de 400 dimensiones de Twitter [11]. La idea consiste en representar los documentos computando la media de vectores de palabras de cada documento [12].

Finalmente, la tercera se basa en usar el modelo de "Long Short-Term Memory (LSTM)" [15], normalmente usadas para capturar "long-term dependencies" con dos finales distintos. Uno usando "Support Vector Machines (SVM)" [6] para combinar con el LSTM y finalmente clasificar los datos y otro simplemente usando una función Sigmoid [12].

Ellos, trataron distintas combinaciones con las tres partes, pero la que mejor resultado obtuvo en la variante cubana (que fue la mejor de la tarea de IroSvA) fue usando una combinación del BoT tradicional, el embedding de Word2Vec de Google y finalmente las "hidden units" del LSTM. Los resultados generales de la tarea IroSvA se muestran en la Tabla 1 [12].

Aun siendo estos muy buenos resultados, he decidido centrar mi investigación en el modelo del equipo de ELiRF-UPV, ya que obtiene mejores resultados en la variante en español de España.

El equipo de ELiRF-UPV planteó una solución basada en codificadores por transformadores (Transformer encoders).

Para empezar, al igual que muchos de los participantes, usaron word embeddings [5]. En concreto usaron word embeddings de 300 dimensiones de un modelo skip-gram [18] entrenado con 87 millones de tweets usando Word2Vec [12].

En cuanto al modelo, ellos se basaron en una de las partes del modelo original de transformadores, en concreto usaron la parte de codificación para poder clasificar[12] en vez de traducir que era tarea inicial del modelo de transformadores[22]. En la *figura 2*, podemos observar la estructura final del modelo que presentaron al concurso.

Durante el testeo de este modelo, el equipo de ELiRF-UPV consiguió los mejores resultados en las variantes española y mexicana, comprobando así que este modelo se adecua con un alto rendimiento a la tarea de detección de ironía[5][12].

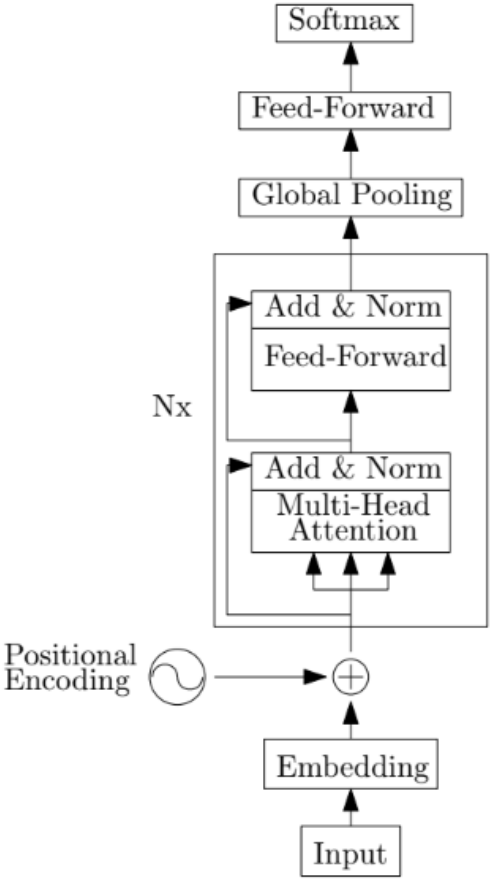


Figura 2; Representación del modelo usado por ELiRF-UPV en IroSvA 2019 extraída de [12]

2.2.2 SemEval 2018

Durante 2018, una de las tareas que SemEval lanzó (Tarea 3) fue dirigida a la detección de ironía en tuits en inglés. Esta se dividió en dos partes, la tarea A que consistió en determinar si un tuit era irónico o no, y la tarea B que consistió en clasificar los tuits según diferentes tipos de ironía, en concreto como: ironía por contradicción, ironía situacional, ironía verbal o sin ironía [14]. Me centro en la tarea A, dada la correlación con mi trabajo.

Los sistemas presentados por los participantes, fueron valorados según la “acuraccy”, la“precision”, el “recall” y el “F1-score”.

Además, la organización propuso un dataset de tuits en inglés para la tarea, que ellos mismos anotaron [14]. En total 43 equipos participaron en esta tarea, donde el equipo de UCDCC obtuvo el mejor resultado con un F1 score de 0,724 considerando los sistemas con restricciones [14]. En la tabla 2 podemos observar los resultados.

team	acc	precision	recall	F_1
UCDCC	**0.797**	**0.788**	0.669	**0.724**
THU_NGN	0.735	0.630	0.801	0.705
NTUA-SLP	0.732	0.654	0.691	0.672
WLV	0.643	0.532	**0.836**	0.650
NLPRL-IITBHU	0.661	0.551	0.788	0.648
NCL	0.702	0.609	0.691	0.648
RM@IT	0.691	0.598	0.679	0.636
#NonDicevo-SulSerio	0.666	0.562	0.717	0.630
DLUTNLP-1	0.628	0.520	0.797	0.629
ELiRF-UPV	0.611	0.506	0.833	0.629

Tabla 2: Resultados de los equipos participantes en la tarea A de SemEval 2018 Task 3, para los "constrained systems", extraída de [14]

Este equipo de manera similar al equipo que obtuvo mejores resultados en la variante cubana para la tarea de IroSvA 2019, hizo uso de las LSTM en combinación con embeddings [5][6|21][14], pero con el añadido de una red siamesa. Para ser más detallistas, crearon esta red siamesa donde cada una de las dos arquitecturas fue provista de diferentes partes del tuit, bajo la premisa de que un tuit irónico suele contener contradicciones o contrastes.

3. BACKGROUND TÉCNICO

En los siguientes sub apartados, trato de desarrollar algunos aspectos técnicos relacionados con el tema del trabajo y necesarios para entenderlo.

3.1 Modelos de aprendizaje automático

Los modelos de aprendizaje automático son modelos computacionales capaces de aprender patrones o características específicas de conjuntos de datos.

La mayoría de estos se basan en un entrenamiento, donde se procesan los datos y el sistema aprende estos patrones o características, y en una optimización de parámetros con el fin de acabar de ajustar el algoritmo con el conjunto de datos.

Existen varios tipos de modelos de aprendizaje automático, entre algunos de los más populares encontramos:

- Aprendizaje supervisado: En este caso, los modelos se basan en que estos tengan datos de entrada además de datos de salida. De esta forma el modelo aprende según si su predicción es correcta o no.

- Aprendizaje no supervisado: Este enfoque se basa en dar al modelo datos de entrada pero no de salida, de esta manera el modelo aprende los patrones de los datos sin esta restricción.

- Aprendizaje por refuerzo: Esta variante de modelo, se enfoca en que el modelo aprenda a través del entorno a según una serie de recompensas o penalizaciones, para así intentar que el modelo trate de tomar siempre las decisiones más óptimas para el problema.

- Aprendizaje profundo (DL): Este tipo de sistemas utiliza el concepto de las redes neuronales artificiales, compuestos por, normalmente, múltiples capas de neuronas, donde cada neurona intenta aprender características sobre un conjunto de datos. A lo largo de los años, se ha ido demostrando cómo este tipo de modelos de aprendizaje, son muy eficientes en algunos campos de investigación, entre ellos el PLN.

3.2 Redes de transformadores (Transformers)

Entre los muchos modelos de DL que existen, las redes de transformadores han demostrado tener un gran impacto en aplicaciones para el PLN entre otras.

Este modelo, publicado en 2017 se creó como una alternativa a los problemas que tenían las redes para la tarea de traducción automática, mostrando un modelo paralelizable y que

obtiene mejores resultados en esta tarea que las complejas redes neuronales recurrentes o convolucionales [22].

Sobre la estructura de del modelo, primero se procesa la entrada como embeddings para obtener una representación de vectores y luego se hace uso de un codificador posicional que con la ayuda de funciones sinusoidales, consigue almacenar de manera única la posición de cada token [22].

Seguidamente, empieza la parte de codificación, que está compuesta por 6 codificadores que son idénticos en cuanto a estructura. Cada uno de ellos está compuesto por dos componentes, un "multi-head self-attention mechanism"y un "position wise fully connected feed-forward network". El primer componente lleva la entrada a 3 redes neuronales, QUERIES, KEYS y VALUES, que se usan para entender las relaciones que existen a diferentes niveles de la secuencia, a través de la siguiente fórmula extraída del paper original [22].

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Seguidamente por cada una de las proyecciones de QUERIES, KEYS y VALUES repiten el proceso y concatenan todo en un solo output [22].

El segundo componente es una red que trata de procesar y transformar características o representaciones de manera independiente para cada posición en la secuencia [22]. Además, estos componentes reciben como entrada tanto la salida como la entrada del anterior componente evitando, de esta manera, problemas con pérdida de información [22].

Una vez todo codificado se pasa a la parte de decodificación con 6 decodificadores, también constituidos por 3 componentes. Dos de ellos prácticamente idénticos a los de los codificadores pero con un tercero adicional que también es una "multi head self attention layer" pero con la diferencia de que también obtiene datos de la salida del codificador. Además a la primera "multi head self attention layer" se le añade una máscara a tokens [22].

Finalmente, con una capa extra y la ayuda de la función de activación Softmax [4] el modelo puede generar la traducción. En la *figura 3* podemos ver la estructura del modelo.

