



Clasificación temática de comentarios de YouTube mediante el ajuste fino de modelos Transformer

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:

Jose Luis Galiano Gómez

Tutores:

Rafael Valencia García

Jose Antonio García Díaz

20 de Enero de 2026



Clasificación temática de comentarios de YouTube mediante el ajuste fino de modelos Transformer

Análisis comparativo de rendimiento e interpretabilidad frente a aproximaciones clásicas y técnicas zero-shot

Autor

Jose Luis Galiano Gómez

Tutores

Rafael Valencia García

Departamento de Informática y Sistemas

Jose Antonio García Díaz

Departamento de Informática y Sistemas



Grado en Ingeniería Informática



DIS

Departamento de
Informática y Sistemas

UNIVERSIDAD DE
MURCIA



Murcia, 20 de Enero de 2026

Agradecimientos

Quiero dar las gracias a mis tutores, Rafael y Jose Antonio, por su compromiso e implicación con el trabajo. También agradezco enormemente a mi familia el apoyo y confianza incondicional durante estos años de carrera, y a todos mis amigos, nuevos y no tan nuevos, que me han acompañado en esta etapa.

Declaración firmada sobre originalidad del trabajo

D. Jose Luis Galiano Gómez, con DNI **52021828Z**, estudiante de la titulación de **Grado en Ingeniería Informática** de la Universidad de Murcia y autor del TF titulado “**Clasificación temática de comentarios de YouTube mediante el ajuste fino de modelos Transformer**”.

De acuerdo con el Reglamento por el que se regulan los Trabajos Fin de Grado y de Fin de Máster en la Universidad de Murcia (aprobado C. de Gob. 30-04-2015, modificado 22-04-2016 y 28-09-2018), así como la normativa interna para la oferta, asignación, elaboración y defensa de los Trabajos Fin de Grado y Fin de Máster de las titulaciones impartidas en la Facultad de Informática de la Universidad de Murcia (aprobada en Junta de Facultad 27-11-2015)

DECLARO:

Que el Trabajo Fin de Grado presentado para su evaluación es original y de elaboración personal. Todas las fuentes utilizadas han sido debidamente citadas. Así mismo, declara que no incumple ningún contrato de confidencialidad, ni viola ningún derecho de propiedad intelectual e industrial

Murcia, a 20 de Enero de 2026



Fdo.: Jose Luis Galiano Gómez
Autor del TF

Resumen

En el ecosistema digital contemporáneo contamos con repositorios masivos de información, como por ejemplo las plataformas de contenido generado por el usuario. Una de las plataformas de este tipo más importantes es YouTube, que contiene contribuciones textuales de cientos de millones de usuarios de todo el mundo en la forma de comentarios. Estos comentarios constituyen un conjunto de datos valioso para tareas de procesamiento del lenguaje natural (NLP), entre ellas la clasificación temática de texto.

Este trabajo pretende aplicar métodos de clasificación de texto automáticos basados en el NLP y el *Machine Learning* (ML) sobre un dataset de comentarios de YouTube. En concreto, haremos uso de modelos basados en la arquitectura *Transformer*, la cual supuso un salto decisivo gracias a la introducción del mecanismo de atención, principalmente. Este enfoque novedoso permite al modelo capturar patrones contextuales más complejos, cualidad especialmente interesante para tratar textos como los comentarios: informales, ambigüos y cargados de neologismos y referencias.

El desarrollo del trabajo abarca todos los pasos necesarios para implementar un sistema de clasificación temática de textos, empezando por la obtención y etiquetado de los datos a través de la API de YouTube. Tras ser recolectados, los comentarios pasan por un *pipeline* de procesado completo y reproducible, aplicando desde filtros básicos basados en reglas sencillas hasta métricas estadísticas formales como la Entropía de Shannon o la *Perplexity*. Finalmente procedemos a implementar los *pipelines* de entrenamiento. No solo aplicamos *fine-tuning* sobre un modelo DistilBERT, sino que también entrenamos un modelo *baseline* basado en TF-IDF junto a Regresión Logística para justificar el coste y complejidad del *Transformer* frente a alternativas más simples.

El análisis cuantitativo de los resultados, realizado a través de la métrica F1 macro principalmente, revela que a la hora de usar un *Transformer* el proceso de *fine-tuning* está más que justificado, ya que obtenemos una aumento del F1 de casi 20 puntos porcentuales al comparar nuestro modelo DistilBERT ajustado frente a un modelo *zero-shot*. Por otro lado, la mejora respecto al *baseline* TF-IDF es moderada, de 4 puntos porcentuales. Sin embargo, un análisis cualitativo de los resultados, analizando los errores del *baseline* que corrige el *Transformer*, revela que efectivamente el modelo DistilBERT capture relaciones complejas y no lineales que el *baseline* no sabe resolver, debido a su dependencia de palabras clave.

Finalmente realizamos un análisis de interpretabilidad del modelo, para cubrir así la principal debilidad de esta arquitectura: la cantidad de parámetros e interacciones no lineales complejas dificultan la explicación de las decisiones que toma el modelo,

convirtiéndolo en una "caja negra". El método SHAP nos permite confirmar que la importancia de un token para una predicción concreta depende del contexto en el que aparece, y la herramienta BertViz facilita la visualización de los mecanismos de atención internos, validando así que el modelo captura fenómenos complejos como el sarcasmo.

Las principales limitaciones encontradas no vienen de la arquitectura *Transformer* sino de cómo se ha enfocado la recolección y etiquetado de datos; el análisis de los resultados muestra que no existe una separación suficientemente clara de las temáticas. Por lo tanto, existen diferentes formas de evolucionar este sistema en un futuro, como por ejemplo pasar a un sistema de clasificación multietiqueta, utilizar los metadatos del dataset además del texto, o incluso pivotar a un enfoque de clasificación no supervisada en el que las etiquetas temáticas sean emergentes.

Extended Abstract

In today's digital ecosystem, we have multiple massive repositories of information and opinion. This is especially true for user-generated content platforms, among which YouTube is one of the most popular and culturally significant: YouTube encompasses contributions from hundreds of millions of users around the world. One of these contributions is comments, usually short texts that users post in the comments section of a video. If we interpret the comments posted on YouTube as a set of data to be analyzed, their value becomes apparent: they can help us understand trends, reactions, and social interactions, organize the platform's data internally, or moderate content, among many other applications. In addition, these texts are the ideal testing ground for understanding natural language, being informal in nature, lexically diverse, and presenting challenges such as sarcasm, neologisms, and ambiguities that, if resolved, allow for a deep understanding of the global context of digital conversation today. Among all these possible processing tasks, one of the most popular in recent years is topic categorization. Classifying comments according to the topic of the video in which they appear can be useful for filtering, summarizing, and prioritizing automatically, among many other possible applications.

However, the explosion of YouTube users in recent years, and with it the volume of comments posted daily, which is increasing every day, makes the manual management and analysis of this data an impossible task. It is in this context that the need arises to apply automatic text classification methods based on Natural Language Processing (NLP) and Machine Learning (ML). We can divide this task into two parts: on the one hand, there is the representation of the text, and on the other, the use of these representations to train models capable of capturing the characteristics of language through machine learning. Traditionally, the former was addressed through frequency-based representations, such as Bag of Words or TF-IDF, and these were combined with linear, classic, and efficient models, such as Logistic Regression or SVM. These simple models work well with identifiable signals in explicit words and relatively stable patterns. However, the field evolved towards Deep Learning Neural Networks, which allow more complex and non-linear patterns to be captured, eventually leading to models capable of processing sequences and capturing contextual regularities (RNNs, LSTMs). The big breakthrough came with Transformer architecture, which replaced the "sequentiality" of previous models with the new concept of attention. Attention is a mechanism whereby the meaning of a token (word or sub-word) within a text depends on its interaction with the rest of the tokens, thus generating global contextual representations capable of capturing long-distance dependencies. This improvement in

capturing more complex contextual patterns is particularly interesting for informal, ambiguous, or reference-laden language.

Even so, this superior performance offered by the Transformer architecture comes at a cost. Attention-based models, such as BERT and its variants, have a huge number of parameters and complex nonlinear interactions that make it difficult to explain why a certain prediction is what it is. This is what is popularly called a “black box”: a system whose internal workings are indecipherable. In a real-world problem, it is not enough for the model to be accurate on average; it is necessary to validate whether decisions are made based on coherent linguistic features, or whether spurious correlations and statistical noise are being captured. This need to understand the internal reasoning of the model is called interpretability, and it is precisely interpretability that is currently the weak point of models based on the Transformer architecture.

Within this context and with this limitation in mind, the proposal of this work consists of implementing a thematic classification system for YouTube comments by fine-tuning a model based on the Transformer architecture, specifically multilingual DistilBERT. The added value of this work lies in demonstrating comparatively when and why the complexity of a Transformer is justified over simpler alternatives. To do this, we also implement baseline models based on classical approaches (TF-IDF and Logistic Regression) or those that are simple or practically zero effort to implement (zero-shot learning).

The development of the work covers all the steps necessary to implement a thematic text classification system, starting with obtaining and labeling data through the YouTube API. We developed a data collection methodology that allows us to easily obtain a dataset that is sufficiently extensive for the task, maintaining thematic balance and ensuring that there are no data leaks that could influence our model with biases. After being collected, the comments go through a complete and reproducible processing pipeline. First, we apply filters based on simple rules: length, language, linguistic noise. Subsequently, we use formal statistical metrics such as Shannon Entropy, unigram diversity, and Perplexity to further filter the comments, eliminating those that, despite appearing valid, are demonstrably uninformative or noisy.

Once we have the data properly labeled and processed, the next step is to train the models. On the one hand, we train a baseline model based on TF-IDF to vectorize and represent the texts, combining it with a simple and classic logistic regression. On the other hand, we apply fine-tuning to a DistilBERT model, which is a lightweight version of BERT, a model based on the Transformer architecture. In this step, we pay special attention to the design of the training pipeline and its configuration using the chosen hyperparameters. In addition to the two trained models, we also use a zero-shot model, that is, a pre-trained Transformer for the task of Natural Language Inference (NLI), in which we reformulate the text to be classified as a premise and each label as a hypothesis, so that the model tries to infer whether the hypothesis follows from the premise. This eliminates the fine-tuning step in which we narrow the linguistic domain of the Transformer.

The comparative analysis performed after development has allowed us to justify the use of a computationally expensive model such as Transformer over simpler techniques. On the one hand, the improvement over the zero-shot model is evident: we observe an increase in F1 of almost 20 percentage points, thus fully justifying the fine-tuning process when using these models. This is especially necessary when we want to apply general language knowledge from pre-training to a specific and complex domain such as YouTube comments. On the other hand, the Transformer's success over the TF-IDF baseline is more moderate, with an improvement of less than 4 percentage points. However, error analysis confirms that the linear model relies on keywords, while the Transformer, thanks to its attention mechanism, has the ability to capture contextual and non-linear relationships, which is critical for success in complex domains such as digital.

Finally, an interpretability analysis of our DistilBERT model was performed. This was approached from two angles: external interpretability using the SHAP method, and internal interpretability using the BertViz tool. The SHAP local explainability method allowed us to verify the importance that the Transformer model attributes to words within each comment, thus validating that the weight of a token does not depend solely on its frequency or lexical characteristics, but also on the context in which it appears within the text. Moving on to internal interpretability, we generated a series of internal visualizations of the model's attention mechanisms using BertViz. This has shown that the model does not infer randomly, but rather understands phenomena such as sarcasm, negation, and the use of neologisms, which is directly reflected in the way attention heads are triggered in the multiple layers of the model, with the deepest layers being responsible for resolving the most complex linguistic ambiguities.

The greatest limitation found in the analysis lies in the semantic overlap of the defined categories. A two-dimensional PCA visualization of the dataset showed us that the vector space did not present a clear and simple separation for a model to learn, and this is clearly reflected in results such as those for the "travel" class or classification errors where the content of the comment was ambiguous enough to reasonably belong to more than one category. The high proportion of errors made by both the Transformer and the baseline model suggests that there is an upper limit to what we can achieve by further refining the training or even with more powerful architectures. It is rather a problem of inevitable residual noise in YouTube comments, as well as a suboptimal approach in terms of maximizing results.

Therefore, this work leaves ample room for the system to evolve in the future. Beyond simply testing other models or varying the training configuration, the results suggest that it is more worthwhile to work on the data labeling and processing part. For example, one possible improvement would be to implement a multi-label classification system to manage the presence of comments that address several topics at once. It would also be interesting to refine the dataset more specifically for the weaker classes, or to include additional metadata to improve the context of the comment. Finally, it is worth considering pivoting from a supervised classification approach to an unsupervised

one, leveraging the power of the attention mechanism to generate embeddings that automatically cluster comments, thereby generating a set of emerging themes rather than constraining ourselves to a fixed one.

Índice general

1. Introducción	1
2. Estado del arte	5
2.1. Clasificación de Texto Clásica y Fundamentos del Procesamiento de Lenguaje Natural (NLP)	5
2.1.1. Representación y Preprocesamiento del Texto	5
2.1.2. La Clasificación como Problema de Optimización	6
2.2. Arquitecturas Profundas (<i>Deep Learning</i>) y <i>Transformers</i>	8
2.2.1. Evolución de las representaciones de palabras: <i>word embeddings</i>	8
2.2.2. La arquitectura <i>Transformer</i>	9
2.2.3. BERT: Representaciones bidireccionales	11
2.3. Estado del arte en clasificación de texto con <i>Transformers</i>	13
2.4. Interpretabilidad de Modelos de Lenguaje	14
2.4.1. Método SHAP (<i>SHapley Additive exPlanations</i>)	14
2.4.2. SHAP en el Contexto de <i>Transformers</i> (Kernel SHAP y TransSHAP)	16
3. Metodología y objetivos	19
3.1. Objetivos	19
3.2. Metodología	20
3.2.1. Entorno de desarrollo	20
3.2.2. Proceso de desarrollo	21
4. Desarrollo del trabajo	25
4.1. Obtención de datos	25
4.1.1. Acceso a los datos: Youtube Data API v3	25
4.1.2. Diseño del flujo de obtención de comentarios	26
4.1.3. Diseño y estructura del dataset de comentarios	27
4.1.4. Script de recolección de comentarios	29
4.2. Limpieza y procesado de datos	29
4.2.1. Limpieza mínima y preparativos para el procesamiento del dataset crudo	29
4.2.2. Filtrado básico mediante reglas	32
4.2.3. Filtrado avanzado mediante métricas estadísticas de calidad	33
4.2.4. Reducción, balanceado y división del dataset	37
4.2.5. Características del dataset final	38

4.3.	Implementación de modelo <i>baseline</i> : TF-IDF + Regresión Logística	41
4.3.1.	Modelo <i>baseline</i> basado en TF-IDF y Regresión Logística	41
4.3.2.	Modelo <i>zero-shot</i> basado en <i>Transformers</i>	42
4.4.	Entrenamiento supervisado de modelo <i>Transformer</i> basado en DistilBERT	43
4.4.1.	Características del modelo a ajustar	43
4.4.2.	Diseño e implementación del <i>pipeline</i> de entrenamiento	44
4.5.	Análisis de interpretabilidad	47
4.5.1.	Explicabilidad basada en resultados: método SHAP	48
4.5.2.	Visualización interna de la atención del modelo: BertViz	48
5.	Análisis y evaluación de los resultados	51
5.1.	Entorno de evaluación y configuración final	51
5.2.	Evaluación comparativa de los modelos	52
5.2.1.	Resultados del modelo <i>baseline</i>	53
5.2.2.	Resultados del modelo <i>Transformer</i> basado en DistilBERT	58
5.2.3.	Ánalisis comparativo de resultados	61
5.3.	Ánalisis de errores	62
5.3.1.	Análisis cuantitativo	62
5.3.2.	Análisis cualitativo	63
5.4.	Intepretabilidad del modelo <i>Transformer</i>	66
5.4.1.	Explicabilidad externa: SHAP	67
5.4.2.	Visualización interna: BertViz	70
6.	Conclusiones	75
	Bibliografía	77
A.	Ejemplos de errores por modelo	81

Índice de figuras

2.1.	Autoatención mediante producto punto escalado (izquierda) y atención multi-cabeza (derecha) [1]	10
2.2.	Arquitectura del modelo <i>Transformer</i> [1]	11
2.3.	Procedimientos de pre-entrenamiento y <i>fine-tuning</i> para BERT [2] . . .	13
2.4.	Adaptación de SHAP "TransSHAP" para el modelo BERT [3]	17
4.1.	Flujo de obtención de comentarios y metadatos del script de recolección	27
4.2.	Visualización <i>Bag of Words</i> de los tokens más frecuentes en el dataset .	31
4.3.	Visualización PCA 2D del dataset previo al filtrado	32
4.4.	Distribución de la entropía de Shannon del dataset de comentarios . . .	34
4.5.	Distribución de la diversidad de unigramas del dataset de comentarios .	35
4.6.	Visualización PCA 2D del dataset tras el <i>pipeline</i> de procesado completo	38
5.1.	Matriz de confusión para el modelo <i>baseline</i> TF-IDF + Regresión Logística	55
5.2.	Matriz de confusión para el modelo <i>zero-shot</i>	57
5.3.	Matriz de confusión para el modelo <i>Transformer</i> basado en DistilBERT	59
5.4.	Curvas ROC-AUC para el modelo <i>Transformer</i> basado en DistilBERT	60
5.5.	Explicación SHAP para comentario clasificado correctamente por el <i>Transformer</i> e incorrectamente por el <i>baseline</i>	69
5.6.	Explicación SHAP para comentario clasificado erróneamente por ambos modelos	70
5.7.	Vista <i>model</i> para comentario sarcástico	71
5.8.	Vista <i>head</i> de la novena cabeza, última capa para comentario sarcástico	72
5.9.	Vista <i>head</i> de la cuarta cabeza, penúltima capa para comentario sarcástico	72
5.10.	Vista <i>head</i> de la décima cabeza, última capa para comentario sarcástico	73