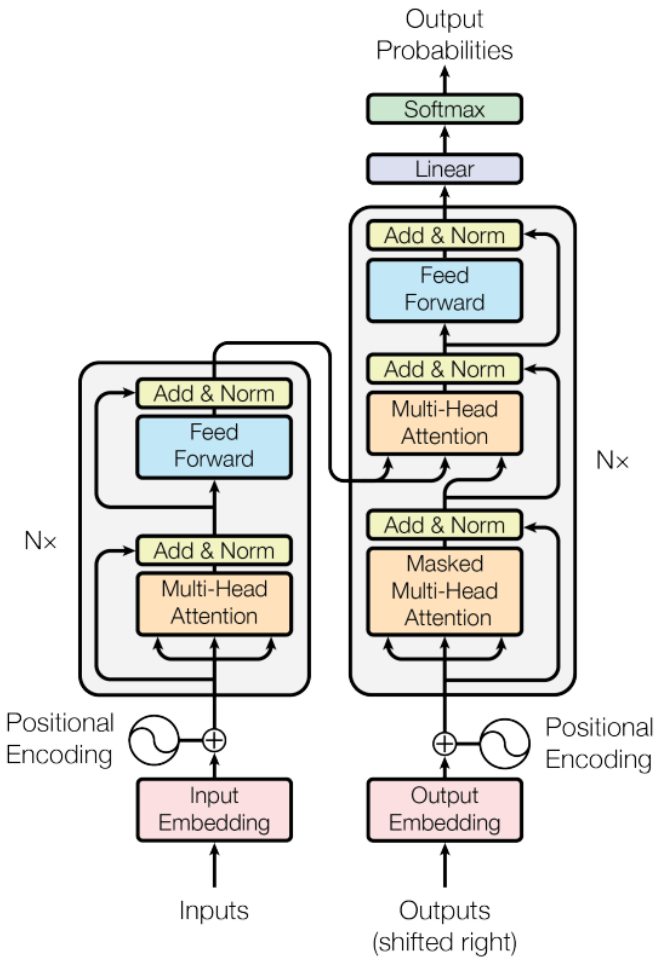


Figura 3: Estructura básica de una red transformador, extraída de [22]

3.3 Ajuste fino

El ajuste fino, o mejor conocido como “fine-tunning” en inglés, es una técnica para el aprendizaje automático que se basa en, a partir de un modelo ya entrenado para una tarea concreta, hacer unas pequeñas modificaciones y un entrenamiento para que ese modelo se refine en una tarea similar.

Pongamos que tenemos un modelo que dada una imagen, detecta si es o no un balón de fútbol, y queremos un nuevo modelo para detectar si una imagen es un balón de baloncesto o no lo es. Además, nuestro dataset no es demasiado grande para la tarea, y sabemos que el modelo que detecta balones de fútbol tiene una muy buena base. Dada la similitud entre las dos tareas, podemos reentrenar parte del modelo de balones de fútbol para que este sea capaz de detectar balones de baloncesto.

Para ello se suelen congelar capas del modelo, lo que en resumidas cuentas significa impedir que esas capas actualicen sus pesos, i . e., no sean modificados y por lo tanto, no aprendan [10]. Es decir, la data entra pasando por todo el modelo hasta el final “forwardpropagation”, luego la pérdida se calcula y comienza la “backpropagation”

donde normalmente se actualizan los pesos según las predicciones, pero en la mayoría de capas, no se actualizan los pesos.

En concreto se suelen congelar las capas iniciales que son las que aprenden patrones y/o características más generales de la imagen, como podría ser la redondez del balón, pero descongelando alguna capa final ,que suelen ser las que aprenden patrones más concretos y detallados, para que se puedan refinar las características específicas más relacionadas con las pelotas de baloncesto. Además se suele añadir una capa final extra con tal de refinar la clasificación. Una vez hecho esto, obtendremos un modelo capaz de detectar balones de baloncesto sin tener que entrenar un modelo desde cero, evitando problemas de optimización, de falta de datos o de consumo excesivo de energía y tiempo.

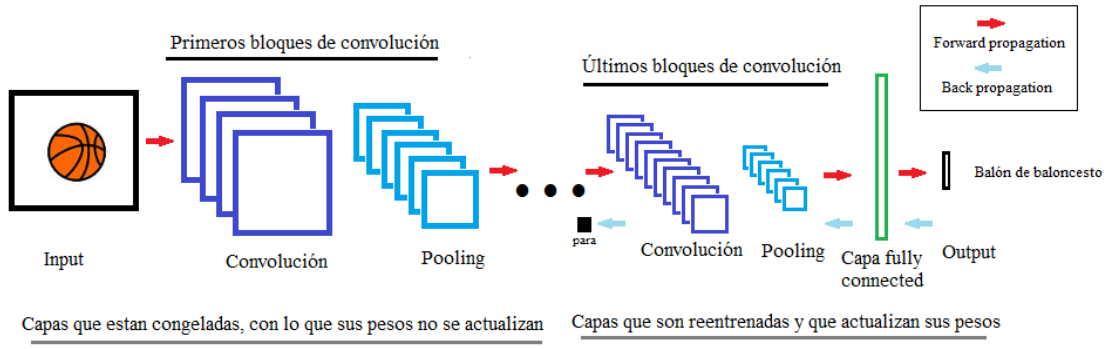


Figura 4: Ilustración del concepto de "fine-tuning" con ejemplo de un modelo de detección de balones de baloncesto usando una red convolucional simple.

3.4 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

"Bidirectional Encoder Representations from Transformers", BERT es un modelo de PLN publicado por Google en 2018 que consiguió hacer avanzar considerablemente el estado del arte en varias tareas del PLN [8].

Este modelo, a diferencia de sus predecesores, no fue desarrollado para una tarea concreta como podría ser la clasificación de texto, la traducción de texto o el análisis de sentimiento, sino que fue desarrollado para poder interpretar el lenguaje natural en general, para seguidamente, poder añadirle un extra como, podría ser una pequeña red neuronal, y finalmente afinar el modelo en una tarea en particular [8]. Es decir, la idea de BERT es que sea reentrenado para tareas como podrían ser la equivalencia semántica o la aceptabilidad lingüística, entre otras, pero sin la necesidad de usar mucha data y con una muy buena base de modelo [8].

BERT, fue entrenado con el BooksCorpus [25] y la wikipedia en Inglés compuestos por 800 y 2500 millones de palabras respectivamente [8]. Aun así se debe tener en cuenta que varias versiones de Bert fueron lanzadas, variando en el idioma o en el tamaño de estos, como podrían ser Bert-Base, Bert-Large o Bert-Multilingual [8].

La arquitectura que usa BERT, está basada en las redes de transformadores, de manera específica, BERT es un modelo que solo usa prácticamente la parte de codificación de un

transformador [8], es decir, que se deshace totalmente de la parte de decodificación, y se queda con la parte de codificación.

En cuanto a la entrada de datos de BERT, estos se separan siempre en dos frases a las que se le añade un token de separación entre ellas [SEP] , y un token de clasificación al principio [CLS] [8]. Seguidamente por cada token se suman tres representaciones, una usando WordPiece[24] embeddings para representar el contenido de estos, otra con embeddings posicionales representando la información posicional de cada token y finalmente otro embedding indicando a qué segmento de la frase pertenece cada token [8]
.

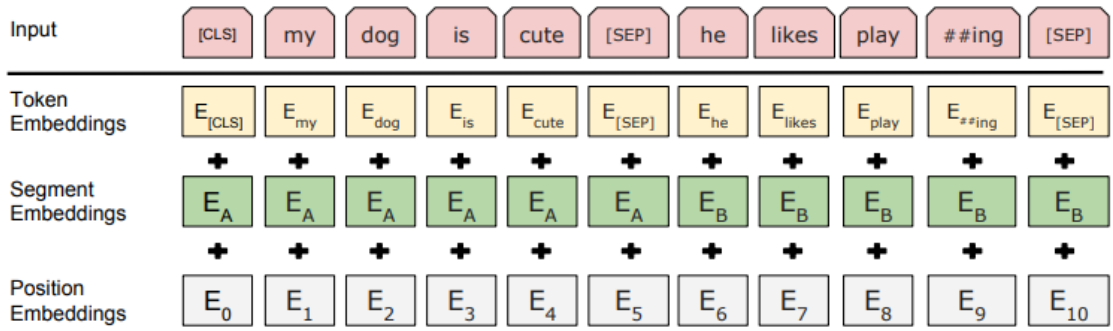


Figura 5: Proceso de entrada de datos del modelo BERT, extraída de [8]

Para el entrenamiento de BERT, se usan dos tareas distintas.

La primera consiste en enmascarar de manera aleatoria tokens en una frase, y entrenar al modelo para predecir el token que falta , se usa esto para no tener un modelo que entienda la data de derecha a izquierda o de izquierda a derecha, sino que sea bidireccional y tenga un mayor entendimiento de los inputs[8], lo cual, les resultó de manera correcta.

La segunda consistió en entrenar al modelo para predecir si dos frases son consecutivas o no. De esta manera “ el modelo es capaz de entender la relación entre dos frases, que no es captado directamente por el modelo lingüístico [8]”.

De esta manera, el modelo ya puede ser “fine-tuned” para otro tipo de tarea, de manera menos costosa en cuanto a tiempo y data [8].

Los resultados confirman el rendimiento del modelo, obteniendo este, el nuevo estado del arte para once diferentes tareas para el PLN [8], entre ellas, el análisis de sentimientos que guarda relación con la detección de ironía.

4 DATASET

En esta sección se explican y analizan los dos datasets usados para este trabajo. El primero, siendo parte del dataset de la tarea IroSvA 2019, y el segundo concebido durante este proyecto. El nuevo dataset ha sido publicado, para poder hacer uso de este en https://github.com/LaSTUS-TALN-UPF/IroMovies .

4.1 IroSvA dataset

Durante la tarea de IroSvA, varios datasets fueron proporcionados por los organizadores, uno en la variante española, otro en la variante cubana y otro en la variante mexicana.

En concreto los datasets constan de una serie de mensajes cortos (tweets y comentarios de noticias) anotados como irónicos o no, además de un pequeño contexto, como podría ser el hashtag de un tweet (#PañalesIglesias). Cada dataset fue anotado por anotadores nativos de cada país, y cada uno se rigió por su propio concepto de la ironía. Además, indicaciones fueron dadas de no diferenciar entre tipos de ironías y de incluir el sarcasmo como un caso especial de ironía [5].

En este trabajo, he decidido centrarme en ironía en la variante en español, con lo que los datasets de las variantes mexicanas y cubanas no han sido usadas para experimentar o entrenar el modelo.

Este dataset consta de 3000 frases, 1000 irónicas y 2000 no irónicas, fueron anotadas por dos anotadores, y se decidieron como irónicas solo las frases donde ambos anotaron como irónicas [5].

Para consolidar la base del modelo, usamos este único dataset repartiendo el 80% para el proceso de entrenamiento, el 20% para el proceso de testeo y el 20% del dataset de testeo para validación. De esta manera, los resultados pueden ser comparados en cierta medida con los conseguidos en la tarea anterior, ya que los organizadores especifican el uso de las mismas particiones de dataset a excepción del conjunto de validación que no se especifica.

Para añadir contexto, como ejemplo de frase irónica del dataset encontramos:”*Si llega tarde al juicio... ¿Joan tarda?*” donde podemos ver el estilo de frases que encontramos en este conjunto.

4.2 Movie reviews dataset

Nuestro objetivo es crear un modelo de detección automática de ironía en español, con lo que puede resultar interesante testear nuestro modelo con datos provenientes de otro dataset para verificar la robustez del modelo al cambio de tipo de texto y dominio. Es por

esto, y por varias razones que hemos decidido crear un nuevo dataset, con un origen de datos distinto al usado en el entrenamiento (IroSvA 2019).

A continuación relato todo el proceso de creación de este dataset.

Para empezar, encontramos un dataset de reseñas de películas que fue usado para una tarea de clasificación de opiniones [17] , de las cuales obtenemos y guardamos en archivos XML para mayor manipulabilidad. Entre otra información como es el autor o el título de las reseñas, los datos originales también aportan una clasificación para cada reseña. Esta clasificación refleja que tan positiva o negativa es la reseña, usando 5 clases donde 0 significa que al autor le ha desagradado mucho la película y donde 5 significa que al autor le ha gustado mucho la película.

Una vez obtenidos los datos, se ha pasado por un preproceso y selección de reviews. Partiendo del contenido original se han descartado todas las reviews con extensión muy larga o muy corta, todas las reviews con una media de longitud de frases muy larga o muy corta, y escogiendo un número uniforme de reseñas según su clase, es decir, tratando de obtener un dataset centrado en la media.

Aún así, se debe tener en cuenta que los datos originales son difíciles de preprocesar. Esto es debido a que es complicado extraer las frases de un texto que se ha escrito con poco cuidado, con lo que una revisión manual de todas las reseñas elegidas habría ayudado a mejorar la calidad del dataset.

El siguiente ejemplo muestra un posible problema : "*Esta película es como contar ovejitas. es decir. el mayor muermo que he visto en mi vida, sobre todo cuando huyen de U.S.A en ese intento de avión*". En este caso podemos deducir que ".es decir." debería estar entre "," , y que no se trata del final de una frase, al igual que pasa con el término "U.S.A", que son unas siglas. Estos pequeños detalles, sin revisión manual o una IA de gran calidad, son difíciles de detectar a través de código convencional. Aun así las reseñas elegidas muestran escasos ejemplos como el anterior.

Una vez preprocesadas las reseñas, se han escogido 50 repartidas en 3 batches de manera aleatoria, aun manteniendo la uniformidad de clases, para generar las frases del dataset. De esta manera dos anotadores expertos en lenguaje figurativo y con experiencia en el campo de las anotaciones, pudieron etiquetar.

Para evaluar la calidad de las anotaciones, se ha decidido usar el coeficiente kappa de cohen, propuesto por Jacob Cohen en 1960 y comúnmente aceptado como medida de calidad de la concordancia entre anotaciones por ayudar a discernir si el acuerdo entre los anotadores va más allá de lo que se esperaría solo por casualidad o aleatoriedad [7]. Los resultados se muestran en la *tabla 2*.

Batches	Coeficiente KAPPA de COHEN
Batch 1	0.1459
Batch 2	0.3121
Batch 3	0.0255

Tabla 2, coeficiente KAPPA resultante de cada batch

Dada la poca compatibilidad entre anotaciones, se decidió añadir un tercer anotador a las reseñas elegidas. De esta manera hemos decidido tomar como irónicas las frases donde al menos 2 anotadores así lo han decidido. Aun así, cabe destacar que se pueden tomar varias variantes de este datasets, como podría ser tomar como irónicas todas las frases donde al menos un anotador tomó como irónica.

Cabe resaltar que aun siendo el coeficiente KAPPA de COHEN tan bajo, esto solo muestra el grado de concordancia, no muestra la calidad de las anotaciones.

Una vez todo anotado, hemos vuelto a reunir todas las reseñas en el formato inicial (XML) con las anotaciones y las frases separadas, de esta manera el dataset ya está consolidado. Hemos guardado este dataset en un github https://github.com/LaSTUS-TALN-UPF/IroMovies , para uso open source, y hemos añadido un código en python que procesa este dataset en el tipo dataframe de la librería de python PANDAS. A futuro, sería interesante actualizar el código para que sea fácil de exportar en matlab.

Finalmente, la distribución de este dataset se resume en un total 883 instancias de las cuales 63 son irónicas y 820 no son irónicas, si tomamos como frases irónicas las que dos o más anotadores decidieron como irónicas. Esto implica que puede ser buena idea hacer redistribuir el tamaño del dataset al momento de testearlo.

Para añadir contexto, como ejemplo de frase irónica del dataset encontramos: "*Tras dos secuelas que explotaban el mito de la máquina de matar humana, aunque entonces contra los peores enemigos de los USA, la Unión Soviética se derrumbó, el comunismo desapareció como gran amenaza y Rambo se quedó sin enemigos a los que masacrar*." y donde a simple vista podemos ver lo diferentes que son las frases de ambos datasets.

4.3 Comparativa entre datasets

En este apartado recalco las diferencias entre datasets, con el fin de mostrar la variabilidad de data entre estos.

Los datos originales del dataset de IroSvA provienen de twitter lo que significa que cada oración (dadas las limitaciones de longitud de tweets en 2019) no puede tener más de 280 caracteres, que podemos estimar o suponer como unas 15 palabras. Incluso así, hay que tener en cuenta que la naturaleza de los tweets es más bien de mensajes cortos, tal y como se comenta en [5]. Por otra parte las reseñas elegidas para el segundo dataset tienen una media de 44 palabras (calculada durante la creación de los batches). Esto resulta en que

de media, una frase del segundo dataset tiene aproximadamente 3 veces más palabras que una frase del primer dataset.

Además, considerando el dominio de la data original de cada dataset podemos ver que disciernen mucho. Twitter es una red social con un gran alcance y variabilidad en cuanto al usuario estándar, por otro lado, la página de reseñas de películas tiene un menor alcance y una menor variabilidad en cuanto a su público, el cual podríamos considerar como "de nicho". El simple factor del público general o el contexto de cada origen de datos influye en el resultado final.

También cabe destacar que los encargados de la anotación de ambos datasets tienen conceptos de ironía distintos, de hecho, el en dataset de IroSvA se especifica que los anotadores usan sus propios conceptos de ironía para la anotación [5].

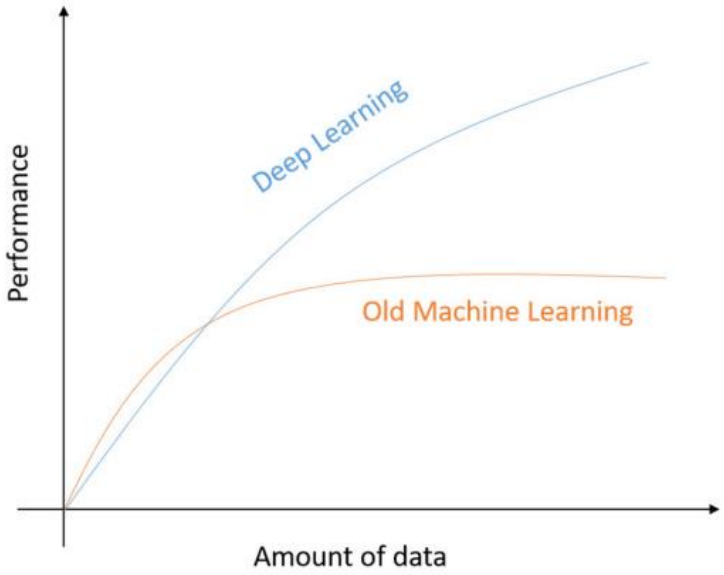
En general estas diferencias darán más riqueza a los experimentos al poder comparar dos ámbitos muy distintos.