Índice de tablas

4.1.	Distribución de la <i>perplexity</i> del dataset de comentarios	36
4.2.	Resultados del pipeline de procesado	39
4.3.	Longitud y estadísticas medias por clase del dataset	39
4.4.	Canales/autores únicos y número de filas por clase del dataset	40
5.1.	Métricas de rendimiento del modelo TF-IDF + Regresión Logística . .	53
5.2.	Modelo <i>baseline</i> : top 10 rasgos (tokens) más relevantes por clase . . .	54
5.3.	Métricas de rendimiento del modelo <i>zero-shot</i>	56
5.4.	Métricas de rendimiento del modelo <i>Transformer</i> basado en DistilBERT	58
5.5.	Resultados de entrenamiento de DistilBERT y BETO	60
5.6.	Comparativa de métricas globales de rendimiento	61
5.7.	Desglose de errores entre modelo <i>baseline</i> y modelo <i>Transformer</i> . . .	63
5.8.	Errores de <i>zero-shot</i> y BETO corregidos por DistilBERT	65
5.9.	Top 10 tokens con contribución SHAP (positiva/negativa) para cada clase	74

1. Introducción

En el ecosistema digital contemporáneo contamos con múltiples repositorios masivos de información y opinión. Esto es especialmente cierto para plataformas de contenido generado por el usuario, entre ellas siendo YouTube una de las más populares y con mayor presencia cultural: YouTube abarca contribuciones de cientos de millones de usuarios de todo el mundo. Una de estas contribuciones es el comentario, textos usualmente breves que los usuarios publican en la sección de comentarios de un vídeo. Si interpretamos los comentarios publicados en YouTube como un conjunto de datos a analizar, el valor de estos resulta aparente: nos pueden ayudar a entender tendencias, reacciones e interacciones sociales, a organizar internamente los datos de la plataforma o a moderar contenidos, entre muchas otras aplicaciones. Además, estos textos son el campo de pruebas ideal para la comprensión del lenguaje natural, siendo de naturaleza informal, léxicamente diversos y presentando retos como el sarcasmo, neologismos y ambigüedades que, de resolverse, permiten una comprensión profunda del contexto global de la conversación digital hoy en día. De entre todas estas posibles tareas de procesamiento, una de las más populares en los años recientes es la categorización de temáticas. Clasificar comentarios según el tópico del vídeo en el que aparecen puede ser útil para filtrar, resumir, y priorizar automáticamente, entre otras muchas aplicaciones posibles.

Sin embargo, la explosión de usuarios de YouTube en los últimos años, y con ello el volumen de comentarios publicados diariamente, que no hace más que aumentar cada día, hace que la gestión y análisis manual de estos datos sea una tarea inabordable. Es en este contexto donde surge la necesidad de aplicar métodos de clasificación de texto automáticos basados en el Procesamiento del Lenguaje Natural (NLP) y el *Machine Learning* (ML). Podemos dividir esta tarea en dos partes; por un lado está la representación del texto, y por otro el uso de estas representaciones para entrenar modelos capaces de capturar las características del lenguaje mediante aprendizaje automático. Tradicionalmente, lo primero se abordaba mediante representaciones basadas en frecuencia, como *Bag of Words* o TF-IDF, y estas se combinaban con modelos lineales, clásicos y eficientes, como pueden ser la Regresión Logística o SVM. Estos modelos simples funcionan bien con señales identificables en palabras explícitas y patrones relativamente estables. Sin embargo, el campo evolucionó hacia las Redes Neuronales Profundas (*Deep Learning*), que permiten capturar patrones más complejos y no lineales, eventualmente llegando a modelos capaces de procesar secuencias y capturar regularidades contextuales (RNNs, LSTMs.). El gran salto decisivo llegó con la arquitectura *Transformer*, que sustituía la "secuencialidad" de los modelos anteriores por

el nuevo concepto de atención. La atención es un mecanismo por el que el significado de un token (palabra o sub-palabra) dentro de un texto depende de su interacción con el resto de tokens, generando así representaciones contextuales globales y capaces de capturar dependencias a larga distancia. Esta mejora a la hora de capturar patrones contextuales más complejos es especialmente interesante para el lenguaje informal, ambigüo o cargado de referencias.

Aun así, este rendimiento superior que nos ofrece la arquitectura *Transformer* tiene un coste. Los modelos basados en atención, como por ejemplo BERT y sus variantes, cuentan con un número enorme de parámetros e interacciones no lineales complejas que dificultan explicar por qué una cierta predicción es la que es. A esto es lo que se le llama popularmente "caja negra": un sistema cuyo funcionamiento interno resulta indescifrable. En un problema real, no basta con que el modelo acierte en promedio, sino que se necesita validar si las decisiones son tomadas en base a rasgos lingüísticos coherentes, o si se están capturando correlaciones espurias y ruido estadístico. A esta necesidad de entender el razonamiento interno del modelo se le llama interpretabilidad, y es precisamente la interpretabilidad el punto débil actual de los modelos basados en la arquitectura *Transformer*.

Dentro de este contexto y con esa limitación en mente, la propuesta de este TFG consiste en implementar un sistema de clasificación temática de comentarios de YouTube mediante ajuste fino (*fine-tuning*) de un modelo basado en la arquitectura *Transformer*, en concreto DistilBERT multilingüe. El trabajo no se reduce al entrenamiento del clasificador, sino que también definimos una metodología de obtención de comentarios a través de la API de YouTube, e implementamos un *pipeline* completo y reproducible de procesamiento de los comentarios, aplicando desde filtros básicos hasta métricas estadísticas formales como la Entropía de Shannon y la *Perplexity*. Esto nos permite contar con un dataset de comentarios de un tamaño suficiente a la vez que aseguramos la ausencia de comentarios de baja información, excesivamente aleatorios o con ruido extremo que pueda introducir sesgos a nuestro modelo.

Finalmente, el valor añadido de este trabajo reside en demostrar de manera comparativa cuándo y por qué la complejidad de un *Transformer* está justificada frente a alternativas más simples. Para ello implementamos también modelos *baseline* basados en aproximaciones clásicas (TF-IDF y Regresión Logística) o de implementación sencilla o prácticamente nula (*zero-shot learning*). Realizamos un análisis cuantitativo de los resultados, pero el componente realmente diferencial e interesante del TFG viene en el análisis cualitativo posterior, en el que analizamos comentarios especialmente interesantes y casos de error, diferenciando errores cometidos por todos los modelos de aquellos errores que el *Transformer* consigue corregir. Además, incorporamos técnicas de interpretabilidad tanto externas (SHAP) como internas (la herramienta BertViz) para inspeccionar a fondo el funcionamiento de nuestro modelo y analizar cómo este resuelve fenómenos típicos del lenguaje digital: ambigüedades, sarcasmo, neologismos...

En conjunto, el enfoque de este TFG es aterrizar esa pregunta central: cuándo merece la pena pagar el coste de un *Transformer* para clasificar comentarios reales y qué evidencias (cuantitativas e interpretables) sostienen esa elección frente a *baselines* más simples.

2. Estado del arte

Antes de empezar a definir la metodología de desarrollo, en este capítulo establecemos el marco teórico fundamental del proyecto, proporcionando la base conceptual necesaria para las decisiones posteriores. Nuestro objetivo es doble: por un lado, establecer los fundamentos del Procesamiento del Lenguaje Natural (NLP), desde los modelos de clasificación clásica hasta las representaciones distribuidas; y por otro, realizar una investigación sobre el estado del arte en la clasificación de texto mediante arquitecturas avanzadas, trazando la evolución desde los modelos clásicos hasta las Redes Neuronales Profundas (*Deep Learning*) para justificar el uso del modelo *Transformer* en la tarea de clasificación de comentarios de YouTube.

2.1. Clasificación de Texto Clásica y Fundamentos del Procesamiento de Lenguaje Natural (NLP)

2.1.1. Representación y Preprocesamiento del Texto

En el Procesamiento del Lenguaje Natural (NLP), la unidad lingüística fundamental para el análisis es el token, que se obtiene mediante el proceso de tokenización [4]. Tradicionalmente, esto implicaba la segmentación en palabras, si bien los modelos modernos como los *Transformer* operan con unidades más flexibles, incluyendo subpalabras (generadas por codificación BPE) y n-gramas (secuencias contiguas de n tokens), que son esenciales para manejar la complejidad morfológica y los vocabularios extensos [5].

El preprocesamiento del texto es la etapa inicial crítica, ya que las decisiones tomadas aquí, como la tokenización y la normalización, impactan directamente y de manera muy notable en el rendimiento de los clasificadores, especialmente en modelos neuronales. Se ha demostrado que una tokenización sencilla es a menudo suficiente para el rendimiento, teniendo esta ya un gran impacto [4]. Adicionalmente, una técnica común en el procesamiento de lenguaje es la eliminación de *stopwords*, palabras muy frecuentes que, si bien son necesarias para la cohesión del texto, se consideran de bajo valor informativo o poco discriminativas para la clasificación [6].

Una representación numérica del texto usada ampliamente en los modelos clásicos es el modelo de espacio vectorial, en el que los documentos se convierten en vectores. Dentro de este modelo, la ponderación TF-IDF (*Term Frequency-Inverse Document Frequency*) es una técnica clave que asigna un peso a cada término que es directamente

proporcional a su frecuencia en un documento e inversamente proporcional a la rareza con que aparece en el corpus total [7]. Formalmente, se define como:

$$\text{tfidf}(t, d) = \text{tf}(t, d) \cdot \log\left(\frac{N}{\text{df}(t)}\right)$$

Donde $\text{tf}(t, d)$ es la frecuencia del término t en el documento d , N es el número total de documentos y $\text{df}(t)$ es el número de documentos que contienen t [7]. Este enfoque permite la representación del texto ponderando la importancia de los términos para que los algoritmos de clasificación clásicos (como la Regresión Logística) puedan distinguir las categorías.

2.1.2. La Clasificación como Problema de Optimización

La clasificación de texto, que es la tarea central de este trabajo, se enmarca dentro del *Machine Learning* clásico como un problema de optimización de funciones, donde el objetivo es encontrar los parámetros de un modelo que minimicen el error de predicción en un conjunto de datos etiquetados [5]. Para ello, es necesario definir matemáticamente cómo se modelan las probabilidades de pertenencia a una clase.

La regresión logística es un modelo discriminativo probabilístico que, en el caso multiclase, estima la probabilidad de que una observación pertenezca a una de las K categorías posibles. Para transformar la puntuación lineal (el logit, z) en una distribución de probabilidad válida (valores entre 0 y 1 que suman 1), se utiliza la función softmax (o exponencial normalizada) [8]. Una interpretación útil es ver cada vector de pesos como un prototipo de su clase, y el producto punto con el *input* actúa como una función de similitud, asignando la clase al prototipo más similar [5].

Continuando con nuestro caso, multiclase, el modelo aprende parámetros W (*weights*) y b (*biases*) tales que:

$$p(y = k | \mathbf{x}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x} + b_k)}{\sum_{j=1}^K \exp(\mathbf{w}_j^\top \mathbf{x} + b_j)}.$$

Este se trata de un algoritmo sencillo, clásico y matemáticamente sólido: la regresión logística funciona porque aprende a separar clases encontrando una "frontera de decisión" suave basada en una combinación lineal de las características. Intuitivamente, toma todas las señales numéricas que describen un ejemplo y calcula una puntuación: cuanto mayor sea esa puntuación, más "evidencias" hay a favor de una clase. Luego, en vez de decidir de manera brusca, pasa esa puntuación por una función que convierte dicha evidencia en una probabilidad entre 0 y 1, ajustándose suavemente a cómo cambian los datos. Esta función es ideal para problemas donde las fronteras entre clases no son rígidas. El modelo se entrena ajustando sus pesos para que estas probabilidades coincidan con las etiquetas observadas.

Una vez que el modelo produce una estimación de la probabilidad, es necesario cuan-

tificar qué tan lejos está esta predicción del resultado correcto. Para entrenar el modelo (es decir, encontrar los pesos óptimos), se emplea una función de pérdida que mide la distancia entre la probabilidad predicha por el modelo y la etiqueta correcta, conocida como pérdida por Entropía Cruzada (*Cross Entropy Loss*). Minimizar esta pérdida es matemáticamente equivalente a la maximización de la verosimilitud de la clase correcta, asegurando que los parámetros se ajusten para asignar la mayor probabilidad posible a la etiqueta verdadera. La naturaleza del *Cross Entropy Loss* garantiza que el error sea cero cuando la probabilidad de la respuesta correcta es uno, e infinito si es cero, siendo una métrica que empuja fuertemente al modelo a corregir predicciones incorrectas [5] [8].

Para lograr esta minimización de la función de pérdida, el algoritmo utilizado para actualizar iterativamente los parámetros del modelo (pesos y sesgos) es el Descenso de Gradiente. Este método se basa en calcular el vector de gradiente, que apunta en la dirección de máximo aumento de la pérdida, y luego mover los parámetros en la dirección opuesta, en un proceso iterativo de "descenso". Para modelos con grandes conjuntos de datos, se utiliza el Descenso de Gradiente Estocástico (SGD), que estima el gradiente a partir de un pequeño subconjunto de datos (minibatch), reduciendo el costo computacional por actualización de $O(m)$ a $O(1)$ con respecto al tamaño total del dataset [9]. En la práctica, se utilizan optimizadores avanzados como Adam para el entrenamiento de modelos *Transformer*, que ajustan dinámicamente la tasa de aprendizaje para cada parámetro [10].

El objetivo final de cualquier modelo no es solo desempeñarse bien en el conjunto de entrenamiento (*training error*), sino también en datos nunca vistos (*generalization error* o *test error*), lo que se denomina generalización. El problema surge del riesgo de sobreajuste (*overfitting*), que ocurre cuando el modelo se adapta demasiado a los detalles ruidosos del conjunto de entrenamiento, haciendo que el error de prueba sea significativamente mayor que el error de entrenamiento. Por otro lado, el subajuste (*underfitting*) ocurre cuando el modelo carece de la capacidad suficiente para aprender los patrones de los datos de entrenamiento. El control de la complejidad efectiva del modelo es crucial para equilibrar el sesgo y la varianza, y así lograr la mejor generalización [9].

Para mitigar el *overfitting* en modelos con alta capacidad (como lo son los *Transformer*), se emplea la regularización, que consiste en añadir un término de penalización a la función de pérdida. La Regularización L2 (también conocida como *ridge regression* o *weight decay*) penaliza la suma de los cuadrados de los pesos, lo que tiene el efecto de encoger los valores de los parámetros hacia cero [8]. Este encogimiento reduce la complejidad efectiva del modelo, ya que los pesos grandes son menos probables, lo que a su vez controla la varianza. Esta penalización tiene una interpretación bayesiana, ya que corresponde al logaritmo negativo de una distribución a priori Gaussiana impuesta sobre los pesos del modelo [8].

Finalmente, existen otras técnicas estándar en la tarea de clasificación como el uso de *Early Stopping* (detener el entrenamiento en el punto de menor error en el conjunto de

validación, limitando la capacidad efectiva del modelo y previniendo así el sobreajuste al conjunto de entrenamiento [9]) y la partición del dataset en conjuntos disjuntos de entrenamiento (utilizado para ajustar los parámetros del modelo), validación (utilizado para ajustar los hiperparámetros) y test (utilizado únicamente para proporcionar una estimación final, no sesgada, del rendimiento del modelo en datos no vistos). Esta división de los datos es crucial para obtener un modelo no sesgado [9].

2.2. Arquitecturas Profundas (*Deep Learning*) y *Transformers*

2.2.1. Evolución de las representaciones de palabras: *word embeddings*

Las representaciones vectoriales continuas, conocidas como *word embeddings*, surgieron para superar las limitaciones de los modelos clásicos (tokens, n-gramas) que trataban a las palabras como unidades atómicas e independientes, sin una noción inherente de similitud semántica. Un avance fundamental en este ámbito fue la introducción de Word2Vec en 2013, que propuso arquitecturas log-lineales para aprender representaciones vectoriales densas a partir de grandes volúmenes de datos [11]. Estas representaciones se denominan distribuidas porque el significado de cada palabra se codifica a través de múltiples dimensiones, lo que permite que el modelo generalice al tratar de manera similar aquellos términos que comparten atributos comunes.

Estas representaciones se obtienen fundamentalmente a través de redes neuronales, que constituyen una progresión natural de los modelos de clasificación clásicos vistos en la sección anterior. Una red neuronal puede entenderse como una estructura de unidades de cómputo organizadas en capas (entrada, capas ocultas y salida), donde cada unidad realiza una suma ponderada de sus entradas más un sesgo y aplica una función de activación no lineal [5]. En la práctica, esto equivale a apilar múltiples capas de regresión logística, permitiendo que el modelo aprenda representaciones internas de los datos (aprendizaje de representación) de forma automática. Para generar los *embeddings*, la red se entrena mediante auto-supervisión utilizando el propio texto como señal de entrenamiento para predecir la vecindad de las palabras; durante este proceso, los pesos aprendidos por la red se extraen para constituir los vectores densos que capturan la semántica del lenguaje [5].

El éxito de estos modelos reside en su capacidad para capturar regularidades sintácticas y semánticas complejas mediante operaciones algebraicas simples. Por ejemplo, se demostró que la operación vectorial "Rey - Hombre + Mujer" resulta en un vector muy cercano a la representación de "Reina" [11]. A diferencia de las representaciones dispersas basadas en conteo (como TF-IDF), los *embeddings* densos de baja dimensión requieren que los clasificadores aprendan menos pesos, lo que favorece la capacidad de generalización y el manejo de la sinonimia. No obstante, estos modelos se consi-

deran estáticos, ya que asignan un único vector fijo a cada palabra del vocabulario, independientemente del contexto específico en el que aparezca la unidad lingüística [5].

La principal limitación de las representaciones estáticas es su incapacidad para resolver la polisemia, el fenómeno por el cual una misma palabra puede tener significados distintos según su entorno. Para solventar este problema, el campo evolucionó hacia las representaciones contextualizadas, con modelos como ELMo que utilizaban redes BiLSTM para extraer características sensibles al contexto [5]. Este desarrollo culminó con la aparición de BERT, que preentrena representaciones bidireccionales profundas condicionando conjuntamente el contexto izquierdo y derecho en todas las capas de la arquitectura [2]. A diferencia de sus predecesores, estos modelos generan un vector diferente para cada instancia de una palabra basándose en sus tokens vecinos, permitiendo una comprensión mucho más precisa de los matices del lenguaje.

2.2.2. La arquitectura *Transformer*

El surgimiento de las representaciones contextualizadas descrito en la sección anterior fue posible gracias a la introducción del *Transformer* en 2017, una arquitectura que supuso un cambio de paradigma al prescindir por completo de la recurrencia y la convolución, basándose exclusivamente en mecanismos de atención. Mientras que las arquitecturas anteriores, como las RNN y LSTM, procesaban la información de manera secuencial, lo que limitaba la paralelización y dificultaba la captura de dependencias a larga distancia debido al desvanecimiento del gradiente, el *Transformer* permite relacionar señales de posiciones arbitrarias de forma directa [1].

El núcleo de esta arquitectura es el mecanismo de autoatención (*self-attention*, figura 2.1, izquierda), que permite que cada token de una secuencia integre información de todos los demás tokens del contexto para construir su propia representación vectorial. Para lograr esto, el modelo asigna tres roles distintos a cada entrada mediante proyecciones lineales: una consulta (*query*), una clave (*key*) y un valor (*value*) [1]. La importancia de un token vecino se determina calculando la similitud (mediante un producto punto escalado) entre la consulta del token actual y la clave del vecino; este resultado se normaliza mediante una función softmax para obtener unos pesos que, aplicados sobre los valores, generan la nueva representación contextualizada [5] [1].

Para aumentar la capacidad del modelo de capturar diferentes tipos de relaciones semánticas y sintácticas, el *Transformer* emplea atención multi-cabeza (*multi-head attention*, figura 2.1, derecha). En lugar de un único cálculo de atención, se ejecutan varios en paralelo sobre diferentes subespacios de representación, permitiendo que una cabeza se enfoque, por ejemplo, en la estructura gramatical mientras otra identifica entidades mencionadas [1]. En el contexto de nuestro proyecto, esta capacidad es vital para manejar la naturaleza informal y léxicamente diversa de los comentarios de YouTube, donde el sentido de una palabra depende críticamente de otros términos que pueden estar alejados en el texto.

Un bloque del *Transformer* no solo contiene la capa de atención, sino que se com-

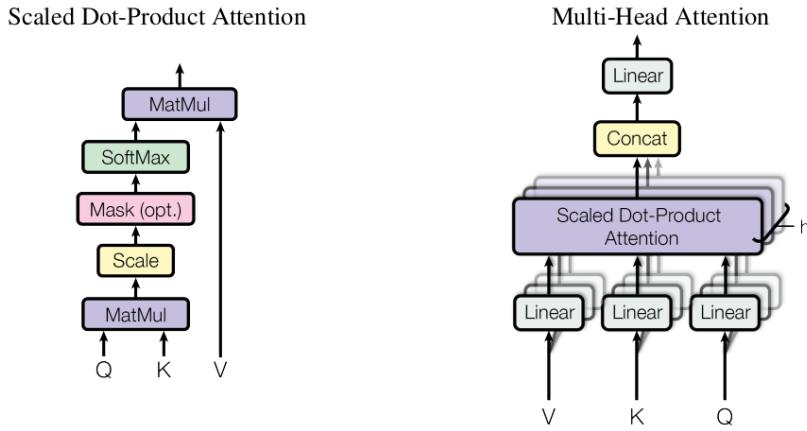


Figura 2.1: Autoatención mediante producto punto escalado (izquierda) y atención multi-cabeza (derecha) [1]

plementa con una red *feedforward* totalmente conectada, conexiones residuales y normalización de capa (*layer normalization*) [1]. Dado que el modelo no posee una noción intrínseca del orden secuencial al no ser recurrente, es necesario inyectar codificaciones posicionales (*positional encodings*) a los embeddings de entrada para que la red pueda distinguir la posición relativa de cada token.

Un bloque del *Transformer* no se limita únicamente a la capa de atención; se organiza mediante una estructura modular compuesta por capas apiladas que transforman las representaciones en abstracciones de mayor nivel. Cada bloque estándar integra, tras el mecanismo de atención, una red neuronal *feedforward* totalmente conectada (*Position-wise Feed-Forward Network*), la cual aplica transformaciones lineales y activaciones no lineales (como ReLU o GELU) de forma independiente e idéntica a cada posición de la secuencia. Para estabilizar el aprendizaje y facilitar el entrenamiento de estas arquitecturas profundas, cada una de estas subcapas se rodea de una conexión residual seguida de un proceso de normalización de capa (*layer normalization*), lo que permite que la información original fluya a través de la red y mitiga el problema del desvanecimiento del gradiente. Esta estructura modular permite apilar múltiples bloques (comúnmente entre 12 y 24 en modelos base) para aprender abstracciones cada vez más ricas [1] [2].

Un aspecto central en la definición de esta tecnología es su concepción original bajo una estructura de codificador-decodificador (*Encoder-Decoder*), diseñada inicialmente para tareas de traducción donde se transforma una secuencia de entrada en otra de salida de distinta longitud. En este esquema dual, el codificador procesa la secuencia de entrada completa para generar una representación contextualizada, mientras que el decodificador utiliza dicha información para generar la salida de manera autorregresiva [1] [5]. La figura 2.2 muestra la arquitectura completa.

Finalmente, la gran ventaja competitiva del *Transformer* es su eficiencia compu-

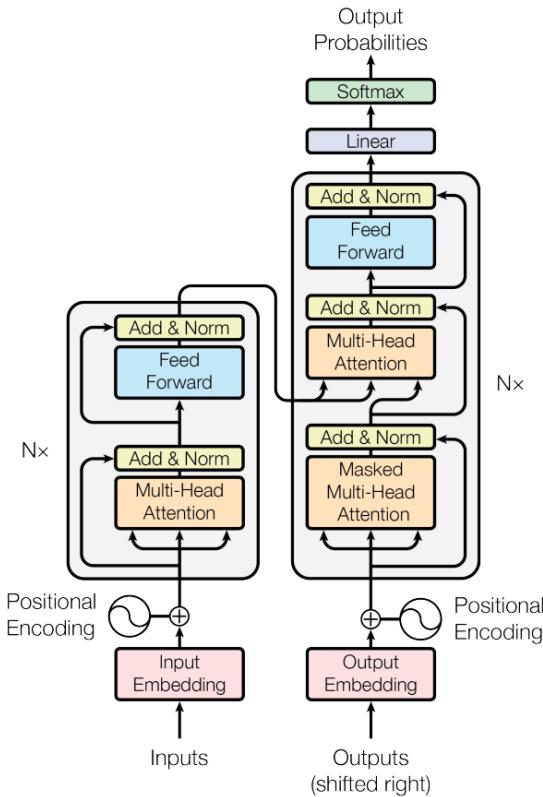


Figura 2.2: Arquitectura del modelo *Transformer* [1]

tacional y capacidad de paralelización. Al procesar todos los tokens simultáneamente, el tiempo de entrenamiento se reduce drásticamente en comparación con las RNN, permitiendo el uso de corpus masivos de datos [1]. Esta arquitectura sirve como base para el modelo BERT, que detallaremos a continuación.

2.2.3. BERT: Representaciones bidireccionales

Como hemos visto, la arquitectura *Transformer* introdujo la capacidad de capturar dependencias globales mediante la atención; sin embargo, los primeros modelos que la utilizaron seguían siendo mayoritariamente unidireccionales (de izquierda a derecha), lo que limitaba la comprensión profunda de oraciones donde el sentido de una palabra depende críticamente de lo que aparece después. Para superar esta limitación, surge BERT (*Bidirectional Encoder Representations from Transformers*), un modelo diseñado para preentrenar representaciones bidireccionales profundas condicionando conjuntamente el contexto izquierdo y derecho en todas las capas de la arquitectura [2]. A diferencia de sus predecesores, BERT no concatena dos modelos unidireccionales, sino que utiliza un codificador bidireccional puro que permite que cada token "atienda" a toda la secuencia simultáneamente [5].

Para que BERT sea capaz de procesar diversas tareas lingüísticas, su esquema de entrada utiliza una representación enriquecida. Cada secuencia comienza obligatoriamente con un token especial de clasificación, [CLS], cuyo estado oculto final se utiliza como la representación agregada para tareas de clasificación de secuencias. Además, cuando el modelo procesa pares de oraciones, estas se separan mediante el token [SEP]. La entrada final que recibe el *Transformer* es la suma de tres tipos de *embeddings*: los de token (generalmente basados en subpalabras), los de segmento (que indican si el token pertenece a la primera o segunda oración) y los posicionales (que inyectan el orden secuencial en el modelo) [2] [5].

El núcleo del éxito de BERT reside en sus dos objetivos de preentrenamiento no supervisado sobre corpus masivos (como Wikipedia y BooksCorpus):

- **Masked Language Model (MLM)**: Inspirado en la tarea de *cloze*, consiste en ocultar aleatoriamente un porcentaje de los tokens de entrada (generalmente el 15%) sustituyéndolos por el token [MASK]. El objetivo del modelo es predecir la identidad original de esos tokens basándose únicamente en su contexto bidireccional [2]. Esta técnica es fundamental porque impide que el modelo "vea" la respuesta de forma trivial, forzándolo a aprender una representación semántica profunda para reconstruir la información faltante.
- **Next Sentence Prediction (NSP)**: Dado que muchas tareas (como la inferencia de lenguaje natural) requieren entender la relación entre oraciones, BERT se entrena para predecir si una oración B sigue lógicamente a una oración A. El modelo recibe pares de oraciones donde el 50% de las veces son consecutivas y el otro 50% son aleatorias, utilizando el vector del token [CLS] para realizar esta clasificación binaria [2].

Una de las aportaciones más importantes de BERT, fundamental para nuestro trabajo, es la consolidación del paradigma de *fine-tuning* (figura 2.3); mientras que modelos anteriores como ELMo requerían diseñar arquitecturas complejas y altamente especializadas para cada tarea, utilizando las representaciones preentrenadas como entradas fijas, BERT demuestra que es posible alcanzar resultados iguales utilizando una arquitectura unificada para casi cualquier problema de lenguaje natural. Este proceso consiste en inicializar el modelo con los parámetros obtenidos durante el preentrenamiento y ajustarlos de extremo a extremo añadiendo únicamente una capa de salida adicional y ligera (generalmente una matriz de pesos sobre el estado oculto del token [CLS]) [2]. La gran ventaja práctica de esta estrategia reside en su eficiencia operativa y computacional; frente a los masivos recursos requeridos para el preentrenamiento, el ajuste fino es notablemente económico, pudiendo replicarse en apenas unas pocas horas utilizando una GPU convencional. Este enfoque es extremadamente robusto incluso con pocos datos etiquetados [2]. En lugar de diseñar arquitecturas complejas para cada problema, el conocimiento lingüístico ya está codificado en los pesos de BERT, permitiendo que el modelo alcance resultados más que aceptables con un coste computacional de ajuste relativamente bajo.

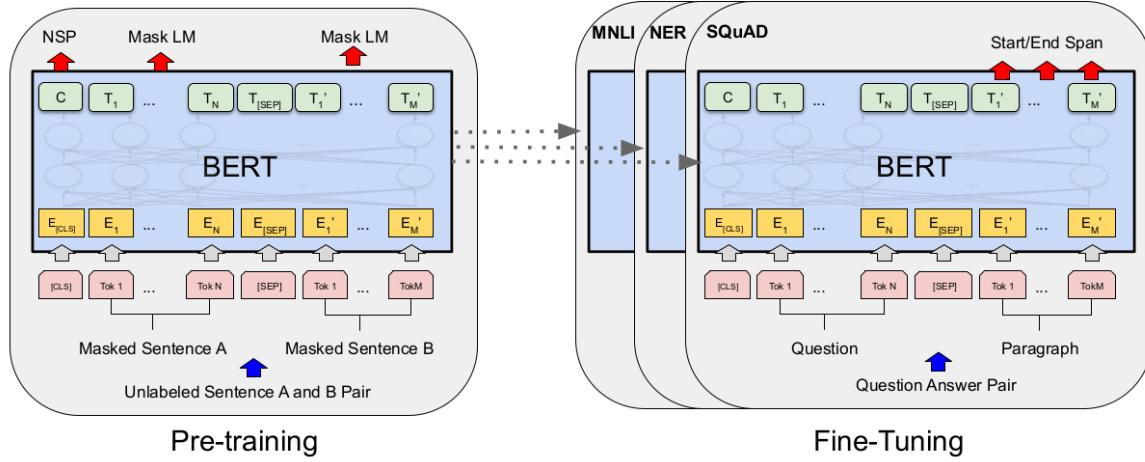


Figura 2.3: Procedimientos de pre-entrenamiento y *fine-tuning* para BERT [2]

2.3. Estado del arte en clasificación de texto con *Transformers*

Tras haber explicado los fundamentos teóricos de las arquitecturas basadas en atención, pasamos a examinar diversos casos prácticos y estudios recientes que ilustran cómo se ha aplicado esta tecnología en tareas reales de clasificación de lenguaje natural en diversos dominios.

Un caso de uso directo de la arquitectura BERT es la clasificación de categorías de noticias (negocios, entretenimiento, política, deportes y tecnología). Un estudio reciente demostró que el uso de modelos como `bert-base-uncased` permite alcanzar una precisión cercana al 94%, superando notablemente a clasificadores clásicos como *Naïve Bayes* [12]. La superioridad de este enfoque radica en la capacidad de BERT para capturar el contexto y las relaciones semánticas entre palabras, lo cual es fundamental cuando las categorías comparten vocabularios parcialmente solapados.

La eficacia de la transferencia de aprendizaje (*transfer learning*) con *Transformers* se hace especialmente evidente en áreas con lenguajes técnicos o escasez de datos etiquetados, como la clasificación anatómica en informes radiológicos. Un estudio mostró que modelos BERT preentrenados en corpus clínicos logran clasificar textos con un rendimiento superior a *baselines* de tipo BiLSTM o TF-IDF, incluso en categorías que cuentan con muy pocos ejemplos positivos en el conjunto de entrenamiento [10]. Esto sugiere que el conocimiento lingüístico previo codificado en el Transformer compensa la falta de datos específicos, una propiedad valiosa para cualquier clasificador de dominio especializado.

Además del *fine-tuning*, en el estado del arte se está explorando también el aprendizaje *zero-shot* (ZSL), que permite clasificar secuencias sin necesidad de un conjunto de entrenamiento anotado para las clases específicas. Una técnica predominante consiste

en reformular la clasificación como una tarea de Inferencia de Lenguaje Natural (NLI): el texto se presenta como una "premisa" y cada etiqueta se transforma en una "hipótesis" (por ejemplo, "Este texto trata sobre política"). Utilizando modelos preentrenados en NLI (como BART) se pueden obtener clasificadores "listos para usar" con una capacidad de generalización notable ante etiquetas nunca vistas por el modelo durante su entrenamiento original [13].

Finalmente, es crucial considerar la relación entre el rendimiento y la eficiencia computacional. En un estudio donde se evalúa la clasificación de documentos extensos, se ha observado que, si bien modelos avanzados como Longformer o RoBERTa suelen liderar las métricas de precisión, una arquitectura clásica basada en TF-IDF y Redes Neuronales constituye un baseline sumamente sólido y eficiente. En muchos escenarios, este enfoque tradicional resulta ser cientos de veces más rápido (especialmente en ejecuciones sobre CPU) con una pérdida de rendimiento de apenas el 1-2% respecto al estado del arte [14], lo que obliga a justificar cuidadosamente la necesidad de recurrir a la complejidad de los *Transformers* en función de los requisitos del proyecto.

2.4. Interpretabilidad de Modelos de Lenguaje

A pesar del excelente rendimiento de las arquitecturas basadas en atención para la clasificación de texto, estos modelos presentan un desafío fundamental: su naturaleza de "caja negra" [15]. La complejidad intrínseca de los modelos como BERT radica en la existencia de millones de parámetros y en las interacciones altamente no lineales que ocurren en sus múltiples capas de atención. Aunque el mecanismo de autoatención permite rastrear qué tokens influyen en otros, en la práctica, estas atenciones no siempre se correlacionan de forma directa con la importancia semántica que un humano atribuiría a una palabra.

Esta falta de transparencia genera una barrera entre la precisión del modelo y su explicabilidad. En el contexto de nuestro clasificador de comentarios de YouTube, no basta con que el modelo asigne correctamente una etiqueta; lo interesante sería validar que la decisión se basa en rasgos lingüísticos coherentes y no en sesgos del dataset o ruidos estadísticos.