5 ARQUITECTURA DEL MODELO

En este apartado se explica la arquitectura usada para la creación y construcción del modelo de detección automática de ironía.

Una de las muchas diferencias que existen entre los modelos de “machine learning” (ML)y DL es la relación que existe entre la cantidad de datos con el rendimiento obtenido [1].

Para ser exactos, en la *gráfica 1*, podemos observar un umbral que indica que según la cantidad de datos usados en el entrenamiento del modelo, un candidato de modelo debería obtener mejores o peores resultados.



Gráfica 1: Mostrando la cantidad de datos relacionados con el rendimiento en DL y ML, extraída de [1]

Pero además podemos observar que en los modelos de DL, llegados a un punto, la pendiente que relaciona los datos con el rendimiento se hace más y más prolongada, incluso recordando a una función logarítmica.

Es por ello que la cantidad de datos que un modelo de DL requiere, ha de ser muy presente en el momento de decidir entre los tipos de modelos.

Aun con todo esto, como se ha explicado anteriormente en el apartado de estado del arte, podemos observar que algunos modelos de DL acaban superando con creces a los modelos de ML por lo general. Además teniendo en cuenta que el dataset que se usa para el entrenamiento da mejores resultados con modelos de DL [5] como vemos en los resultados de la tarea IroSvA 2019, la decisión final ha sido usar un modelo de DL.

En concreto, dados los buenos resultados que los modelos basados en redes de transformadores obtienen [22][12], y la salida del modelo BERT que se basa en esta arquitectura, y marcó un antes y un después en el campo del PLN, la decisión final ha sido partir del modelo de BERT.

5.1 BERT para la detección de ironía

BERT es un modelo muy completo que pudo mejorar la "baseline" en varias tareas del PLN [8]. Entre las tareas mencionadas está el análisis de sentimientos que podríamos decir que es de las que más relación tienen con la detección de ironía. Añadido al propio hecho de que BERT fue concebido con la idea de ser pre-entrenado para tareas más específicas del PLN, encontramos y sabemos que la Universidad de Valencia, que participó en IroSvA 2019 como el equipo de ELiRF-UPV siguió investigando el tema y presentó una metodología muy parecida para un nuevo modelo de detección de ironía [13].

Como sabemos, hay varias versiones de BERT, entre ellas BERT-Base o BERT-Large por ejemplo. Dado que trato de crear un modelo de detección de ironía en español, usó la versión de BERT-Multilingual cased.

En concreto BERT, usa la misma estructura de modelo que se propuso en el paper de redes de Transformadores [8][22], pero solo teniendo en cuenta la parte de codificación.

Por lo tanto, podríamos decir que la arquitectura de mi modelo es idéntica a la de BERT, pero con un añadido para poder clasificar. Para ser exactos he decidido añadir layers al final, como se recomienda para el uso de BERT[8].

Específicamente he añadido una capa de dropout 0,1 que me ayuda a regularizar la y prevenir el overfitting. Para ser exactos, esta capa se encarga de "apagar" el 10% de las neuronas de la capa anterior lo que significa que estas neuronas "apagadas" no se tienen en cuenta para esa época o "epoch". También he añadido capa totalmente conectada o "fully connected layer" para así ayudar a perfeccionar el modelo y finalmente una capa Softmax para poder clasificar los resultados.

La idea de las dos últimas capas extra, ha venido de un tutorial de la biblioteca de Tensorflow que explica cómo usar BERT [16].

Considerando que estamos tratando de obtener un modelo de clasificación binaria, usamos la función de pérdida "BinaryCrossEntropy" que está precisamente creada para este tipo de tareas, y como optimizador, tras algunas pruebas he decidido usar Adam.

Finalmente, hay casos en que durante el entrenamiento, el modelo empieza a aprender demasiado del conjunto de entrenamiento y la pérdida del conjunto de validación empieza a aumentar. Es por ello que se usa el concepto de parada anticipada, que consiste en parar el entrenamiento cuando este caso se da durante varias épocas seguidas.

5.2 Ajuste fino para BERT

En nuestro caso, como se ha explicado anteriormente, el modelo de BERT fue precisamente creado para ser reentrenado y afinarlo para otras tareas relacionadas con el PNL [8], además el equipo de ELiRF-UPV mostró una resolución similar, obteniendo de los mejores resultados en la tarea de IroSvA [5][12]. Es por ello que he decidido reutilizar

el mismo concepto y hacer “fine-tuning” al modelo de BERT para la tarea de clasificación de ironía.

En cuanto al número de capas congeladas, los autores de BERT recomiendan congelar capas según el tamaño del conjunto de datos, y aun siendo este un conjunto de datos no demasiado grande, los mejores resultados se obtienen a partir de reentrenar todas las capas, con lo que se ha decidido no congelar ninguna capa.

6. METODOLOGÍA

En este apartado se relata la metodología seguida en cada uno de los aspectos del trabajo.

6.1 Preprocesamiento de datos

En cuanto al preprocesamiento de los datos, la metodología se dividirá en dos partes. La primera conformándose con preprocesamiento que considero necesario para el los datos, y la segunda, con el preprocesamiento de los datos que necesita BERT para poder ser entrenado.

Sobre el primer preproceso, nos fijamos en los datos de cada dataset. El dataset de IroSvA, es un dataset compuesto por tweets [5], esto hace que contenga tipo de data que puede resultar en ruido para el modelo, como son las menciones a otros usuarios, los hashtags o una URL, es por ello que se sustituyen éstos por sus correspondientes tags *usuario*, *hashtag* y *URL* , reproduciendo una parte del preproceso que proponen los ganadores de la variante española y mexicana de IroSvA [12]. Sobre el segundo dataset, varios tipos de preproceso serán usados para poder hacer comparaciones.

El segundo preproceso es proporcionado por BERT y ya se encarga del resto. Cada modelo BERT tiene un tipo de preprocesamiento único. En nuestro caso primeramente hay dos tipos, el "cased" y el "uncased", y nosotros usaremos el "cased" que mantiene mayúsculas, minúsculas y tildes.

Primero se procesan las frases como tokens usando la tokenización de WordPice. Seguidamente se añaden tokens especiales necesarios según el caso, los posibles tokens son:

[CLS]: Token para indicar el inicio de una frase.

[SEP]: Token para indicar separaciones entre oraciones y el final de una secuencia.

[PAD]: Token para rellenar e igualar la longitud de las secuencias.

Finalmente se mapean los tokens como ids numéricos únicos, y se crea una mascara de atención que indica que tokens forman parte de la entrada real y cuales son de relleno o especiales.

6.2 Herramientas del entrenamiento

Python, es un lenguaje de programación comúnmente usado en implementaciones de ML y DL debido a la gran cantidad de librerías para estas tareas que dispone. Matlab, también es un excelente lenguaje de programación para este tipo de tareas, pero se ha decidido usar Python dada mi experiencia en el lenguaje.

En cuanto al entrenamiento en sí, he decidido usar Google Colaboratoy, una herramienta que entre otras, permite programar en Python sin tener que instalar dependencias o librerías y que ofrece acceso a una GPU T4 que es muy útil para este tipo de procesos.

Aun así cabe recalcar que esta herramienta tiene utilización limitada, y que su frecuente uso puede llegar a bloquear el uso de la GPT T4 temporalmente.

6.3 Métricas de evaluación

En cuanto a las métricas y la evaluación del modelo, he decidido seguir las formas usadas típicamente en tareas de clasificación binaria y también usadas en trabajos anteriores [5][14][8]. En concreto, para evaluar el modelo respecto a los datos de testeo, computo la "acuraccy", la "precision" , el "recall" y el "F1-score" siguiendo las siguientes fórmulas:

$$\text{Accuracy} = \frac{\text{Número de correctamente clasificados en ambas clases}}{\text{Número total de instancias de ambas clases}}$$

$$\underset{\text{Por clase}}{\text{Precision}} = \frac{\text{Número de correctamente clasificados por clase}}{\text{Número total de clasifificados por clase}}$$

$$\underset{\text{Por clase}}{\text{Recall}} = \frac{\text{Número de correctamente clasificados por clase}}{\text{Número total de instancias por clase}}$$

$$\underset{\text{Por clase}}{\text{F1}} = 2 \times \frac{\underset{\text{Por clase}}{\text{Precisión}} \times \underset{\text{Por clase}}{\text{Recall}}}{\underset{\text{Por clase}}{\text{Precisión}} + \underset{\text{Por clase}}{\text{Recall}}}$$

Estas fórmulas me ayudarán a evaluar el rendimiento del modelo y me darán la oportunidad de comparar con trabajos anteriores que hayan usado las mismas ecuaciones (No en todos los casos).

Varios de mis experimentos se enfocan en el uso de nuevos y distintos datos a los del dataset base. Para poder entender varios conceptos sobre esta nueva data, la uso como testeo a partir del modelo con mejor rendimiento. Esto, me ayudará a entender varios aspectos como son:

- La capacidad del modelo para adaptarse al nuevo conjunto de datos.
- El parecido entre la ironía del dataset de entrenamiento y la ironía de los nuevos datos.
- La calidad de los nuevos datos en términos de ironía.

Para la evaluación del modelo durante el entrenamiento uso un pequeño dataset, conformado del 10% de el dataset de testeo con el cual puedo ver la eficacia del modelo con respecto a datos que no ha visto aún, esta tipo de división de dataset es conocida como "validation split".

7 EXPERIMENTOS

Durante este apartado sigo la metodología propuesta además de varios apartados anteriormente mencionados, para finalmente generar el modelo final. Todo el código que he usado para este apartado y los varios modelos que he obtenido han sido almacenados en github para optimizar el espacio https://github.com/erico089/Irony-Detection-Model-TFG .

7.1 Optimización de hiperparámetros

Los hiperparámetros son una serie de valores indicados antes de realizar el proceso de entrenamiento del modelo que a diferencia de los parámetros del modelo, estos no se aprenden a través de los datos sino que son impuestos por el desarrollador. Según la arquitectura del modelo, unos hiperparametros son impuestos. En este caso, para poder entrenar el modelo necesitamos configurar valores para:

Paciencia de la parada anticipada: Dado que el modelo usa una para anticipada para que este no gaste más tiempo del necesario y el resultado no empeore, se debe decidir la paciencia, que es el número de iteraciones del entrenamiento en el que el modelo no mejora el error de validación. Si el modelo está más iteraciones que las indicadas sin mejorar, este para de iterar.

Épocas: Durante el entrenamiento, una pasada completa por todos los datos de entrenamiento es lo que consideramos como una época, también conocido como iteraciones o “epochs”.

Pasos por época: Número de iteraciones que ocurren durante una época. Este valor está directamente relacionado con el tamaño de la entrada de datos, y del tamaño de lote.

Número de pasos de calentamiento: Número de pasos adicionales al principio del entrenamiento que se usan para acabar de calibrar la tasa de aprendizaje.

Tasa de aprendizaje inicial: La tasa de aprendizaje controla la proporción de la actualización de los pesos durante el entrenamiento. Esta es solo la inicial ya que con el parámetro de número de pasos de calentamiento, acabamos de refinarla.

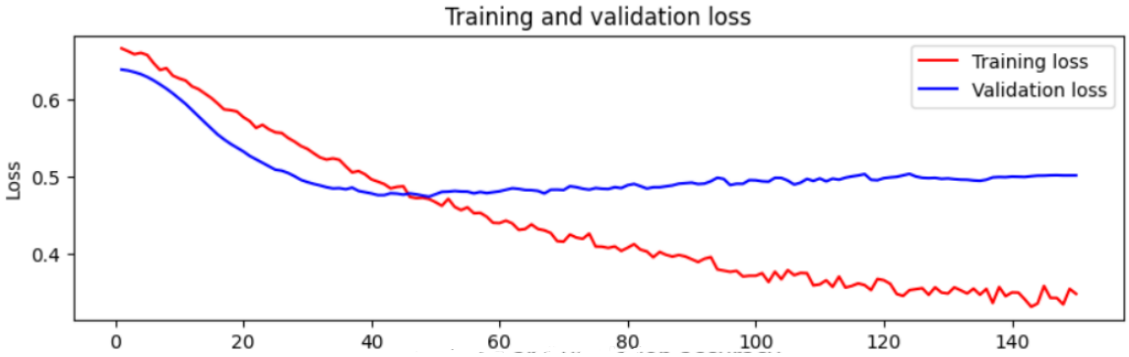
Tamaño de lote: Más conocido como “batch size”, es la cantidad de datos que se usan en cada paso de una época.

Durante las pruebas de entrenamiento, siguiendo las métricas de evaluación, he probado varias combinaciones de hiperparametros. Tras varias pruebas, he recolectado las más significativas y he seleccionado como modelo base, el que mejor “F1-score” ha obtenido. En la *tabla 3* se muestran algunas de las diferentes pruebas que se han hecho, junto a los resultados. Cabe destacar que los hiperparametros: Número de pasos de calentamiento, Pasos por época y tamaño de lote siempre son los mismos en este caso.

Versión del modelo	Épocas	Tasa de aprendizaje inicial	Paciencia	Accuracy	Precision	Recall	F1-Score
V1	100	3e-08	no	0,658	0	0,658	0
V2	150	3e-07	no	0,75	0,661	0,785	0,718
V3	60	3e-06	no	0,745	0,641	0,79	0,708
V4	60	3e-07 + 3e-07/2	no	0,708	0,574	0,776	0,66
V5	62	3e-07	15	0.682	0.531	0.772	0.629
V6	98	3e-07	60	0,68	0.526	0.783	0.629

Tabla 3: Resultados de varias combinaciones de hiperparametros

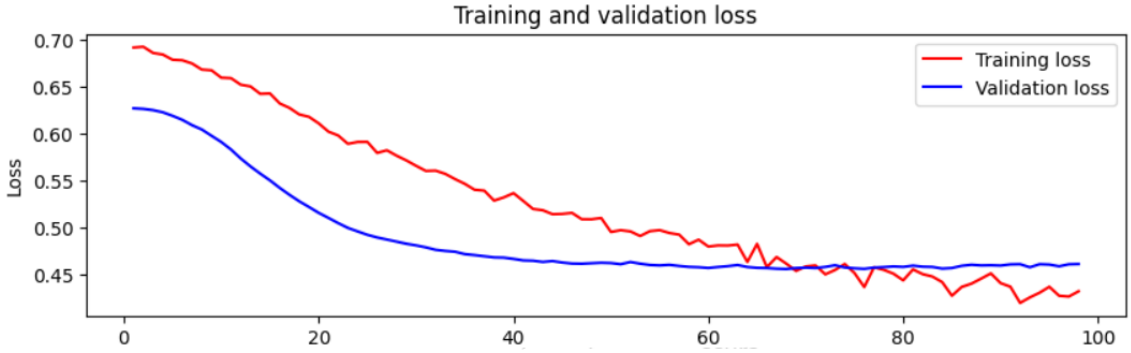
Es interesante también ver la *gráfica 2*, que muestra la pérdida de validación y la pérdida de entrenamiento durante el entrenamiento del modelo con mejores resultados. En esta podemos ver cómo llegado a un punto la pérdida de validación deja de bajar incluso a subir, en cambio la pérdida del entrenamiento sigue bajando, lo que significa que el modelo no generaliza bien, lo que también se conoce como "overfitting" o sobreajuste, aun así, los resultados son mucho más acertados que cuando usamos la parada anticipada.



Gráfica 2: Valores de pérdida por número de épocas durante el entrenamiento del modelo V2

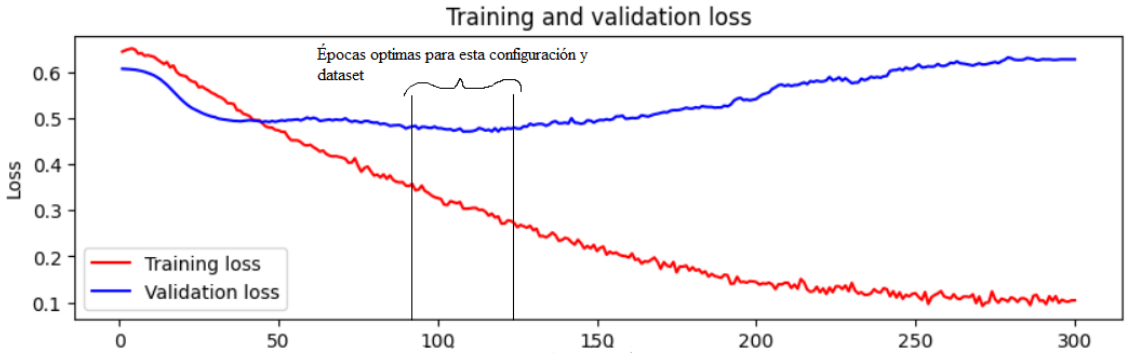
Como comparación, a esta configuración de modelo le añadimos una parada anticipada de paciencia 15 y 60, donde la diferencia entre las pérdidas existe pero aún no es tan notoria y la pérdida de validación no incrementa demasiado. Aun con esto, los resultados en cuanto a F1-score y accuracy, son notablemente mejores en el modelo base.

Gráfica 3: Valores de pérdida por número de épocas durante el entrenamiento del modelo V6 (paciencia 15)



Gráfica 4: Valores de pérdida por número de épocas durante el entrenamiento del modelo V5 (paciencia 60)

Como añadido, también he entrenado el modelo con esta configuración pero con 300 épocas y sin parada anticipada. Esto nos permite ver la zona óptima donde el modelo llega a su mejor momento, ya que llegados a un punto, la pérdida de validación empieza a crecer y la pérdida de entrenamiento empieza a disminuir considerablemente lo que implica que el modelo será totalmente incapaz de clasificar casos nuevos.



Gráfica 5: Valores de pérdida por número de épocas durante el entrenamiento, donde la zona optima esta marcada.

Podemos observar que la zona óptima presenta unos valores de pérdida muy altos, entre el 0.5 y 0.3. Esto se puede deber a varias razones, entre ellas, por falta de datos y/o por la complejidad del problema. Aún así, los resultados no varían mucho respecto a algunos de los resultados obtenidos por los participantes de IroSvA, con lo que aun obteniendo valores de pérdida muy altos, voy a centrarme en los métodos de evaluación especificados en el apartado de metodología, que sí son comparables.