2.4.1. Método SHAP (*SHapley Additive exPlanations*)

Para abordar la opacidad de los modelos complejos, el marco SHAP se ha consolidado como un método unificado de explicabilidad que asigna a cada característica una contribución cuantitativa única para una predicción específica. Este enfoque se basa en la identificación de una nueva clase de medidas de importancia denominadas métodos de atribución aditiva de características, los cuales definen un modelo de explicación g que es una función lineal de variables binarias:

$$g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i$$

En esta expresión, g representa el modelo de explicación, $z' \in \{0, 1\}^M$ es el vector de variables binarias (donde 1 indica que la característica está "presente" y 0 que está "ausente"), M es el número de características de entrada y $\phi_i \in \mathbb{R}$ es el efecto atribuido a cada rasgo [15].

La base de SHAP son los valores de Shapley, propuestos originalmente por Lloyd Shapley en 1953 y derivados de la teoría de juegos cooperativos, donde las características de entrada actúan como "jugadores" en una coalición y la predicción del modelo es el "pago". En el contexto del *Machine Learning*, los "jugadores" son las características de entrada (tokens del comentario) y el "pago" es la predicción del modelo.

Para formalizar esta base, el método SHAP define los valores de atribución ϕ_i mediante la siguiente expresión matemática, que representa la única solución posible que satisface un conjunto de propiedades deseables [15]:

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|! (M - |z'| - 1)!}{M!} [f_x(z') - f_x(z' \setminus i)]$$

En esta formulación, f es el modelo original, x es la entrada a explicar y x' es su representación simplificada. El término $|z'|$ representa el número de entradas no nulas en el vector binario z' , mientras que la notación $z' \subseteq x'$ indica que se consideran todos los vectores z' cuyas entradas no nulas son un subconjunto de las de x' . La clave de la ecuación está en el término $[f_x(z') - f_x(z' \setminus i)]$, el cual cuantifica la contribución marginal de la característica i al comparar la predicción cuando dicha característica está presente frente a cuando se omite [15]. Al promediar estas contribuciones sobre todas las permutaciones posibles de las características, SHAP garantiza una distribución justa del impacto de cada palabra en la clasificación final del comentario [15].

SHAP se define como un método de atribución aditiva de características, lo que significa que la explicación sigue un modelo lineal de variables binarias donde la suma de las contribuciones de cada rasgo (ϕ_i) iguala la salida del modelo original. Un valor positivo indica que el token empuja la predicción hacia la clase elegida, mientras que un valor negativo indica que se opone a ella [15].

La superioridad teórica de SHAP frente a otros métodos (como LIME) reside en que es la única solución que satisface tres propiedades críticas [15]:

- **Precisión Local (Eficiencia):** La suma de las atribuciones de las características debe ser igual a la diferencia entre la predicción del modelo para esa instancia y el valor base esperado (el promedio de las predicciones del modelo).
- **Ausencia (*Missingness*):** Si una característica no está presente en la entrada original, su atribución debe ser cero.

- **Consistencia:** Si un modelo cambia de modo que la contribución marginal de una característica aumenta, su atribución SHAP no debe disminuir.

2.4.2. SHAP en el Contexto de *Transformers* (Kernel SHAP y TransSHAP)

La aplicación de SHAP en modelos basados en la arquitectura *Transformer* presenta retos técnicos específicos debido a la complejidad de sus interacciones no lineales y su enorme cantidad de parámetros. Para gestionar esto, el marco de SHAP utiliza principalmente dos aproximaciones: Kernel SHAP y Deep SHAP. Kernel SHAP es un método agnóstico al modelo que estima la importancia de las características mediante una regresión lineal ponderada, utilizando un kernel de Shapley que garantiza el cumplimiento de propiedades teóricas como la consistencia y la precisión local. Por su parte, Deep SHAP es una variante diseñada específicamente para redes profundas que integra los valores de Shapley con la lógica de DeepLIFT; este método aprovecha la naturaleza composicional de la red para propagar las contribuciones de las características hacia atrás a través de todas las capas, lo que resulta en un cálculo más eficiente en modelos con millones de parámetros [15].

El mayor obstáculo al interpretar estos modelos es el uso de la tokenización por subpalabras, fundamental en BERT para manejar vocabularios extensos y palabras fuera de diccionario [5]. Dado que el significado semántico completo suele residir en la palabra y no en sus fragmentos aislados, los métodos de perturbación (enmascaramiento) tradicionales suelen fallar al intentar asignar crédito de forma intuitiva para un analista humano. Para solventar esta desconexión, surge TransSHAP como una adaptación que permite gestionar las explicaciones en clasificadores de tipo *Transformer* (figura 2.4). TransSHAP opera mediante una función de clasificación personalizada que recibe el texto íntegro, realiza las perturbaciones a nivel de palabra y, posteriormente, mapea estos fragmentos a la representación de subpalabras que el modelo BERT necesita para generar sus predicciones [3].

Esta metodología permite realizar un análisis secuencial de la importancia de los tokens, respetando la estructura gramatical del comentario original en lugar de presentar una simple lista de palabras clave. En el contexto de nuestro proyecto, esto es vital para entender cómo el modelo utiliza el principio de exclusión contextual, donde la predicción final no solo depende de la presencia de términos positivos para una clase, sino también de la contribución negativa de tokens asociados a categorías competidoras. De este modo, SHAP puede revelar si el *Transformer* está capturando correctamente el significado de un neologismo o jerga humorística de YouTube basándose en su contexto, o si la predicción está siendo interferida por la fuerza de señales léxicas de clases dominantes.

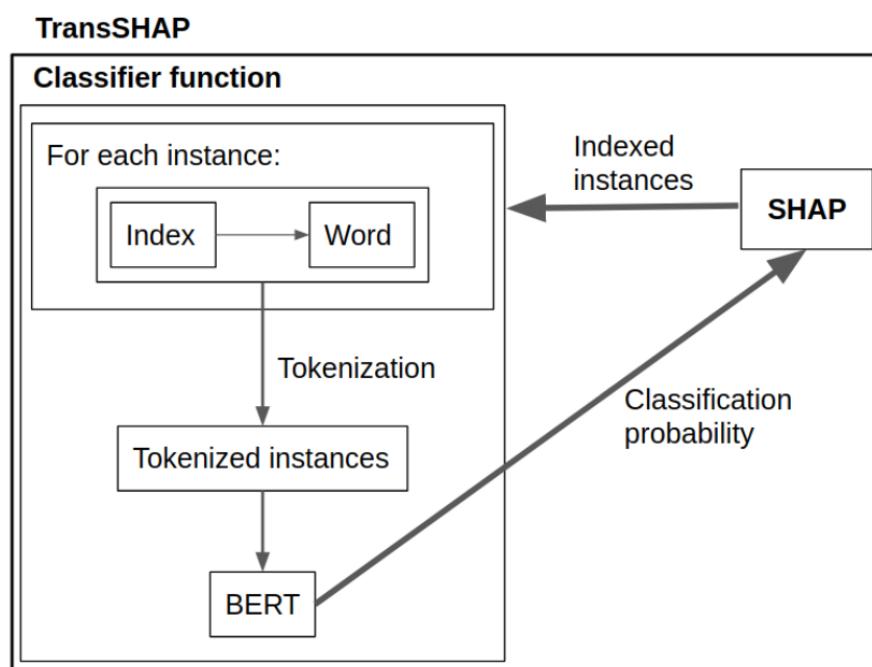


Figura 2.4: Adaptación de SHAP "TransSHAP" para el modelo BERT [3]

3. Metodología y objetivos

El análisis de los objetivos y la metodología a seguir en el desarrollo del presente proyecto es un pilar fundamental para garantizar la coherencia y viabilidad del mismo. Primero estableceremos nuestro objetivo y posteriormente pasaremos a estudiar la estrategia metodológica seguida para llevarlo a cabo, incluyendo el proceso de desarrollo escogido y las herramientas utilizadas.

3.1. Objetivos

El objetivo general de este TFG es poner en marcha un modelo capaz de clasificar comentarios de vídeos de la plataforma YouTube según la temática del vídeo en el que fueron publicados. En esencia, se trata de una tarea de clasificación de texto escrito en lenguaje natural, lo cual se enmarca dentro del campo del *Natural Language Processing* (NLP), a su vez dentro de las ciencias de la computación. Para ello, utilizaremos técnicas de procesamiento de lenguaje natural basadas en *Word Embeddings* y modelos de *Deep Learning* basados en la arquitectura *Transformer*.

Podemos desglosar este objetivo en los siguientes objetivos específicos:

- Elaborar un dataset de comentarios de YouTube correctamente etiquetados por temática. Este deberá ser suficientemente extenso y variado, además de poseer idealmente un número de muestras de cada clase balanceado. Una vez obtenidos los datos, procederemos a limpiar y preprocesar debidamente el dataset elaborado para asegurar la máxima calidad posible de los datos.
- Estudiar las distintas opciones de modelos *Transformers* base disponibles para ajustar a nuestra tarea, así como las herramientas y plataformas necesarias para realizar el trabajo. Una vez elegido el modelo, establecer un *pipeline* de entrenamiento adecuado para conseguir el mejor resultado posible.
- Explicar los resultados del modelo entrenado usando técnicas de interpretabilidad global y local, visualización interna, y análisis de casos extremos y errores.
- Realizar un análisis comparativo del rendimiento de nuestro modelo basado en *Transformers* y un modelo *baseline* de menor exigencia computacional, con el objetivo de justificar el uso de la solución desarrollada.

3.2. Metodología

La metodología seguida se ha diseñado a partir de la unión de los objetivos previamente especificados y el análisis del estado del arte realizado en el capítulo anterior. El análisis de la misma se divide en dos secciones: el entorno de desarrollo elegido y el proceso por fases escogido para el desarrollo.

3.2.1. Entorno de desarrollo

El entorno de desarrollo engloba todas las herramientas escogidas para el desarrollo del presente trabajo, desde la elección de bibliotecas y plataformas que implementen las técnicas de NLP y *Deep Learning* requeridas, hasta la elección de sistema operativo en el que realizar y ejecutar la implementación. A continuación se listan todas las decisiones técnicas que configuran el entorno de desarrollo:

- **Hardware utilizado:** En el campo del *Machine Learning* el hardware es de suma importancia a la hora de delimitar qué se puede realizar o no, en concreto la tarjeta gráfica o GPU de la que se disponga. En este caso contamos con una GPU NVIDIA GeForce GTX 1060Ti integrada en un portátil. Esta es una tarjeta de gama media equipada con 4GB de VRAM y basada en la arquitectura Turing, sin unidades Tensor dedicadas. Aunque no está diseñada para cargas pesadas de trabajo como las que suelen ser los entrenamientos de modelos *Transformer*, sí que es suficiente para entrenar modelos ligeros como DistilBERT, siempre y cuando se empleen longitudes de secuencia moderadas y *batching*.
- **Lenguaje de programación:** La elección de lenguaje en este caso es evidente: Python es la opción ideal tanto para tareas de procesado de datos como para *Deep Learning*, gracias a su maduro ecosistema de bibliotecas optimizadas y una enorme comunidad. Algunas de las bibliotecas utilizadas en prácticamente todas las fases del desarrollo son `numpy`, `pandas` y `matplotlib`. En los próximos puntos se especificarán exactamente qué bibliotecas utilizamos para cada tarea.
- **Sistema operativo y entorno local:** Se ha realizado el trabajo íntegramente en el sistema operativo Ubuntu 24.04.3 LTS. Los sistemas con arquitectura basada en Linux como Ubuntu son una excelente opción para la ejecución local de scripts y el manejo de paquetes y dependencias. Se ha optado por un entorno de desarrollo local debido a que la ejecución de scripts localmente permite un control más claro de dependencias, versiones y parámetros; para el manejo de estas se ha utilizado la herramienta `pyenv`. La alternativa, el uso de notebooks como *Jupyter* o *Colab* no ofrece tanta versatilidad y no está tan optimizada para entrenamientos prolongados como un script dedicado ejecutado desde terminal.
- **Obtención de los datos:** La elaboración del dataset de comentarios se ha realizado mediante la API oficial de datos de YouTube. Esta provee toda la informa-

ción necesaria para la elaboración del dataset, además de ser lo suficientemente permisiva con sus tasas de uso como para elaborar un dataset de tamaño considerable sin demasiadas esperas o problemas de cuotas. Por estos motivos basta con un script que interactúe con la API y formatee la información correctamente para elaborar el dataset, sin necesidad de implementar un *scraper* que extraiga la información textual manualmente. Los datos se guardan en formato CSV, simple, ligero y altamente compatible.

- **Técnicas de procesado de lenguaje natural:** Tal y como comentábamos anteriormente, Python ofrece una variedad de bibliotecas de procesado de datos y lenguaje natural extensa. Para filtrar los comentarios por contenido textual, se han utilizado las bibliotecas `re`, `emoji` y `langid`. Para computar métricas sobre el conjunto de datos se ha utilizado la biblioteca `math`, y para generar reportes explorativos que nos permitan visualizar en general las características del dataset se ha utilizado la herramienta `ProfileReport` de la biblioteca `ydata_profiling`. Además, se ha usado la herramienta PCA de `scikit-learn` para reducir la dimensionalidad de los datos con el método PCA. En el capítulo de desarrollo se detallará cómo se usa cada una de estas herramientas.
- **Modelos de Deep Learning:** El entrenamiento de nuestro modelo ajustado se ha realizado mediante la biblioteca `transformers` de *Hugging Face* con backend `PyTorch`. Esta ofrece implementaciones optimizadas, flexibles y ampliamente probadas y respaldadas por una extensa comunidad, a la vez que el acceso a las herramientas es sencillo y cómodo sin dejar de permitir un alto grado de personalización y ajuste del *pipeline* de entrenamiento. Para entrenar el modelo *baseline* y generar las métricas se ha utilizado `scikit-learn`, que proporciona métodos fiables, estandarizados y ampliamente utilizados para entrenar y evaluar modelos.
- **Técnicas de interpretabilidad:** Para interpretar el modelo se ha usado el método SHAP. La biblioteca Python `shap` ofrece todo lo necesario para computar estos valores y generar gráficos interactivos. También se ha utilizado la herramienta BertViz, que permite visualizar de forma intuitiva las matrices de atención del *Transformer*, a través de la biblioteca Python `bertviz`.

3.2.2. Proceso de desarrollo

El proceso de desarrollo sigue un enfoque secuencial que combina investigación, implementación y validación experimental de manera iterativa y por fases, de manera que cada etapa se construye sobre los resultados de la anterior. En gran medida, las fases del desarrollo se corresponden con los objetivos específicos determinados anteriormente. Se distinguen las siguientes fases:

1. **Investigación inicial y estudio del estado del arte:** Antes de iniciar el desarrollo, es importante realizar una investigación inicial que por un lado nos dará los fundamentos teóricos necesarios para entender los métodos y tecnologías utilizadas y poder aplicarlas de la manera más adecuada y óptima, y por otro para entender las fortalezas, limitaciones y puntos críticos actuales en las herramientas y técnicas implementadas, permitiendo un análisis de los resultados productivo y coherente.
2. **Elaboración del dataset mediante la API de YouTube:** En esta fase se debe implementar un script que interactúe con la API de YouTube para elaborar el dataset de comentarios en formato CSV. Dentro de esta tarea se incluye un análisis de la API, qué herramientas y puntos de acceso nos ofrece, y cómo podemos utilizarla para obtener un conjunto de comentarios balanceado según la temática de los vídeos en los que fueron publicados. También se atiende en esta fase al formato en el que guardamos los datos, teniendo que decidir qué atributos tendrá cada comentario.
3. **Limpieza y procesado de los datos:** Una vez tenemos el dataset, es importante filtrarlo para eliminar aquellos comentarios que puedan perjudicar el rendimiento de nuestro modelo. Para ello aplicamos una serie de técnicas de procesamiento del lenguaje natural.
4. **Desarrollo de modelo *baseline*:** Antes de lanzarnos al entrenamiento de nuestro modelo *Transformer*, primero elaboramos un modelo *baseline* basado en una técnica más sencilla y ligera computacionalmente. De esta manera podremos realizar un análisis comparativo más adelante para ver si es justificable el aumento significativo de costo computacional al entrenar un *Transformer*.
5. **Desarrollo de modelo *Transformer*:** En esta fase se desarrollará un *pipeline* de entrenamiento en el que tendremos que estudiar distintas opciones y tomar una serie de decisiones, prestando especial atención a la elección del modelo base que vamos a ajustar, los hiperparámetros del entrenamiento, y las técnicas que aplicamos para prevenir el sobreajuste, entre otros problemas.
6. **Evaluación de resultados y análisis de intepretabilidad:** Para evaluar cómo ha aprendido el modelo y si su comportamiento es fiable utilizaremos métricas de rendimiento (como *accuracy* o *F1 macro*) junto a técnicas de intepretabilidad (como SHAP o BertViz), que nos permiten entender qué información del texto de entrada utiliza el modelo y cómo procesa internamente las relaciones entre tokens. Esto nos ayudará a detectar errores, sesgos y limitaciones.
7. **Análisis final de los resultados:** Una vez finalizado el desarrollo, se realiza un estudio posterior en el que se analizan los resultados finales del desarrollo, extrayendo conclusiones sobre la tarea en cuestión, fortalezas y limitaciones de

las tecnologías utilizadas, áreas de mejora en nuestro método y posible trabajo futuro.