7.2 Data augmentation

Tal y como se ha explicado anteriormente, y como se puede deducir, los modelos de aprendizaje automático aprenden de los conjuntos de datos, y no es ninguna sorpresa entender que para muchas tareas, como es esta, no se dispone de una cantidad enorme de datos anotados. Es por ello, que el concepto de "data augmentation" DA gana reconocimiento. La DA es una técnica, que consiste en a partir de los datos originales o de datos ya existentes, aplicar pequeñas modificaciones o transformaciones para generar nuevos datos. Estos nuevos datos no aportan la misma riqueza que los datos originales, pero sí que pueden ayudar a mejorar el rendimiento del modelo [9].
Para tareas de PLN, varias transformaciones y/o modificaciones son propuestas [9], entre algunas de ellas, he decidido probar la DA a través de modelos generativos de texto y la DA a través de "back translation".

7.2.1 Data augmentation a través de modelos generativos de texto

Recientemente, la empresa OpenAI ha lanzado de manera pública un modelo de generación de texto llamado "chat GPT (Generative Pre-trained Transformer)", que prueba ser un modelo de generación de texto con muy buen rendimiento[20]. Dada esta herramienta pública, y que he podido experimentar con esta durante un tiempo, he decidido usarla como técnica de DA de frases ironicas. En concreto, he decidido generar dos pequeños conjuntos de datos a partir de este modelo.

El primero consiste en frases irónicas generadas por el mismo modelo, sin ningún tipo de influencia por parte de los prompts que piden estos datos nuevos. El segundo, por otra parte, está compuesto por datos que sí han sido influenciados por los prompts. En concreto, estos prompts contienen datos originales del dataset de entrenamiento con los que focalizamos mucho el tipo de respuesta que genera el modelo.

Generación de texto sin influencia: En total he generado 200 frases irónicas y 200 frases no irónicas nuevas. Para ello he usado los siguientes prompts, repitiendo el segundo prompt de cada variante (irónica o no) hasta 9 veces. A los modelos entrenados de esta manera nos referiremos como DAGPT1.

Genera 20 frases irónicas. Cada frase debe estar separada por un salto de línea.

Genera 20 frases diferentes más siguiendo el mismo formato.

Genera 20 frases no irónicas. Cada frase debe estar separada por un salto de línea.

Genera 20 frases diferentes más siguiendo el mismo formato.

figura 6: Prompts para generar datos irónicos

Generación de texto con influencia: En este caso he generado 1000 frases ironicas y 1000 frases no ironicas. Dado que el dataset de entrenamiento contiene 10 tópicos distintos, he generado 20 frases por cada tópico. En el prompt, he añadido un pequeño contexto del tópico de las frases además de 5 ejemplos escogidos de manera aleatoria. Seguidamente

se muestra la plantilla de ejemplo del prompt en la *figura 7*.A los modelos entrenados de esta manera nos referiremos como DAGPT2.

Dado el tema de siguientes 5 frases, quiero que me generes 10 nuevas frases IRÓNICAS

Antes te proporciono algo de contexto del tema:

CONTEXTO:

<contexto>

FRASES DE EJEMPLO:

<5 frases ironicas aleatorias>

figura 7: Plantilla de prompt para generar ironía

7.2.2 Data augmentation a través de back translation

El concepto de "back translation", es una técnica primeramente creada para mejorar la calidad de modelos de traducción automática [9], aunque su uso para la DA ha sido probado para algunas tareas del PLN [9]. Esta técnica consiste en, con la ayuda de un modelo de traducción de texto, traducir una frase de un idioma a otro, y luego volver al idioma original. Esto puede provocar resultar en ningún cambio o pequeños cambios en el texto original, es por ello, que también puede hacerse traducciones intermedias antes de volver a traducir al lenguaje original.

En mi caso, con la ayuda de la librería de Python *googletrans* he podido duplicar la cantidad de datos. En concreto he probado esto tres veces, con varias combinaciones distintas de traducciones distintas.

Primer conjunto: Español → Koreano → Español

Segundo conjunto: Español → Alemán → Portugués → Español

Tercer conjunto: Español → Inglés → Alemán → Ruso → Portugués → Español

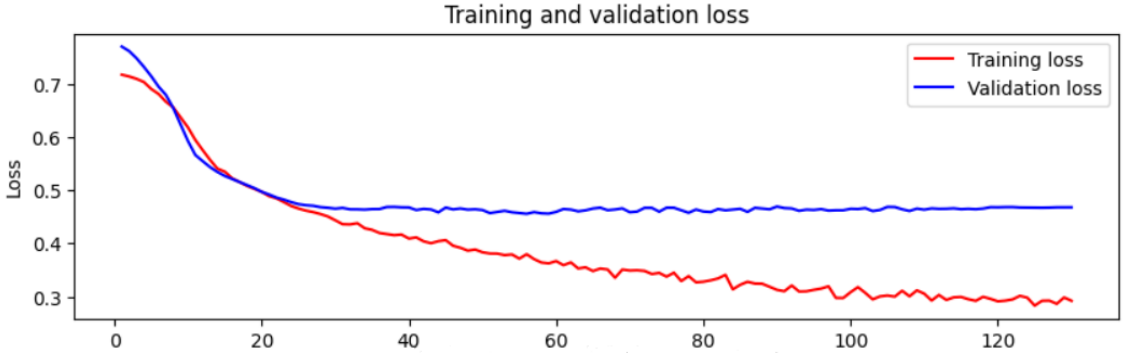
Dado que el primer conjunto solo hace una traducción he tratado de escoger una lengua muy distinta, como puede ser el coreano. En el segundo, he decidido usar el alemán como una lengua distinta , y el portugués como una lengua más similar, para tratar de que los datos varían pero tampoco demasiado. En el tercero he decidido poner los idiomas de manera aleatoria, sin un objetivo en concreto. A los modelos entrenados de esta manera nos referiremos como DATrad1, DATrad2 y DATrad3 respectivamente .

7.3 Resultados

En este sub apartado muestro todos los resultados obtenidos en cuanto a las diferentes pruebas que he podido hacer.

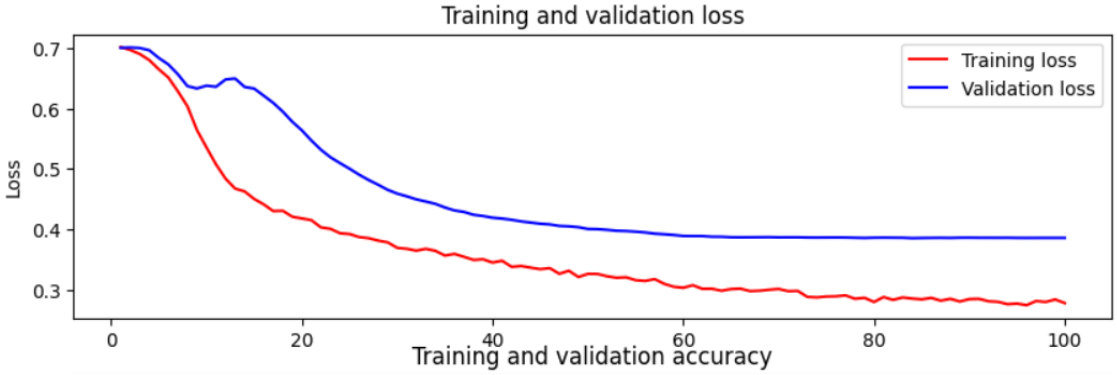
Nos referimos al modelo base como el modelo que mejores resultados ha obtenido en el apartado de optimización de hiperparámetros, el que obtiene un 0.718 de F1-score (V2). Esto es útil porque tanto el modelo, como los hiperparámetros obtenidos en esta configuración son reutilizados para algunos experimentos. No obstante, también haremos pruebas otras configuraciones.

Para los modelos de DA usado GPT como modelo generativo de datos he obtenido los siguientes resultados. Para el primer modelo DAGPT1, que usa datos nuevos generados sin supervisión de estos. Tras hacer varias pruebas con distintas configuraciones, la mejor ha resultado ser parecida a la del modelo base pero los resultados en esta empeoran la calidad del modelo, como podemos ver en la *tabla 4*. Aun así, vemos como el modelo en términos de la gráfica de pérdidas de validación y entrenamiento durante el entrenamiento, obtiene una pérdida menor en ambos casos, como se muestra en la gráfica 6, donde al final del entrenamiento la pérdida de validación queda por debajo del 0,5 y la de entrenamiento alrededor del 0,3 , donde el modelo base obtiene una mayor a a 0,5 y mayor a 0,3 respectivamente.



Gráfica 6 : Valores de pérdida por número de épocas durante el entrenamiento para modelo DAGPT1

En cuanto al segundo modelo de DA usando GPT como modelo generador de nuevos datos pero esta vez con prompts supervisados , DAGPT2 , los resultados respecto a las métricas de evaluación son prácticamente idénticos a su predecesor (DAGPT1) aunque algo mejores. Los resultados siguen siendo peores que los del modelo base, pero en este caso la pérdida de entrenamiento y validación si que hacen una reducción considerable como se puede observar en la *gráfica 7*.



Gráfica 7 : Valores de pérdida por número de épocas durante el entrenamiento para modelo DAGPT2

Para empezar vemos como la pérdida de validación y entreno se reducen mucho antes que los demás modelos, además que las funciones que estas generan son mucho más

suaves que la del modelo DAGPT1 y la del modelo base (*gráficas 6* y *2* respectivamente). Aun así, lo que marca la diferencia de este modelo con los demàs es el valor de la pérdida de validación, que se reduce en 0,1 respectos a los anteriores casos. Aun siendo esto muy bueno, los resultados no se ven reflejados directamente, y las métricas de evaluación marcan que al igual que DAPGT1, este no consigue mejores resultados que el modelo base, tal y como se indica en la *tabla 4*.