Cabe destacar que esta secuencia de fases no es del todo estricta, puesto que en el desarrollo del trabajo a veces se ha necesitado volver a una fase anterior para ajustar algo, o volver a realizar una investigación sobre algo que inicialmente no se pensaba que se iba a usar, entre otras incidencias. En general, el trabajo de las fases tercera, cuarta, quinta y sexta resulta especialmente iterativo, siendo muy común el uso de resultados de la fase de evaluación para realizar cambios en el procesado de datos o en el *pipeline* de entrenamiento, produciendo esto a su vez nuevos resultados para evaluar e interpretar.

4. Desarrollo del trabajo

En este capítulo se describe en detalle el proceso de desarrollo de trabajo, siguiendo la estructura por fases establecida en el capítulo anterior. Para cada fase se atenderá al diseño e implementación de los distintos componentes que integra la solución propuesta en este trabajo, ya sean scripts Python, conjuntos de datos, gráficos y métricas, entre otros. En el caso de los scripts Python, se hará referencia a ellos cuando sea necesario a través de hipervínculos (en el nombre de los scripts a los que se van haciendo referencia) que dirigen a un repositorio en *GitHub* con todo el código fuente. De esta manera evitamos saturar la presente memoria con páginas y páginas de código.

4.1. Obtención de datos

Una vez realizada la investigación inicial sobre el estado del arte podemos comenzar el desarrollo con la primera fase de implementación real; la obtención de los comentarios que usaremos para entrenar nuestro modelo. Primero estudiaremos cómo es el acceso a estos comentarios, para poder determinar qué podemos o no hacer y diseñar un flujo de recuperación de los mismos. Seguidamente decidiremos la estructura del dataset que queremos elaborar, es decir, qué información guardaremos para cada comentario y con qué formato, y finalmente implementaremos el script que genere nuestro dataset de comentarios en base al diseño establecido previamente.

4.1.1. Acceso a los datos: Youtube Data API v3

YouTube Data API v3 es la API que ofrece Google para permitir a aplicaciones externas interactuar con recursos de YouTube, sean estos vídeos, canales, listas de reproducción o, en nuestro caso, comentarios. Al tratarse de una API REST la comunicación con esta funciona mediante peticiones HTTP que devuelven respuestas en formato JSON. Cada tipo de recurso se expone a través de métodos (`list`, `insert`, `update`, `delete...`) organizados en la referencia oficial de la API [16]. Para usar YouTube Data API v3 es necesario crear primero un proyecto en Google Cloud Console en el que habilitamos el servicio y creamos credenciales asociadas al proyecto. En nuestro caso utilizaremos una API Key, puesto que sólo pretendemos acceder a datos públicos.

Existe un sistema detallado de cuotas para limitar el uso de la API que tendremos que tener en cuenta a la hora de diseñar nuestro script de recolección de comentarios. Los elementos más importantes a considerar son los siguientes [17]:

- Cada proyecto dispone por defecto de 10 000 unidades de cuota al día.
- Cada petición consume cuota, incluso si es inválida.
- `videos.list`, `commentThreads.list` y `comments.list` suelen costar 1 unidad por llamada.
- `search.list` consume 100 unidades por llamada.
- Cada página adicional recuperada mediante `pageToken` consume una llamada adicional.

Para la recuperación de los comentarios nos interesan principalmente los siguientes *endpoints*:

- `search.list`: Devuelve vídeos, canales o listas de reproducción en función de los parámetros utilizados, siendo el más interesante el parámetro `q`, una cadena de búsqueda por *keywords*, muy similar a lo que sucede al introducir consultas textuales en la barra de búsqueda de la propia página web de YouTube (exceptuando la personalización algorítmica que la API no implementa)[18].
- `videos.list`: Se puede usar o bien para listar vídeos según categoría e idioma, o bien para recuperar metadatos detallados de vídeos cuyo identificador ya conocemos. El más importante de estos metadatos es el llamado `snippet`, que a su vez proporciona datos sobre el título, descripción, canal, idioma y categoría del vídeo [19].
- `commentThreads.list`: Permite recuperar los comentarios de nivel superior (aquellos que no sean respuestas a otros comentarios) asociados a un vídeo, una vez conocemos su identificador. Estos también contienen un campo `snippet` que, al igual que los vídeos, ofrece metadatos sobre el comentario (autor, número de *likes*...) [20].

4.1.2. Diseño del flujo de obtención de comentarios

Ahora que tenemos una visión general de la disponibilidad de la información relacionada con comentarios, el siguiente paso es decidir cómo vamos a recuperar los comentarios y qué datos vamos a guardar para cada uno de ellos. La estructura del dataset que pasaremos a procesar está intrínsecamente ligado a esta decisión, en otras palabras, al diseño del script de recolección.

Para acertar en este diseño es importante recordar la tarea de clasificación para la que vamos a utilizar este dataset: clasificación por temas. Esto implica que más adelante en el desarrollo, cuando procesemos los datos, querremos balancear la cantidad de instancias de cada clase en el dataset, es decir, que haya más o menos la misma cantidad de comentarios de cada tema. Aunque podríamos simplemente escoger los vídeos

subidos más recientemente o los más populares, esta forma de proceder puede dar lugar a desbalances serios difíciles de remediar (o que incluso pueden dar lugar a un dataset inutilizable). Lo más sensato en este punto es recuperar vídeos según su temática por lotes de un tamaño dado, y para cada uno de estos vídeos recuperar un número fijo de comentarios. De esta manera sabremos que tenemos, aproximadamente, una cantidad igual de comentarios según el tema del vídeo en el que fueron publicados. Filtrar vídeos por temática es posible tanto mediante `search.list` como con `videos.list`. Tras probar ambos métodos, usar `search.list` con *keywords* nos ha dado resultados más coherentes que usar `videos.list` filtrando por categoría (este es un valor interno asignado por YouTube y, francamente, no muy bien asignado). Esta manera de proceder cobra aún más sentido cuando tenemos en cuenta que el método `videos.list` nos permite recolectar metadatos sobre un vídeo una vez tenemos su identificador. Estos datos junto a los recuperados con `commentThreads.list` nos permitirán nutrir cada instancia de nuestro dataset con metadatos relacionados con el comentario.

De esta manera, el flujo de obtención de los comentarios será el siguiente:

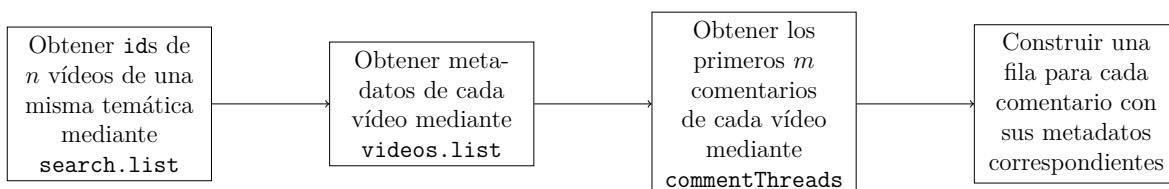


Figura 4.1: Flujo de obtención de comentarios y metadatos del script de recolección

Siguiendo este flujo, cada vez que ejecutemos el script le pasaremos un *keyword* que represente la temática, un valor n , un valor m , y este creará $n \times m$ filas, cada una un comentario con toda la información asociada al mismo.

4.1.3. Diseño y estructura del dataset de comentarios

Antes de pasar a la implementación, necesitamos determinar dos cosas: qué metadatos vamos a querer para cada comentario, en otras palabras, cuáles van a ser las columnas de nuestro dataset, y también tenemos que definir las temáticas a las que van a pertenecer los comentarios. Habiendo seleccionado los métodos de la API que vamos a utilizar y profundizando en sus parámetros y opciones disponibles, podemos determinar que las columnas del dataset de comentarios serán las siguientes:

- **text**: Contenido textual original del comentario en YouTube. Se obtiene mediante la llamada `commentThreads.list` con `part=snippet` y `textFormat=plainText`, extrayendo `snippet.topLevelComment.snippet.textOriginal`.
- **author_id**: Identificador del canal del autor del comentario. Se obtiene de `commentThreads.list`, campo `snippet.topLevelComment.snippet.authorChannelId.value`.

- **author_name**: Nombre público del autor del comentario. Se obtiene de `commentThreads.list`, campo `snippet.topLevelComment.snippet.authorDisplayName`.
- **likes**: Número de “me gusta” del comentario. Se obtiene de `commentThreads.list`, campo `snippet.topLevelComment.snippet.likeCount`.
- **video_id**: Identificador único del video donde se publicó el comentario. Se obtiene mediante `search.list` con `q={keyword}`, extrayendo `items[] .id.videoId`, y se reutiliza en `videos.list` y `commentThreads.list`.
- **video_title**: Título del video. Se obtiene de `videos.list` con `part=snippet`, campo `snippet.title`.
- **channel_id**: Identificador del canal que publicó el video. Se obtiene de `videos.list`, campo `snippet.channelId`.
- **channel_title**: Nombre público del canal que publicó el video. Se obtiene de `videos.list`, campo `snippet.channelTitle`.
- **description**: Descripción textual del video. Se obtiene de `videos.list`, campo `snippet.description`.
- **category**: Categoría del video asignada por YouTube (ID numérico). Se obtiene de `videos.list`, campo `snippet.categoryId`.
- **topicCategories**: Conjunto de temáticas asociadas al video (URLs de ontologías externas). Se obtiene de `videos.list` con `part=topicDetails`, campo `topicDetails.topicCategories`.
- **label**: Palabra clave usada para recuperar el video y sus comentarios, se corresponde a las temáticas. No proviene de la API; corresponde al parámetro `q` empleado en `search.list`.

A la hora de decidir estos atributos se ha seguido la idea de ”cuánta más información, mejor”. Más adelante se verá que realmente la gran mayoría de estos valores no se utilizarán, pero merece la pena y supone relativamente poco esfuerzo incluirlos para tener opciones más adelante en el desarrollo.

En cuanto a la decisión de qué temáticas definir, lo más importante es que estas sean, por un lado, lo suficientemente distintas entre sí para minimizar colisiones y casos difusos (videos que encajen en más de una categoría), y por otro, lo suficientemente amplias y generales para permitir al modelo generalizar.

El conjunto final ha sido diseñado primero basándonos en las categorías que la propia API de YouTube define, y posteriormente refinando y eliminando mediante análisis cualitativo de conjuntos de comentarios recuperados por el script (cantidad de videos que se recuperan en más de una categoría, cantidad de videos que se corresponden

adecuadamente al *keyword* introducido...). Tras este proceso, nos quedamos con las siguientes categorías:

noticias
deportes
musica
tecnologia
cine
animales
ciencia
viajes
vehiculos

4.1.4. Script de recolección de comentarios

Finalmente, tras haber realizado el proceso de diseño, podemos pasar a la implementación de nuestro script.

Como puede verse en el script `scraper`, el proceso de recolección sigue la lógica de diseño descrita en los apartados anteriores. Solo queda comentar un detalle sobre el filtrado por idioma: el valor `defaultLanguage` inicialmente era utilizado para eliminar todo vídeo en el que dicho valor no coincidiese con "es". Sin embargo, YouTube clasifica ocasionalmente vídeos que no están en español con ese código erróneamente, además de utilizar códigos como "es-27" y similares para diferentes dialectos y variantes geográficas. Por lo tanto, se decidió omitir ese filtro y depender únicamente de `regionCode`. Más adelante se filtrarán aquellos comentarios que no estén en español con métodos más fiables.

4.2. Limpieza y procesado de datos

Tal y como vimos en la investigación inicial del estado del arte, la calidad de los datos es uno de los principales, si no el principal factor de éxito en las tareas de entrenamiento y ajuste fino de modelos de *Deep Learning*. Por lo tanto, merece la pena detenernos en este punto para implementar un *pipeline* de limpieza y procesamiento riguroso y completo para dejar los datos de la mejor manera posible antes de pasar al entrenamiento.

4.2.1. Limpieza mínima y preparativos para el procesamiento del dataset crudo

Antes de pasar a técnicas de filtrado propiamente dichas, podemos aplicar una serie de reglas básicas para eliminar comentarios que sean poco informativos de manera evidente y trivial.

En el script `limpieza` aplicamos una serie de filtros sencillos sobre la columna `text`:

- Normalizamos el contenido de los comentarios eliminando saltos de línea y tabulaciones, de manera que estos incluyan únicamente texto y espacios.
- Eliminamos comentarios cuyo contenido sea únicamente emoticonos.
- Eliminamos comentarios cuyo contenido sea únicamente enlaces.

También hacemos un poco de limpieza en las columnas `author_name` y `topicCategories` eliminando información redundante. En el caso de `author_name` los "@" al inicio de todos los nombres, y en `topicCategories`, que contiene una lista de enlaces a artículos de Wikipedia, nos quedamos sólo con el nombre del artículo, que es el tema relevante, y eliminamos el resto del enlace (que es igual siempre).

Una vez dado este paso previo, es necesario extraer los tokens de cada uno de los comentarios para poder pasar a filtrados más específicos. El beneficio de aplicar estas técnicas sobre los tokens en vez de sobre el contenido textual del comentario directamente reside en que al tokenizar un texto obtenemos una representación estructurada, normalizada y comparable del contenido real del comentario, reduciendo "ruido lingüístico". Esto permite aplicar las técnicas sobre unidades consistentes, dando lugar a métricas estadísticas más fiables y por lo tanto un mejor filtrado [4].

En el script `tokenizar` cargamos nuestro dataset tras la primera limpieza básica para generar los tokens. Para ello usamos la biblioteca NLTK (*Natural Language Toolkit*), diseñada precisamente para tareas NLP. Esta nos ofrece implementaciones de tokenización y muchas otras herramientas, además de ser fiable y madura, mantenida desde 2001 y utilizada de forma estandarizada en ámbitos académicos [21]. Además de tokenizar los textos aprovechamos para eliminar *stopwords*.

Cabe mencionar que el tokenizador que estamos utilizando en este paso no es el mismo que va a aplicar nuestro *Transformer*, que utilizará su propio tokenizador especializado sobre el contenido textual (columna `text`). Estos tokens que estamos extrayendo con NLTK únicamente se usan para mejorar el *pipeline* de filtrado de comentarios, eliminando filas completas (comentarios) que puedan introducir ruido al *Transformer*. Por lo tanto, no estamos afectando al contenido de los comentarios en ninguno de estos pasos y dejamos que el propio tokenizador del modelo los segmente de la manera óptima para ese modelo específico.

La figura 4.2 muestra una representación visual de los tokens más frecuentes en el dataset. Esta es parte del informe generado por el script `eda_report`. Este script genera un informe exploratorio de datos (EDA) en formato HTML mediante la biblioteca `ydata-profiling`, antes conocida como `pandas-profiling`. La biblioteca proporciona una multitud de herramientas de generación de estadísticas y visualización de datos [22]. En el caso concreto de nuestro script, simplemente genera estadísticas descriptivas y distribuciones para cada una de las columnas, además de detectar correlaciones e interacciones entre variables. Cabe destacar que este script ha resultado ser *extremadamente* útil en todas las fases del desarrollo, como herramienta rápida y ligera

para "echar un vistazo rápido" a los datos en cada paso del desarrollo, permitiendo detecciones tempranas de problemas y errores en los filtros.



Figura 4.2: Visualización *Bag of Words* de los tokens más frecuentes en el dataset

Por último, antes de pasar al filtrado podemos aplicar una técnica sencilla de visualización a partir de los tokens generados, con el objetivo de comparar esta imagen del dataset antes y después del procesado completo. Esta técnica se denomina *Principal Component Analysis* (PCA). El método PCA reduce la dimensionalidad de un conjunto de datos con muchas variables, transformando las variables originales (que pueden estar correlacionadas) en unas pocas variables no correlacionadas llamadas componentes principales, estando estos componentes ordenados de tal modo que el primero explica la mayor parte de la varianza posible, el segundo explica la mayor parte de la varianza restante, y así sucesivamente [23]. De esta forma, en vez de tener muchas dimensiones (una por variable) nos quedamos con unos pocos componentes principales que capturan la mayor parte de la información.

En nuestro caso reducimos a dos dimensiones para visualizar de manera sencilla y fácilmente interpretable nuestro dataset. El script `embeddings` genera los *embeddings* a partir del texto de los comentarios, y estos son procesados en el script `pca` para ser reducidos a dos componentes PC1 y PC2 con los que generaremos la figura 4.3.

Una decisión que cabe mencionar en este paso es la de qué biblioteca utilizar para generar los *embeddings*. Hemos optado por utilizar `distiluse-base-multilingual-cased-v2` del paquete `SentenceTransformers` de *Hugging Face*, un modelo diseñado específicamente para oraciones completas como lo son comentarios de YouTube. En gran parte la decisión de utilizar este modelo se debe a que está basado en la arquitectura *DistilBERT*, exactamente la misma que el modelo que entrenaremos más adelante [24].

La visualización PCA 2D (Figura 4.3) muestra que los comentarios presentan una estructura densa y continua, pero con un solapamiento casi total entre las diferentes categorías temáticas. Es evidente la ausencia de *clusters* definidos, lo que indica que la

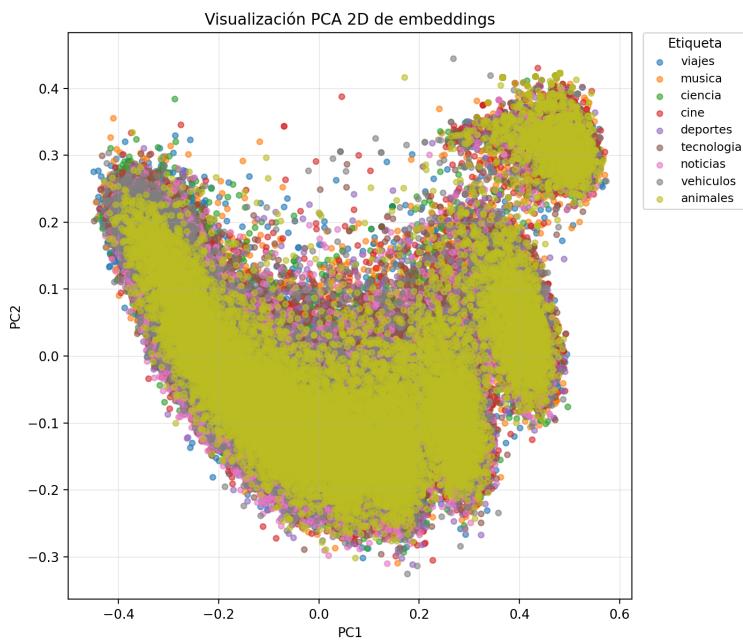


Figura 4.3: Visualización PCA 2D del dataset previo al filtrado

similitud semántica capturada por los *embeddings* en un espacio de baja dimensionalidad no es suficiente para segregar las clases de forma lineal. Las regiones periféricas más dispersas confirman la presencia de comentarios atípicos o ruidosos que actúan como *outliers*. Esta mezcla indiferenciada de puntos es un indicador temprano de la complejidad de la tarea, sugiriendo que las fronteras entre temas en el lenguaje informal de YouTube son sumamente difusas y no dependen de una separación geométrica simple, sino de relaciones contextuales más profundas.

4.2.2. Filtrado básico mediante reglas

El primer tipo de filtrado que podemos aplicar es sencillo pero potente; reglas basadas en características sencillas de los textos a clasificar como pueden ser la longitud o el tipo de símbolos que aparecen. En el script `filtro_reglas` aplicamos las siguientes reglas de filtrado sobre los comentarios:

- **Filtro de longitud:** Eliminamos aquellos comentarios que sean demasiado cortos o demasiado largos. Los comentarios muy breves suelen ser poco informativos (por ejemplo "ok", "lol"). Por el contrario, los comentarios demasiado grandes tienden a mezclar temas, introducir información irrelevante o diluir las señales discriminativas, por lo que conviene eliminar también aquellos que estén muy por encima de la longitud media. Para este paso interpretamos la longitud no como cantidad de caracteres en la columna `text`, sino como la cantidad de tokens en la columna `tokens`, puesto que esa es la unidad de significado semántico.

- **Filtro de ruido simbólico:** Los comentarios con muchos caracteres no alfabéticos (símbolos, emojis, URLs) suelen aportar poca información semántica útil, pudiendo distorsionar la tokenización. Si detectamos comentarios con una alta proporción de estos caracteres (más de un 30%), lo eliminamos. Cabe destacar que sí permitimos ciertos caracteres no alfabéticos muy comúnmente usados, como números, guiones y apóstrofes, con tal de no ser demasiado estrictos.
- **Filtro de idioma:** Anteriormente se mencionó que la información interna de la API de YouTube no es del todo fiable a la hora de filtrar comentarios por idioma. Por ello, en este paso utilizamos la biblioteca `langid` para detectar el idioma de cada comentario y sólo quedarnos con aquellos detectados como español. La herramienta detecta el idioma de un texto usando un clasificador *Naive Bayes* entrenado sobre n-gramas de 97 idiomas distintos [25].

4.2.3. Filtrado avanzado mediante métricas estadísticas de calidad

Los filtros basados en reglas sencillas capturan propiedades superficiales del texto (longitud, idioma...), pero no su calidad informativa real. Calculando métricas estadísticas con un fundamento teórico fuerte podemos detectar textos repetitivos, degenerados o cercanos al ruido que sí pasan esos filtros básicos, pero que afectan negativamente al entrenamiento del modelo, ya que no aportan información semántica útil.

En el script `filtro_estadistico.py` calculamos tres métricas sobre el dataset (en su estado posterior al filtrado básico), y filtramos en base a umbrales definidos para cada métrica.

Entropía de Shannon

La entropía de Shannon es una medida de la incertidumbre promedio asociada a una variable aleatoria discreta. En el contexto de un texto, dicha variable representa la distribución de probabilidad de los símbolos que lo componen. La entropía cuantifica cuánta información, en promedio, se obtiene al observar un símbolo generado por esa fuente. Sea X una variable aleatoria discreta que puede tomar valores x_1, x_2, \dots, x_n con probabilidades $P(X = x_i) = p_i$. La entropía de Shannon se define como:

$$H(X) = - \sum_{i=1}^n p_i \log_2 p_i$$

En el caso de un texto, las probabilidades p_i se estiman empíricamente a partir de la frecuencia relativa de cada símbolo en el comentario [26].

Intuitivamente hablando, la entropía mide cómo de predecible o variado es un texto. En el contexto de comentarios en vídeos de YouTube, aquellos con entropía muy baja suelen estar dominados por repeticiones (por ejemplo, spam, cadenas de un solo carácter o plantillas), mientras que valores extremadamente altos suelen indicar ruido, texto

corrupto o secuencias casi aleatorias. Por ello, la entropía es útil para filtrar comentarios que, aún teniendo una longitud adecuada y un idioma válido, carecen de estructura lingüística informativa.

En el script `compute_entropy` calculamos la entropía para cada comentario, usando los tokens como unidad simbólica e implementando directamente la definición matemática clásica. El script `eda_report` previamente introducido nos permite visualizar la distribución de la entropía sobre el dataset de comentarios completo (figura 4.4)

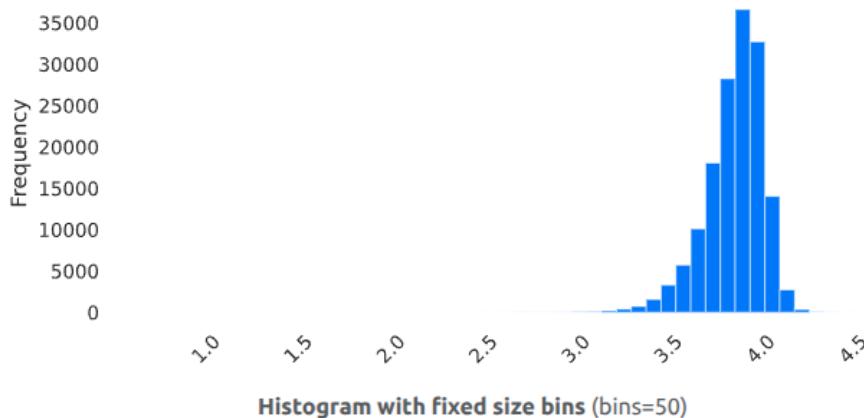


Figura 4.4: Distribución de la entropía de Shannon del dataset de comentarios

Diversidad de n-gramas

La diversidad de n-gramas es una medida que cuantifica la riqueza léxica y estructural de un texto evaluando la proporción de n-gramas distintos que aparecen en él. Un n-grama se define como una secuencia contigua de n símbolos, y la diversidad mide cuántos de esos n-gramas son únicos respecto al total generado por el texto.

Sea un texto T compuesto por una secuencia de tokens, y sea $G_n(T)$ el conjunto de todos los n-gramas extraídos de T , la diversidad de n-gramas se define como:

$$D_n(T) = \frac{|\text{unique}(G_n(T))|}{|G_n(T)|}$$

donde $|G_n(T)|$ es el número total de n-gramas del texto y $|\text{unique}(G_n(T))|$ es el número de n-gramas distintos [5].

Esta métrica mide hasta qué punto un texto reutiliza las mismas secuencias frente a generar combinaciones variadas. Comentarios con diversidad de n-gramas muy baja suelen corresponder a textos repetitivos, plantillas, spam o mensajes generados automáticamente, mientras que valores más altos indican mayor variabilidad lingüística. Por ello, la diversidad de n-gramas permite detectar comentarios que, aun siendo largos y

con entropía aceptable, no aportan contenido informativo real.

En el contexto de nuestra tarea, nos interesa fijar n a 1. Esto fija la métrica en el nivel léxico elemental, ya que con unigramas, la diversidad depende solo de cuántos tokens distintos aparecen, no de su orden. De esta manera eliminamos comentarios con vocabulario pobre (que reutiliza constantemente las mismas palabras).

En el script `compute_ngram` calculamos la diversidad de unigramas de cada comentario, que a nivel de código resulta sumamente sencillo al haber fijado $n = 1$. Visualizamos la distribución de la métrica mediante el script `eda_report` (figura 4.5)

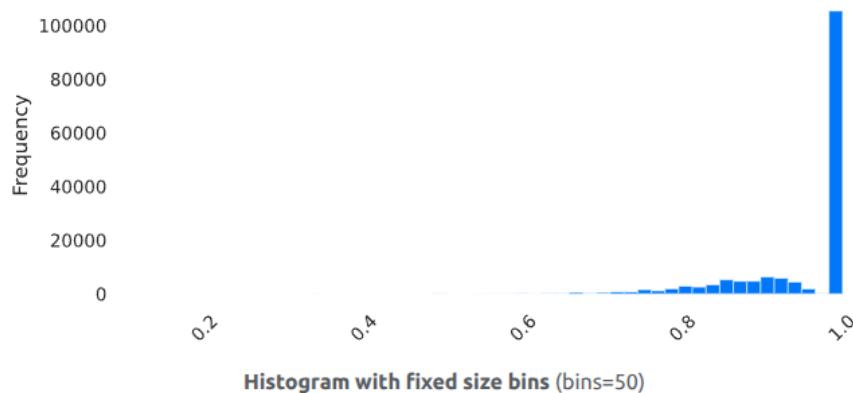


Figura 4.5: Distribución de la diversidad de unigramas del dataset de comentarios

Valor *Perplexity*

La *perplexity* es una medida que evalúa qué tan bien un modelo probabilístico de lenguaje predice una secuencia de texto. Formalmente, se define como la exponencial negativa de la media del logaritmo de la probabilidad asignada por el modelo a cada símbolo de la secuencia. Un valor menor de *perplexity* indica que el modelo considera el texto más predecible. Sea una secuencia de tokens $W = (w_1, w_2, \dots, w_N)$ y un modelo de lenguaje que asigna una probabilidad conjunta $P(W)$, la *perplexity* se define como:

$$\text{PP}(W) = P(W)^{-\frac{1}{N}} = \exp \left(-\frac{1}{N} \sum_{i=1}^N \log P(w_i | w_1, \dots, w_{i-1}) \right)$$

Esta formulación es equivalente a la exponencial de la entropía cruzada entre la distribución empírica del texto y el modelo de lenguaje [5].

La idea intuitiva de esta métrica es medir el grado de sorpresa que un texto produce en un modelo de lenguaje entrenado sobre datos lingüísticamente correctos. Comentarios con *perplexity* muy alta suelen corresponder a texto ruidoso, mal formado o aleatorio, mientras que valores extremadamente bajos pueden indicar repeticiones triviales

o secuencias artificialmente simples. Esta es por ello una métrica eficaz para filtrar comentarios que, aunque pasen filtros básicos de longitud o idioma, no se ajustan a patrones lingüísticos naturales.

En el script `compute_pp1` calculamos el valor de *perplexity* de cada comentario. Validamos y tokenizamos el texto del comentario, se le pasa al modelo usando los propios `input_ids` como etiquetas, lo que permite que el modelo calcule internamente la pérdida de entropía cruzada media entre la secuencia real y la predicha. La *perplexity* se obtiene como la exponencial de dicha pérdida. El modelo utilizado para esta tarea es GPT-2, ya que es un modelo de lenguaje causal entrenado explícitamente para predecir el siguiente token.

Visualizamos la distribución de la métrica mediante el script `eda_report` (tabla 4.1). Aquí tenemos una distribución extremadamente asimétrica, muy fuertemente sesgada hacia la derecha (la mayoría de comentarios tienen valores relativamente bajos). Esto indica que una fracción muy pequeña del dataset contiene el texto que el modelo GPT-2 considera altamente improbable. El sesgo hace que la visualización de la distribución sea muy poco informativa, por lo que en su lugar incluimos una tabla con unos pocos datos estadísticos que explican la distribución.

Tabla 4.1: Distribución de la *perplexity* del dataset de comentarios

Métrica	Valor
Valores distintos	98.2%
Mínimo	2.0203142
Mediana	318.22244
Media	714.0681
Máximo	4 308 129
Desviación estándar	11 204.089
Asimetría (skewness)	366.9444

Umbrales elegidos para cada métrica

Una vez calculadas las columnas con los valores de cada métrica, es sencillo ordenar el dataset completo de mayor a menor valor para cada una (con un script de Python sencillo o, en nuestro caso, usando herramientas de visualización de ficheros CSV como LibreOffice Calc). Los valores de los percentiles 25 y 75 nos sirven como una orientación inicial, y tras examinar los comentarios en torno a esos valores elegimos los umbrales concretos donde observemos un cambio significativo en la calidad del comentario.

En el caso de la entropía fijamos como umbral mínimo 3.6 y como máximo 4.38. En el caso de la diversidad de unigramas, decidimos no fijar un umbral máximo en absoluto, ya que el valor máximo posible, 1, constituye el 68% del dataset, además

de tratarse de comentarios en los que no se repite ningún token, lo cual es ideal. El umbral mínimo lo fijamos en 0.4, puesto que comentarios con valores inferiores presentan grandes proporciones de una misma palabra repetida (por ejemplo repeticiones de las palabras, "si", "gracias", o "ja"). En el caso de *perplexity*, se ha optado por ser permisivo en el umbral inferior, pidiendo sólo un valor mínimo de 10, ya que la mayoría de comentarios de mala calidad con valores bajos de *perplexity* son aquellos que ya se van a eliminar por tener valores no deseables de entropía o diversidad de unigramas. El límite superior se establece en 1000; es a partir de este valor cuando encontramos comentarios que, o bien no tienen sentido, o contienen una cantidad muy elevada de faltas ortográficas y gramaticales.

4.2.4. Reducción, balanceado y división del dataset

A pesar de que partimos de un dataset balanceado, el proceso de filtrado descrito en este capítulo puede dar lugar a desequilibrios importantes entre las clases, ya que las técnicas y métricas aplicadas no se basan en nada más que el contenido textual del comentario, ignorando por completo el resto de campos del dataset.

El script `minimize` reduce el número de instancias por clase a un número objetivo, asegurando dos propiedades: cada clase tiene como máximo el mismo número de instancias, y no se rompen grupos lógicos de datos asociados a un mismo `video_id` (es decir, eliminamos todos los comentarios de un mismo vídeo, no comentarios sueltos).

Tras aplicar la reducción, tenemos la versión final del dataset que vamos a utilizar para entrenar nuestro modelo. Recalculamos los *embeddings* y aplicamos PCA. La figura 4.6 muestra el resultado.

La distribución resultante (Figura 4.6) se vuelve notablemente más compacta y homogénea, logrando eliminar con éxito la mayoría de los *outliers* y regiones dominadas por ruido lingüístico. Sin embargo, la visualización revela que, a pesar de haber saneado el dataset, el solapamiento entre etiquetas persiste de forma dominante. Esto demuestra que el filtrado mejora la calidad estadística de los datos, pero no crea por sí solo fronteras de decisión lineales. Es precisamente esta incapacidad de los *embeddings* básicos para separar las temáticas en el plano lo que justifica la necesidad de utilizar un modelo *Transformer*. Al no existir una separación clara en el espacio vectorial, el éxito de la clasificación dependerá de la capacidad del modelo para realizar un procesamiento semántico no lineal y contextual mediante mecanismos de atención, capturando patrones sutiles que las proyecciones geométricas de baja dimensionalidad son incapaces de resolver.

Por último, y antes de pasar al diseño de los *pipelines* de entrenamiento, optamos por realizar la división del dataset en un script dedicado a ello, `split`. De esta manera podemos saber con exactitud las características de los splits, si estos están balanceados en sí mismos, y asegurarnos de que todos los *pipelines* en todas sus versiones reciben exactamente los mismos datos, útil para hacer comparaciones y elegir los mejores hiperparámetros.

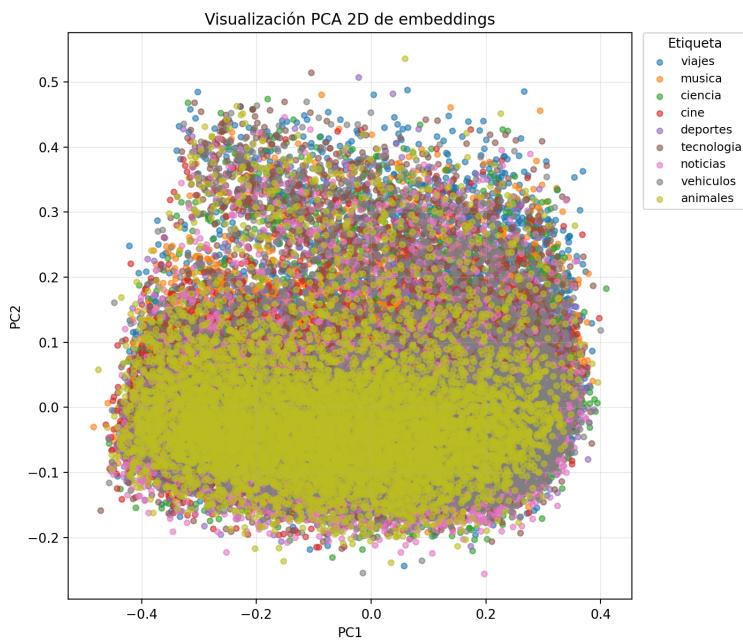


Figura 4.6: Visualización PCA 2D del dataset tras el *pipeline* de procesado completo

En cuanto al split utilizado, dividimos en split de entrenamiento, split de validación y split de test con una proporción 70/15/15. Estas proporciones son ampliamente utilizadas y además son las que nos han dado lugar a mejores resultados en la práctica.