Revisando ahora los modelos entrenados usando DA pero a través de DATrad podemos ver que las métricas de estos también son peores que las del modelo base, pero si comparamos entre ellas, podemos ver como hay diferencias entre el rendimiento. En concreto, como se establece en la *tabla 4*, los resultados del DATrad1 son los mejores, y los resultados de la DATrad3 son los peores. Para ser más exactos, los resultados de este primer modelo obtienen resultados muy similares a los obtenidos en el modelo base.

Modelo	**Acuraccy**	**Precision**	**Recall**	**F1-Score**
BaseModel 1	0,75	0,661	0,785	0,718
DAGPT1	0,716	0,595	0,769	0,671
DAGPT2	0,72	0,601	0,774	0,677
DATrad1	0,742	0,648	0,779	0,707
DATrad2	0,718	0,597	0,774	0,674
DATrad3	0,724	0,605	0,777	0,681

Tabla 4: Resultados de modelos de data augmentation

Esta diferencia entre los resultados de los modelos que usan ‘back translation’, viene directamente relacionada con la calidad de los datos. Para ser exactos, seguidamente vemos ejemplos de los resultados de esta técnica en el mejor y peor de los casos.

Como ejemplo ideal, podemos ver la siguiente frase irónica sacada de los datos generados por el caso de DA por traducción 1, donde la segunda frase ha cambiado respecto a la primera pero sigue manteniendo su significado irónico.

ORIGINAL : `¿Ha dicho pan salvaje? ¿Como se hace el pan salvaje? ¿El panadero hace el pan en una hoguera en medio del campo?`

BACK TRANS 1 : `¿Dijiste pan salvaje? ¿Cómo se hace el pan salvaje? ¿El panadero hornea el pan sobre una fogata en medio del campo?`

Cabe recalcar que no todos los ejemplos son tan buenos como este, donde bajo mi consideración es el caso ideal de DA, también encontramos ejemplos donde la frase no cambia o donde el sentido de la ironía se pierde.

Por otra parte, mirando los resultados del DA por traducción 3, son peores y claramente lo podemos ver reflejado en el ejemplo siguiente ejemplo. Esta frase, ha perdido totalmente su sentido irónico y su sentido como tal.

ORIGINAL : `¿Ha dicho pan salvaje? ¿Como se hace el pan salvaje? ¿El panadero hace el pan en una hoguera en medio del campo?`

BACK TRANS 3 :`¿Dijiste ciervo? ¿Cómo se hace el pan de ciervo? ¿Un panadero hornea pan sobre un fuego abierto en medio de un campo?`

Los resultados de el DA por traducción 2 muestran ejemplos muy similares al DA por traducción 3, al igual que los resultados.

Con todo esto podemos deducir que, pocas repeticiones de traducción son favorables para esta tarea, pero que aún así, el modelo no consigue mejorar el rendimiento usando esta técnica, incluso lo empeora.

Hemos testeado el nuevo dataset de reseñas de películas y hemos obtenido los siguientes resultados usando varios modelos anteriormente mencionados. Además, hemos testeado el modelo con dos distribuciones del dataset distintas, ya que el dataset en su estado base, tiene una distribución de ejemplos desbalanceada. Para ser concretos, la distribución balanceada tiene una distribución similar al conjunto de datos usado para testear el modelo base. La distribución desbalanceada es conocida como Dist1 y la balanceada como Dist2. Los resultados mostrados en la *tabla 5*.

Los resultados de la *tabla 5*, muestran poca compatibilidad, prácticamente podríamos decir que en este caso, los modelos no pueden clasificar los datos. Esto se puede deber a varias razones, ya que los modelos sí que obtienen buenos resultados con el conjunto de testeo original.

Modelo	**Acuraccy**	**Precision**	**Reacall**	**F1-Score**
BM Dist1	0.512	0.074	0.931	0.137
BM Dist2	0.486	0.244	0.746	0.368
DATrad1 Dist1	0.544	0.077	0.934	0.143
DATrad1 Dist2	0.526	0.26	0.761	0.388
DAGPT2 Dist1	0.077	0.07	0.778	0.128
DAGPT2 Dist2	0.253	0.246	0.6	0.349

Tabla 5: Resultados de testear el dataset de reseñas versión 1, con varios modelos

Observando las distribuciones, vemos claramente que una buena distribución de testeo tiene más sentido que una distribución ilógica o sesgada. Podemos ver esto por ejemplo en el caso de BM Dist1, donde la precisión es muy cercana a 0, y en cambio, en BM Dist2, la precisión es muy cercana a 0.25, lo cual es una mejora.

En cuanto a los modelos, podemos observar que todos obtienen malos resultados. Esto se puede deber a varias razones, pero la razón más lógica se la podríamos atribuir a los datos. Como se explica en el apartado de comparación de datasets, el tipo de datos entre los datasets es muy distinto, ademàs de que, aun teniendo alrededor de 3000 frases para entrenamiento, estamos hablando de pocos datos y por último le añadimos la complejidad de la tarea, que puede ser difícil de realizar hasta por personas.

Aun así, podemos ver que los resultados no varían demasiado, incluso que en este caso, el DATrad1 funciona mejor que el modelo base, indicando de forma no muy resolutiva (ya que solo es una prueba) que ese modelo podría generalizar mejor que el modelo base.

Aun así, para mejorar esto, hay varias opciones, entre ellas aumentar la variabilidad de los datos, aumentar la calidad de los datos, aumentar los datos y/o incluso considerar otra arquitectura de modelo.

Para concluir, he probado otra versión del dataset de reseñas, que ha diferencia del anterior usado tiene como frases irónicas aquellas que al menos un anotador predijo como irónicas. Estas, solo las he probado con la distribución balanceada del dataset y en la *tabla 6* podemos ver los resultados de estas.

Modelo	**Acuraccy**	**Precision**	**Reacall**	**F1-Score**
BM	0.433	0.385	0.494	0.433
DATrad1	0.443	0.384	0.505	0.436
DAGPT2	0.433	0.41	0.49	0.447

Tabla 5: Resultados de testear el dataset de reseñas versión 2, con varios modelos

En este último caso, las conclusiones se anteriores se repiten, pero el modelo base actúa peor que los otros dos modelos, reforzando así la idea de que ese está sobre ajustado a los datos de entrenamiento.

8 MODEL DEPLOYMENT

Además de la creación del modelo de detección de ironía automático, he decidido crear una interfaz para que un usuario pueda probar el modelo.

He decidido crear una interfaz web, ya que es una plataforma muy útil y tengo experiencia en la creación y desarrollo de estas. En concreto he decidido usar HTML, con css, javascript y python. HTML y css para crear el diseño de la interfaz y javascript con python para crear el código funcional de la página. Para poder usar el modelo, existen librerías de javascript capaces de cargar el modelo en javascript como puede ser Tensorflow.js , pero no he sido capaz de que esto funcionara, con lo que he decidido mantener el código de python que realiza las predicciones del modelo y a través de llamadas, comunicar los dos programas.

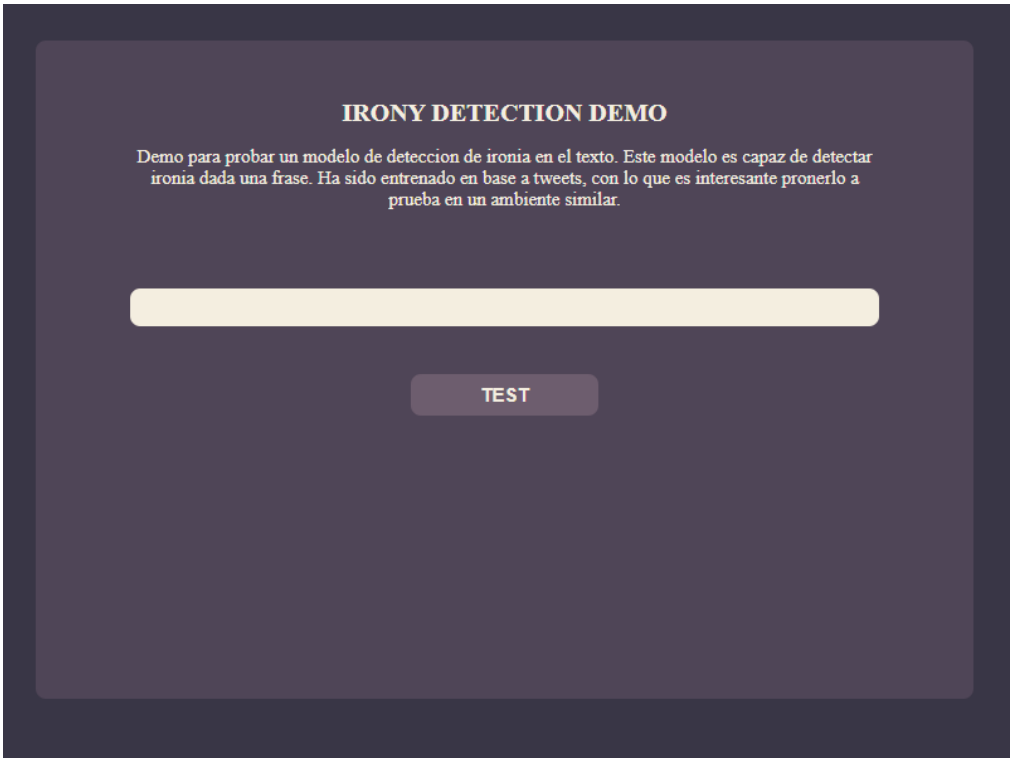


Figura 8: Interfaz para probar el modelo en web

Todo el código para ejecutar la interfaz está subido a github https://github.com/erico089/Irony-Detection-Model-TFG , pero la idea futura sería subir una web para que su uso sea público y sencillo de usar. Cabe recalcar que en este github también se encuentra todo el código usado para la creación del dataset y todo el entrenamiento de los modelos, además de varios ejemplo

9 CONCLUSIONES

La detección de ironía es una tarea complicada hasta para sus propios creadores, los humanos, por tanto, la detección de ironía automática no es una tarea fácil. Aun así, durante este trabajo conseguimos obtener varias versiones de modelos capaces de automáticamente detectar ironía en el texto en español.