Lo más destacable de la implementación del script es que nos aseguramos de que todos los comentarios de un mismo vídeo entren en un mismo split (agrupando por `video_id`). Es decir, no habrán comentarios de un mismo vídeo en más de uno de los splits. Esto evita fugas de información (*data leaking*) entre entrenamiento, evaluación y test, garantizando que el modelo se evalúa sobre contextos no observados previamente. Esta decisión impide que similitudes temáticas o léxicas específicas de un mismo vídeo influyan artificialmente en las métricas de rendimiento, proporcionando una estimación más realista de la capacidad de generalización del modelo.

4.2.5. Características del dataset final

Una vez completado el procesado y particionado de los datos, procedemos a validar el resultado antes de pasar a implementar los modelos que utilizarán el dataset. La tabla 4.2 muestra la cantidad total de comentarios en cada paso del procesado y la proporción de las clases en ese momento del *pipeline*, a modo ilustrativo del impacto de cada filtro en la versión final. La numeración de los pasos del *pipeline* de procesado se corresponde a los filtros descritos en las secciones anteriores:

1. Dataset crudo
2. Limpieza inicial
3. Filtro de longitud
4. Filtro de ruido simbólico
5. Filtro de idioma
6. Filtros estadísticos
7. Dataset final (reducido y balanceado)

También se han numerado las etiquetas para una mejor visualización de la tabla, del 1 al 9 siguiendo el orden Noticias, Deportes, Música, Tecnología, Cine, Animales, Ciencia, Viajes y Vehículos.

Tabla 4.2: Resultados del pipeline de procesado

Paso	Total	% et1	% et2	% et3	% et4	% et5	% et6	% et7	% et8	% et9
1	256383	8.4	9.4	15	8.8	12.7	7.9	14.1	15.8	8
2	249818	8.5	9.5	14.7	8.9	12.7	7.4	14.4	15.9	8.1
3	211658	9	9.7	12.9	8.9	12.3	6.5	15.3	16.8	8.5
4	182889	8.9	9.8	13.2	9	12.5	6.7	15	16.4	8.6
5	155547	9.9	10.3	13	7.7	12.3	6.5	16.1	16.6	7.6
6	127555	10.6	10.7	11.9	7.5	11.7	6	16.3	17.4	7.8
7	71423	11.2	11.1	11.1	11.1	11.2	10.7	11.2	11.2	11.2

También se ha implementado el script `insights` para generar una serie de estadísticas para cada una de las clases, basándonos en los metadatos disponibles en el dataset. Las tablas 4.3 y 4.4 muestran los valores extraídos.

Clase	Long. media (chars)	Long. media (tokens)	Diversidad 1-gram	Entropía media	Perplexity media
Animales	85.12	9.07	0.9639	3.8617	360.60
Ciencia	120.61	13.35	0.9460	3.8849	338.27
Cine	97.26	10.76	0.9579	3.8759	357.00
Deportes	111.17	12.27	0.9496	3.8999	326.98
Música	90.55	10.12	0.9572	3.8609	375.53
Noticias	131.18	13.91	0.9421	3.9082	299.60
Tecnología	114.05	12.54	0.9511	3.8910	337.63
Vehículos	132.49	14.91	0.9415	3.9115	342.45
Viajes	124.75	14.02	0.9406	3.8969	314.72

Tabla 4.3: Longitud y estadísticas medias por clase del dataset

La tabla 4.3 muestra algunas diferencias interesantes en la retórica de los usuarios según la temática del vídeo. Las categorías de Vehículos (132.49) y Noticias (131.18)

presentan la mayor longitud media en caracteres, lo que sugiere que estas temáticas dan lugar a comentarios más descriptivos o informativos, mientras que la clase Animales (85.12), cuenta con la menor media de tokens (9.07). En cuanto a la predictibilidad del lenguaje, la *perplexity* media más baja se encuentra en Noticias (299.60), indicando que los comentarios en esta categoría siguen estructuras gramaticales más predecibles y cercanas al español estándar. Por el contrario, la clase Música presenta la *perplexity* más alta (375.53), lo que refleja un lenguaje más informal y posiblemente cargado de expresiones propias de las comunidades de fans de artistas específicos. Finalmente, la notable estabilidad de la entropía (situada entre 3.86 y 3.91 en todas las clases) valida la eficacia del *pipeline* de filtrado, garantizando que todas las clases tengan un nivel de información aportada homogéneo en el dataset final.

Clase	Canales únicos	Autores únicos	N (filas)
Animales	144	7291	7629
Ciencia	71	7696	7995
Cine	155	7719	7982
Deportes	107	7452	7958
Música	192	7652	7959
Noticias	72	7031	7968
Tecnología	170	7744	7957
Vehículos	175	7508	7993
Viajes	129	7695	7982

Tabla 4.4: Canales/autores únicos y número de filas por clase del dataset

La tabla 4.4 nos permite confirmar la representatividad del dataset final: el número de autores únicos es muy cercano al total de filas en todas las categorías, lo que garantiza una variedad de opiniones y asegura que el modelo no ha sido entrenado con "spam" o sesgado por usuarios recurrentes. Respecto a la procedencia de los datos, la distribución de canales únicos muestra que el contenido de Ciencia (71) y Noticias (72) está más centralizado en grandes medios o divulgadores, mientras que Música (192) y Vehículos (175) presentan un mayor número de fuentes. Esto puede significar que el modelo ha estado expuesto a una mayor variedad de estilos y temáticas concretas en estas últimas clases. Por último, las cifras de la columna N (filas) confirman que el script de minimización logra un balanceo casi perfecto entre las clases, esencial para obtener métricas F1 macro fiables.

4.3. Implementación de modelo *baseline*: TF-IDF + Regresión Logística

Ahora que tenemos la versión final de nuestro dataset de comentarios podemos pasar al entrenamiento del modelo de clasificación. Pero antes de lanzarnos a trabajar con una arquitectura compleja y computacionalmente costosa como lo son los *Transformers*, es conveniente establecer un punto de referencia sencillo y fácilmente interpretable para evaluar si el modelo avanzado aporta una mejora real y justificada.

4.3.1. Modelo *baseline* basado en TF-IDF y Regresión Logística

El *baseline* elegido ha sido el siguiente: aplicar TF-IDF para obtener una representación numérica del texto, y usar regresión logística como clasificador lineal sobre esta representación. Esta combinación es un baseline simple, eficiente y clásico en tareas de clasificación de texto, tal y como vimos en el segundo capítulo.

Una vez definida la técnica de representación de los comentarios y el algoritmo de clasificación utilizado, podemos pasar a la implementación del modelo. El script `regression` implementa el *pipeline* completo, desde el entrenamiento hasta la evaluación y generación de métricas e información sobre el proceso, todo a través del paquete `scikit-learn`, que ofrece las herramientas necesarias.

En el script, la representación de los textos en vectores TF-IDF es realizada por el objeto `TfidfVectorizer`, que se construye pasándole una serie de parámetros. Los parámetros con los que construir el vectorizador dependen de la tarea específica y las características de los datos a vectorizar. Los valores exactos se han elegido combinando razonamiento basado en el contexto de este proyecto y simple prueba y error para obtener los mejores resultados. Resumidamente:

- `ngram_range=(1,2)` hace que se incluyan como características tanto palabras (unigramas), que capturan contenido léxico básico, como pares de palabras (bigramas), que capturan expresiones fijas y patrones cortos. Estas palabras y sueltas y "mini-expresiones" son muy típicas de cómo se suelen escribir los comentarios.
- `min_df=2` significa que eliminamos los términos que aparezcan en menos de dos comentarios, es decir, aquellos términos extremadamente raros y que aparecen en sólo un comentario. Si sólo se menciona una sola vez entre tantos comentarios relacionados al mismo vídeo o tema, es casi imposible que sea útil para aprender patrones o generalizar.
- Por el contrario con `max_df=0.9`, eliminamos los términos que aparecen en más de un 90% de los comentarios. Estos ofrecen un nulo poder discriminativo, añadiendo información que no sirve para discernir pertenencia a ninguna de las clases y, por lo tanto, son ruido.

- Con `sublinear_tf=True` en vez de usar la frecuencia tf directamente, usamos la versión sublinear $tf' = 1 + \log(tf)$ cuando $tf > 0$. Esta manera de tratar el valor evita que palabras que se repiten mucho dominen en exceso el vector solo por aparecer varias veces, puesto que repetir una palabra 20 veces en un comentario no aporta 20 veces más información. La TF sublineal suele casi siempre mejorar el rendimiento sobre la TF "cruda" [7].

El algoritmo de regresión también cuenta con su serie de parámetros en el constructor del objeto `LogisticRegression`. Las decisiones más importantes en cuanto al diseño del entrenamiento son las siguientes:

- **Regularización:** El parámetro `penalty=12` aplica regularización L2. Esta es la forma de regularización estándar en modelos lineales como la regresión logística. El parámetro `C=1.0` implica que se utiliza el inverso del valor de regularización, lo que se traduce en una regularización más moderada.
- **Algoritmo:** Existen múltiples algoritmos de regresión logística siendo `lbfgs` uno de los solvers de `scikit-learn` que optimiza correctamente la formulación multinomial mediante softmax, imprescindible cuando se clasifican textos en múltiples categorías. Además, es un algoritmo eficiente y estable para trabajar en espacios de alta dimensionalidad como en los que están los vectores TF-IDF [27]. Precisamente debido a que TF-IDF produce un espacio de gran dimensión, fijamos el límite de iteraciones para que el solver converja bastante alto (`max_iter=1000`), para evitar que el entrenamiento se detenga antes de tiempo.

Con eso ya tendríamos los dos componentes principales del entrenamiento definidos. `Pipeline` de `scikit-learn` nos permite encadenar ambas etapas, definiendo así el modelo *baseline*. Por último, el script guarda en formato CSV las predicciones que hace sobre cada comentario del conjunto de test junto a la clase real a la que pertenece el comentario. Esta información nos será útil en el capítulo de análisis, donde estudiaremos los resultados del modelo y sus métricas, además de compararlo con el modelo *Transformer* ajustado.

4.3.2. Modelo zero-shot basado en *Transformers*

Otra opción de modelo *baseline* que vale la pena considerar es la de un modelo basado en la clasificación *zero-shot*.

El script `zeroshot` implementa el modelo usando el *pipeline* específico de Hugging Face para clasificación *zero-shot*.

El modelo de elección fue `mDeBERTa-v3-base-mnli-xnli`, una versión multilingüe de DeBERTa v3 (basada en la arquitectura BERT, al igual que el modelo que ajustaremos en la siguiente sección) afinada específicamente para *Natural Language Inference* (NLI). Este tipo de modelos es el más adecuado para nuestra tarea de clasificar comentarios

escritos en lenguaje natural, ya que el *pipeline* convierte cada etiqueta en una hipótesis del tipo “este texto trata sobre X” y el modelo predice si el comentario implica dicha hipótesis.

En cuanto a la configuración del pipeline, destacar que elegimos `multi_label=False` (cada comentario debe pertenecer a una única categoría temática) y que se adopta un tamaño de lote `BATCH_SIZE=1` para evitar problemas de memoria y asegurar compatibilidad total con la implementación interna del pipeline, que realiza múltiples inferencias por entrada. La selección de la clase predicha se realiza mediante `argmax` (escoger la clase con mayor probabilidad asignada) sobre las puntuaciones devueltas, tal como recomienda la propia documentación del *pipeline zero-shot* [28].

4.4. Entrenamiento supervisado de modelo *Transformer* basado en DistilBERT

Tal y como comentamos en el tercer capítulo, trabajaremos con el modelo DistilBERT basado en la arquitectura BERT. En concreto, ajustaremos el modelo pre-entrenado `distilbert-base-multilingual-cased` ofrecido a través de la biblioteca `transformers` de *Hugging Face*.

4.4.1. Características del modelo a ajustar

La decisión de qué modelo pre-entrenado ajustar a nuestra tarea es clave, ya que en gran parte el éxito del trabajo dependerá de este. Utilizaremos un modelo basado en BERT, y dentro de este espacio de opciones, el modelo más adecuado para nuestras capacidades y objetivos es DistilBERT.

DistilBERT es un modelo de lenguaje basado en *Transformers* que se obtiene a partir de BERT base aplicando *“knowledge distillation”*. El trabajo original fue desarrollado por investigadores de *Hugging Face*, que mostraron que es posible reducir el tamaño de BERT base en torno a un 40% manteniendo aproximadamente un 95-97% de rendimiento en una serie de tareas de NLP. Esta reducción de tamaño se traduce en una aceleración del 60% en la inferencia [29].

La técnica antes mencionada, *knowledge distillation*, consiste en entrenar un modelo más pequeño, denominado *student*, para que imite las representaciones y salidas del modelo BERT original, al que se le denomina *teacher*. Esta fase de transferencia de conocimiento entre maestro y alumno es un preentrenamiento que se realiza previamente a un entrenamiento completo ya de manera independiente. El preentrenamiento se realiza con una función de pérdida compuesta que combina pérdida del modelado de lenguaje, pérdida de *distillation* (ajuste a las *logits* del modelo maestro) y un término de distancia de coseno entre las representaciones internas del maestro y el alumno. De esta manera, DistilBERT hereda gran parte de la capacidad de representación de BERT pero con un coste computacional muy reducido. Además, DistilBERT elimina algunas

capas del *encoder* y simplifica ciertos componentes, pero manteniendo la estructura general de BERT [29].

Esto resulta en un modelo con una exigencia computacional mucho más asequible sin por ello perder de manera significativa en rendimiento. Es el modelo ideal para nuestro caso en el que se cuenta con una GPU de gama media integrada en portátil.

4.4.2. Diseño e implementación del *pipeline* de entrenamiento

A continuación entraremos en detalle sobre todas las decisiones tomadas en el diseño del *pipeline* de entrenamiento, y que afectan de manera significativa a los resultados del mismo. El script `train` contiene la implementación del *pipeline*.

Preparación de datos y etiquetado

Al igual que hacíamos con el modelo *baseline*, cargamos los splits de entrenamiento, validación y test como ficheros CSV separados. Aplicamos un último filtrado mínimo sobre estos, eliminando espacios en blanco y descartando filas con texto vacío. Este filtro no debería cambiar nada al aplicarse sobre un dataset correctamente procesado, pero sirve para asegurarnos por completo una última vez de la validez de los textos antes de ser tokenizados.

Se construyen los mapas `label2id` y `id2label`, que implementan la codificación de las etiquetas categóricas en números enteros, como es habitual en el entrenamiento de modelos de redes neuronales. El script utiliza `AutoConfig.from_pretrained` para cargar la configuración base del modelo e inyecta el número de clases y los dos mapas, lo que permitirá que las métricas y salidas del modelo puedan referirse a las clases por nombre, generando así resultados fácilmente legibles e interpretables.

La tokenización se realiza con `AutoTokenizer.from_pretrained` usando el propio modelo DistilBERT. Aplicamos truncamiento con una longitud máxima establecida en 256.

También aplicamos *padding* con un *data collator* dinámico (`DataCollatorWithPadding`). El *padding* dinámico por *batch* optimiza el uso de memoria al ajustarse a la secuencia más larga de cada *batch* en vez de llenar todo al máximo global. Esto junto al truncamiento antes mencionado nos ayudará a reducir memoria y tiempo de cómputo, crucial en un entorno de recursos limitados como en el que se desarrolla este trabajo.

Finalmente, envolvemos los datos en objetos `Dataset` y `DatasetDict`, sustituyendo la etiqueta numérica a una columna `labels` que es la esperada por el `Trainer`.

Función de pérdida y pesos de clase

Utilizamos la función de pérdida estándar en problemas multiclase, *Cross Entropy Loss*. Resumidamente, esta maximiza la probabilidad logarítmica de la clase correcta.

Se define también una subclase `WeightedTrainer` que permite incorporar pesos por clase en la función de pérdida si se activa el flag `--use_class_weights`. Los pesos se

calculan como inversamente proporcionales a la frecuencia de cada clase en el conjunto de entrenamiento, normalizados, de modo que las clases minoritarias reciben un peso mayor en la pérdida, mitigando el efecto de desbalance de clases. Se penalizan más los errores en clases raras para que el modelo no se limite a optimizar la precisión global favoreciendo las clases mayoritarias [8]. Esta técnica se utiliza por lo tanto en casos de desequilibrio importante en el dataset; en teoría no es nuestro caso, pero merece la pena probar y comparar los resultados del entrenamiento con y sin esta técnica.