Para ser más precisos, el modelo que mejores resultados obtiene, consigue resultados comparables a los resultados que obtuvieron los participantes de la tarea de IroSvA, donde mi mejor modelo generado consigue un F1-score de 0.717 y el mejor resultado de la tarea de IroSvA en la variante española obtiene un Average F1-score de 0.7167 [5].

También, conseguimos generar un nuevo dataset para la tarea de detección de ironía que puede dar diversidad en comparación a la mayoría de datasets existentes que se pueden ver para esta tarea, que suelen estar compuestos por tweets.

En cuanto a los resultados de los modelos generados y a las pruebas hechas, podemos concluir que la data augmentation en general, debe ser tratada con mucho cuidado para esta tarea. En el caso de las traducciones, los nuevos datos no sólo no aportan a la mejora del modelo, sino que lo empeoran. Esto puede carecer de sentido, pero al ser la ironía un concepto tan controversial la simple modificación de una palabra puede revertir todo su significado. Este caso también se da en muchas otras tareas como el Sentiment Analysis, donde un solo cambio de palabra puede concluir en pérdida total del sentido original.

Sobre los experimentos usando el modelo generativo de texto, el análisis se complica, pero viendo los resultados y los datos en general podemos llegar a una clara conclusión, los datos generados tienen poca variabilidad y demasiada repetición, lo que puede provocar que el modelo aprenda patrones erróneos, y más en nuestro caso dada la poca cantidad de datos que tenemos.

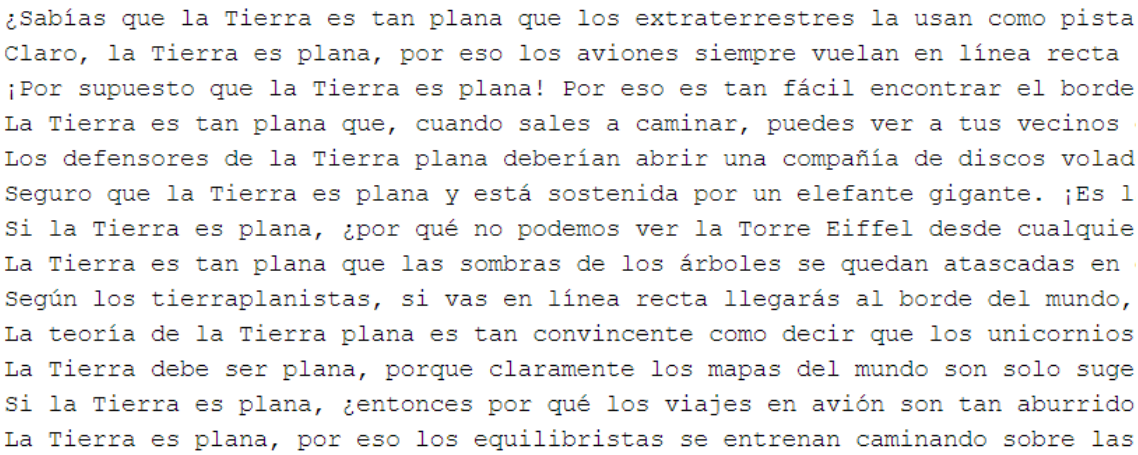


Figura 9: Captura de pantalla de datos nueva generada en el tema del Tierra Planismo

Uno de los contextos de los datos de IroSvA es el del “tierra planismo”, y en la *figura 9* es muy fácil identificar a simple vista, que muchas palabras y conjuntos de palabras se

repiten prácticamente siempre, como son “la Tierra es plana”, o “la Tierra es tan plana”. Eso puede provocar que el modelo se sesgue en estos conceptos, lo que no es la idea.

9.1 Trabajo futuro

De cara a futuros trabajos sería interesante probar más técnicas de DA en este dataset y tratar de refinar las ya probadas, lo que implica tratar de aportar más diversidad al conjunto de datos, además de identificar los problemas que tiene este modelo en relación a los altos niveles de pérdida.

También sería interesante probar otras arquitecturas como podría ser la que usa GPT3, que ha sido usada en este proyecto para generar nuevos datos irónicos. Sobre todo teniendo en cuenta, que esta última podría llegar a tener un contexto que resulta muy útil en el momento de detectar ironía, ya que en la mayoría de casos, esto es fundamental.

Bibliografia

[1] Alom, M.Z., Taha, T.M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M.S., Hasan, M., Van Essen, B.C., Awwal, A.A., & Asari, V.K. (2019). A State-of-the-Art Survey on Deep Learning Theory and Architectures. *Electronics*.

[2]Altin, L.S., Bravo, À., & Saggion, H. (2019). LaSTUS/TALN at IroSvA: Irony Detection in Spanish Variants. *IberLEF@SEPLN*.

[3]Barbieri, F., & Saggion, H. (2014). Modelling Irony in Twitter. *Conference of the European Chapter of the Association for Computational Linguistics*.

[4]Bridle, J. (1989b). Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters. In *Neural Information Processing Systems* (Vol. 2, pp. 211–217). https://papers.nips.cc/paper/195-training-stochastic-model-recognition-algorithms-as-networks-can-lead-to-maximum-mutual-information-estimation-of-parameters.pdf

[5] Bueno, R.O., Pardo, F.M., Farías, D.I., Rosso, P., Montes-y-Gómez, M., & Medina-Pagola, J.E. (2019). Overview of the Task on Irony Detection in Spanish Variants. *IberLEF@SEPLN*.

[6]Chang, C., & Lin, C. (2011). LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol., 2*, 27:1-27:27.

[7] Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement, 20*, 37 - 46.

[8] Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *ArXiv, abs/1810.04805*.

[9] Feng, S.Y., Gangal, V., Wei, J., Chandar, S., Vosoughi, S., Mitamura, T., & Hovy, E.H. (2021). A Survey of Data Augmentation Approaches for NLP. *Findings*.

[10]Fu, Z., Yang, H., So, A.M., Lam, W., Bing, L., & Collier, N. (2022). On the Effectiveness of Parameter-Efficient Fine-Tuning. *ArXiv, abs/2211.15583*.

[11] Godin, F., Vandersmissen, B., Neve, W.D., & Walle, R.V. (2015). Multimedia Lab $@$ ACL WNUT NER Shared Task: Named Entity Recognition for Twitter Microposts using Distributed Word Representations. *NUT@IJCNLP*.

[12] González, J.Á., Hurtado, L.F., & Plà, F. (2019). ELiRF-UPV at IroSvA: Transformer Encoders for Spanish Irony Detection. *IberLEF@SEPLN*.

[13]González, J.Á., Hurtado, L.F., & Plà, F. (2020). Transformer based contextualization of pre-trained word embeddings for irony detection in Twitter. *Inf. Process. Manag., 57*, 102262.

[14] Hee, C.V., Lefever, E., & Hoste, V. (2018). SemEval-2018 Task 3: Irony Detection in English Tweets. *International Workshop on Semantic Evaluation*.

[15]Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation, 9*, 1735-1780.

[16] Libraries & extensions. (s. f.). *TensorFlow*. https://www.tensorflow.org/resources/libraries-extensions

[17] Martínez-Cámara, E., Cruz, F.L., Molina-González, M.D., Martín-Valdivia, M.T., Ortega, F.J., & López, L.A. (2015). Improving Spanish Polarity Classification Combining Different Linguistic Resources. *International Conference on Applications of Natural Language to Data Bases*.

[18] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *ArXiv, abs/1310.4546*.

[19]Miranda-Belmonte, H.U., & Lopez-Monroy, A.P. (2019). Early Fusion of Traditional and Deep Features for Irony Detection in Twitter. *IberLEF@SEPLN*.

[20] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners.

[21]Reyes, A., Rosso, P., & Buscaldi, D. (2012). From humor recognition to irony detection: The figurative language of social media. *Data Knowl. Eng., 74*, 1-12.

[22] Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. *NIPS*.

[23]Von Schlegel, F. (1991b). *Philosophical Fragments*. U of Minnesota Press.

[24]Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J.R., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G.S., Hughes, M., & Dean, J. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *ArXiv, abs/1609.08144*.

[25]Zhu, Y., Kiros, R., Zemel, R.S., Salakhutdinov, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books. *2015 IEEE International Conference on Computer Vision (ICCV)*, 19-27.