Hiperparámetros de entrenamiento

El script configura el objeto `TrainingArguments` con los valores de los hiperparámetros. Estos son de suma importancia a la hora de controlar la optimización y resultados del proceso de entrenamiento, y por lo tanto tienen que ser elegidos con detenimiento. Además, también se ha tenido que hacer un poco de prueba y error para determinar los valores óptimos. Las decisiones más relevantes son las siguientes:

- **Estrategia de evaluación, guardado y logging:** Evaluamos y guardamos el estado actual del modelo al final de cada época, lo cual es apropiado cuando las épocas no son extremadamente largas. `logging_steps` controla cada cuántos pasos del entrenamiento registramos métricas intermedias, en nuestro caso el valor de 100 indica que para cada época se registra 100 veces el valor de la función de pérdida y la tasa de aprendizaje 100 veces por época. Esto nos permite una visión bastante fina de la evolución del proceso de entrenamiento mientras se ejecuta.
- **Selección y carga del mejor modelo:** Configuramos el `Trainer` para que, tras el entrenamiento, recargue el checkpoint que obtuvo el mejor valor de F1 macro en validación. F1 macro promedia el F1 por clase otorgando igual peso a cada una, haciendo la métrica más justa a la hora de evaluar resultados. Al optimizar sobre F1 macro en validación y usarlo como criterio para elegir el mejor modelo, el `pipeline` fuerza al modelo a repartir su capacidad entre todas las clases en lugar de favorecer únicamente las más frecuentes. Esto actúa como un mecanismo indirecto contra el sobreajuste a patrones de frecuencia del conjunto de entrenamiento.
- **Tamaño de batch y acumulación de gradientes:** Los tamaños de batch determinan cuántos ejemplos se procesan simultáneamente en cada paso de entrenamiento o evaluación. Usamos un batch de 16 en el entrenamiento, un tamaño apto para GPUs de capacidad media, equilibrando estabilidad de gradiente y uso de memoria. En evaluación podemos permitirnos usar un batch mayor de 32 al no requerirse *backpropagation*. Por otro lado, la acumulación de gradientes en 1 implica que se actualizan los pesos en cada batch, sin simular batches más grandes, lo que simplifica el esquema de optimización.

- **Número de épocas y *early stopping*:** Se fija un máximo de 10 épocas de entrenamiento, aunque este número es más bien un límite superior, ya que aplicamos *early stopping*: Se añade un `EarlyStoppingCallback` con paciencia 4, de forma que, si la métrica objetivo (F1 macro) no mejora durante cuatro evaluaciones consecutivas, el entrenamiento se detiene antes de alcanzar el máximo de época, evitando que el modelo se adapte en exceso al conjunto de entrenamiento (sobreajuste).
- **Regularización L2:** `weight decay` implementa regularización L2 sobre los pesos del modelo, añadiendo un término que penaliza normas grandes de los parámetros y reduciendo así la complejidad efectiva del modelo. El valor 0.005 representa una regularización moderada, ya que buscamos mejorar la capacidad de generalización pero sin impedir que el modelo se adapte al dataset.
- **Tasa de aprendizaje y scheduler con *warmup*:** La tasa de aprendizaje controla el tamaño de los pasos de actualización de los parámetros en cada iteración de optimización. Los valores habituales para ajuste de modelos basados en BERT están entre $2e - 5$ y $5e - 5$ [29]. En nuestro caso nos quedamos con un `learning_rate = 3e-5`. Se emplea un scheduler lineal, que hace decrecer la tasa de aprendizaje desde su valor inicial hasta cero a lo largo del entrenamiento, y un `warmup_ratio` del 6% de los pasos, durante los cuales la tasa de aprendizaje aumenta linealmente desde 0 hasta el valor objetivo. Esta fase inicial de calentamiento estabiliza la optimización en las primeras iteraciones y evita gradientes excesivos al inicio.
- **Suavizado de etiquetas:** El script incluye un parámetro de *label smoothing*, técnica que consiste en reemplazar las etiquetas *one-hot* por distribuciones ligeramente suavizadas, asignando $1-\epsilon$ a la clase correcta en vez de 1 y distribuyendo ϵ entre las demás en vez de asignarles 0, donde ϵ es un valor muy pequeño (normalmente entre 0.05 y 0.1) [9]. Esto reduce la sobreconfianza del modelo, y es especialmente útil en contextos donde las etiquetas no son del todo fiables.
- **Clipping de gradiente:** Se establece una norma máxima para el gradiente de 1.0, aplicando *gradient clipping*. Esta técnica limita la magnitud de los gradientes en cada actualización y es una defensa clásica contra la explosión de gradientes en redes neuronales, problema que se observó inicialmente en redes neuronales recursivas en las que el gradiente atraviesa muchos pasos temporales compartiendo los mismos pesos [30]. Los *Transformers* no son recurrentes, pero siguen siendo redes profundas con composiciones no lineales, en las que se tienen múltiples capas (6 en el caso de DistilBERT), cada una con atención multi-cabeza, proyecciones lineales y *feed-forward* con dos capas densas. El gradiente atraviesa todas esas transformaciones, por lo que es adecuado aplicar *clipping* de gradiente para prevenir explosiones.

Artefactos y métricas producidas

Por último, el script genera una serie de salidas, siendo las más importantes:

- **Modelo y tokenizer:** Tras el entrenamiento, el `Trainer` guarda el mejor modelo en el directorio indicado, y también se guarda el tokenizer. De este modo el pipeline produce un paquete completo listo para inferencia o posteriores etapas.
- **Métricas de entrenamiento, validación y test:** El script escribe un fichero `train_results` en formato JSON con las métricas devueltas por `trainer.train()` para entrenamiento y por `trainer.evaluate()` para validación y test. Estas métricas incluyen, entre otras, la pérdida, la accuracy y el F1 macro en cada split.
- **Matriz de confusión:** A partir de las predicciones sobre el conjunto de test se calcula la matriz de confusión y se guarda tanto en formato CSV como en formato gráfico, incluyendo recuentos absolutos y normalizados por fila en la representación visual. Este artefacto nos será especialmente útil a la hora de analizar los resultados para detectar con qué clases encuentra problemas el modelo y qué pares de clases suele confundir entre sí.
- **Curvas ROC y AUC:** Se aplica softmax a las logits del test para obtener probabilidades por clase, y a partir de ellas se calculan curvas ROC micro-promedio, macro-promedio y, curvas ROC por clase individual. La curva ROC (relación entre la tasa de verdaderos positivos y la tasa de falsos positivos al variar el umbral aplicado sobre las probabilidades del modelo) y los valores AUC correspondientes nos servirán para analizar la calidad del "ranking" de nuestro modelo, más que su rendimiento a un umbral específico.
- **CSV de predicciones:** Al igual que con el modelo *baseline*, guardamos para el conjunto de test las predicciones realizadas por el modelo entrenado para cada comentario junto a su clase real. Por motivos de comodidad del desarrollo, la inferencia y guardado de estas predicciones se realiza en el script `evaluate`, estando así desacoplado de la ejecución del entrenamiento.

4.5. Análisis de interpretabilidad

Una vez tenemos el modelo ajustado, el siguiente paso es complementar las métricas con un análisis de interpretabilidad que nos permita entender cómo toma las decisiones internamente el modelo. Se ha optado por dos métodos con enfoques distintos, y por lo tanto complementarios: SHAP como método de interpretabilidad externa y BertViz como herramienta de interpretabilidad interna. SHAP trata al modelo como una caja negra y explica sus predicciones únicamente a partir de las entradas y salidas, estimando qué características (tokens o palabras) influyen más en el resultado sin acceder

a su estructura interna. En cambio, BertViz analiza directamente los mecanismos del modelo, en particular los pesos de atención entre tokens y capas, para mostrar cómo el modelo procesa la información. Mientras SHAP explica qué factores influyen en una decisión, BertViz intenta mostrar cómo el modelo construye esa decisión internamente.

A continuación detallamos las implementaciones realizadas para ambos, que serán utilizadas en el siguiente capítulo para analizar y evaluar nuestro modelo.

4.5.1. Explicabilidad basada en resultados: método SHAP

En el segundo capítulo entramos en detalle sobre el método SHAP a nivel teórico. En el contexto de nuestra tarea, la utilidad de este se traduce en detectar cuánto ha aportado positiva o negativamente cada token de un comentario en la probabilidad asignada a cada una de las clases para dicho comentario.

La biblioteca `shap` de Python ofrece toda la funcionalidad necesaria para implementar el análisis con SHAP, que se realiza mediante dos scripts:

El script `compute_shap` calcula los valores SHAP sobre la totalidad del split de test utilizando el modelo ajustado tras el entrenamiento, estimando la contribución media de cada token a las predicciones. A partir de estas contribuciones, se agregan los valores por clase y se generan, para cada una de ellas, visualizaciones separadas que muestran los tokens con mayor impacto positivo y negativo en la probabilidad asignada a dicha clase, ofreciendo así una visión global del comportamiento léxico del modelo.

Por otro lado, el script `explicar_ejemplo` se centra en la interpretabilidad local, calculando los valores SHAP para un único comentario de entrada y generando una visualización interactiva en formato HTML que permite inspeccionar, token a token, cómo cada palabra contribuye positiva o negativamente a las probabilidades de todas las clases. De esta manera, este script nos permitirá analizar casos concretos de comentarios que seleccionemos.

4.5.2. Visualización interna de la atención del modelo: BertViz

BertViz es una herramienta de visualización interactiva diseñada para analizar y comprender el funcionamiento interno de modelos basados en la arquitectura *Transformer*, en particular BERT y sus variantes. Fue desarrollada por Jesse Vig, investigador en interpretabilidad de modelos de lenguaje, y presentada originalmente junto con un artículo que introduce el uso de visualizaciones como medio para inspeccionar los patrones de atención aprendidos por estos modelos. BertViz está implementada en Python y JavaScript, integrándose directamente con modelos de *Hugging Face Transformers* y utilizando visualizaciones interactivas basadas en HTML para mostrar los pesos de atención y activaciones internas de las distintas capas del modelo [31].

La herramienta ofrece tres vistas complementarias que permiten analizar el modelo a distintos niveles de granularidad:

- **Vista head:** se centra en una única capa y un único *attention head*, mostrando de forma explícita las relaciones de atención entre tokens de entrada y salida, lo que permite observar patrones lingüísticos concretos como dependencias sintácticas o concordancias locales.
- **Vista model:** proporciona una perspectiva más global, visualizando simultáneamente todas las capas y cabezas de atención del modelo y permitiendo identificar qué capas o cabezas son más activas o relevantes para un determinado texto.
- **Vista neuron:** muestra las activaciones internas asociadas a tokens concretos y facilita la inspección de cómo ciertas neuronas responden a patrones específicos del lenguaje.

Se ha implementado el script `compute_bert` para aplicar BertViz a comentarios individuales del dataset. El script carga el modelo transformer ajustado y su tokenizador correspondiente, procesa un texto de entrada y extrae las matrices de atención necesarias para ser visualizadas mediante BertViz. A partir de estas salidas, se generan las vistas interactivas que permiten explorar cómo el modelo distribuye su atención entre los distintos tokens y capas para un comentario concreto. Esta implementación complementa el análisis basado en SHAP, aportando una explicación interna del modelo que permite estudiar directamente los mecanismos de atención que intervienen en cada predicción.

5. Análisis y evaluación de los resultados

Una vez descrito en detalle el desarrollo del trabajo, pasamos a analizar los resultados obtenidos tras aplicar las herramientas implementadas, con el objetivo de valorar de forma crítica el rendimiento de los distintos enfoques propuestos para la clasificación temática de comentarios de YouTube.

Además de analizar las métricas cuantitativas, realizaremos un análisis cualitativo basado en ejemplos del conjunto de test de nuestro dataset y utilizando las herramientas de interpretabilidad implementadas al final del desarrollo. Inspeccionaremos manualmente comentarios seleccionados de forma estratégica para cubrir todas las clases en la medida de lo posible, además de prestar especial atención a los errores, entre ellos aquellos casos concretos en los que un modelo falle y otro no. Este análisis cualitativo nos servirá para caracterizar patrones típicos de fallo (por ejemplo ambigüedad temática, dependencia de contexto, presencia de términos polisémicos o jerga) y conectar dichos fallos con las diferencias entre enfoques (representaciones dispersas TF-IDF frente a representaciones contextualizadas en *Transformers*).

5.1. Entorno de evaluación y configuración final

Lo primero que debemos hacer antes de lanzarnos al análisis es establecer claramente el entorno de evaluación: las circunstancias exactas en las que se han producido los resultados. Durante el desarrollo se realizaron múltiples ejecuciones exploratorias variando splits e hiperparámetros (manteniendo la semilla fija). Tras comparar en cada uno de los modelos los resultados de las distintas configuraciones, la configuración final utilizada en cada uno de los enfoques es la siguiente:

- **Baseline TF-IDF + Regresión Logística:** El texto se representa mediante un vector TF-IDF utilizando unigramas y bigramas (`ngram_range=(1, 2)`), descartando términos extremadamente raros (`min_df=2`) y extremadamente frecuentes (`max_df=0.9`), y aplicando escalado sublineal de la frecuencia (`sublinear_tf=True`). Sobre estas representaciones se entrena un clasificador de regresión logística multinomial con regularización L2 (`penalty="l2"`), parámetro de regularización `C=1.0`, optimizador `lbfgs`, un máximo de 1000 iteraciones y paralelización en CPU (`n_jobs=-1`).

- **Zero-shot con Transformer:** La clasificación se realiza sin ajuste supervisado empleando el modelo mDeBERTa-v3-base-mnli-xnli a través del *pipeline* de Hugging Face para clasificación *zero-shot*. Para cada comentario se evalúa su compatibilidad semántica con las nueve etiquetas candidatas, configurando el modo multiclas (multi_label=False). La predicción final se obtiene seleccionando la clase con mayor puntuación.
- **Transformer DistilBERT ajustado:** Se parte del modelo distilbert-base-multilingual-cased, que se ajusta de forma supervisada sobre el conjunto de entrenamiento. Los textos se tokenizan con una longitud máxima de 256 tokens y truncado activado, aplicándose *padding* dinámico durante el entrenamiento. El entrenamiento se realiza con un tamaño de lote de 16 ejemplos y evaluación con lotes de 32, durante un máximo de 10 épocas, utilizando una tasa de aprendizaje de 3e-5, decaimiento de pesos 0.005, planificador lineal con *warmup* del 6% y recorte de gradiente con norma máxima 1.0. La selección del mejor modelo se realiza sobre el conjunto de validación empleando F1 macro como métrica objetivo, activando *early stopping* con una paciencia de 4 épocas y cargando automáticamente el mejor *checkpoint*. La semilla fija 42 se utiliza para garantizar reproducibilidad.

El objetivo final de la evaluación es medir, de forma comparable entre enfoques, la capacidad de generalización de los modelos para clasificar comentarios de YouTube según su temática. Definimos el “éxito” en esta tarea principalmente en términos de rendimiento equilibrado entre categorías. Por ello, la métrica principal para comparar modelos será F1 macro, complementada con métricas globales y análisis cualitativo de errores.

5.2. Evaluación comparativa de los modelos

Con el objetivo de valorar el comportamiento del modelo *Transformer* ajustado en relación con los enfoques más simples desarrollados previamente, vamos a realizar una evaluación comparativa de los distintos modelos implementados. Además de cuantificar el rendimiento, analizaremos qué tipo de errores comete cada enfoque, en qué contextos acierta o falla, y qué implicaciones tienen esas diferencias desde el punto de vista del análisis semántico y la comprensión de lenguaje en contextos informales como YouTube. Esta perspectiva comparativa nos permitirá situar nuestro modelo *Transformer* dentro de un marco de referencia más amplio, identificando sus fortalezas y debilidades relativas respecto a soluciones más sencillas y tradicionales.

5.2.1. Resultados del modelo *baseline*

Resultados del modelo TF-IDF + Regresión Logística

La tabla 5.1 muestra los resultados globales y por clase a nivel de métricas estándar. La tabla 5.2 se compone de los rasgos léxicos más característicos detectados por el modelo para cada categoría temática, y la figura 5.1 corresponde a la distribución de aciertos y errores observada en la matriz de confusión.

Tabla 5.1: Métricas de rendimiento del modelo TF-IDF + Regresión Logística

Clase	Precision (%)	Recall (%)	F1-Score (%)
Animales	59.105	51.825	55.226
Ciencia	59.592	54.759	57.073
Cine	65.760	48.658	55.931
Deportes	73.773	70.528	72.114
Música	58.503	70.367	63.889
Noticias	54.836	71.682	62.138
Tecnología	63.364	52.885	57.652
Vehículos	68.111	66.554	67.323
Viajes	39.364	60.559	47.713
Macro avg	60.267	60.868	59.895
Weighted avg	61.633	60.387	60.446
Accuracy		60.387	

Desde el punto de vista cuantitativo, el modelo alcanza una precisión general del 60,38% y una puntuación F1 macro de 59,89%, lo cual refleja un rendimiento moderado. A nivel de clases, destacan los resultados positivos en categorías como deportes ($F1 = 0.74$), y vehículos ($F1 = 0.68$), mientras que el desempeño desciende notablemente en noticias ($F1 = 0.55$) y especialmente viajes ($F1 = 0.39$). Las clase ciencia y cine presentan una puntuación especialmente baja en recall (0.55, 0.49), lo cual sugiere que el modelo no logra capturar suficientemente los comentarios de esta temática. En conjunto, estas cifras muestran una capacidad razonable del modelo para captar temas con vocabulario más específico y frecuente, pero un margen de mejora significativo en clases más ambiguas o solapadas.

El análisis de los diez rasgos léxicos más influyentes por clase refuerza esta observación. En la mayoría de los casos, el modelo identifica con acierto términos muy representativos y discriminativos. Por ejemplo, para la clase cine destacan tokens como película, actor o escena; en deportes aparecen fútbol, Messi y partido; y en música, términos como canción, canta o voz. Estas asociaciones directas evidencian que el modelo basa gran parte de su capacidad predictiva en la detección de palabras clave. Sin embargo, en algunas clases como ciencia o viajes, ciertos términos altamente ponderados pero no tan evidentemente representativos de la clase como miwu o alquimia podrían

Tabla 5.2: Modelo *baseline*: top 10 rasgos (tokens) más relevantes por clase

Animales		Ciencia		Cine	
Peso	Token	Peso	Token	Peso	Token
10.69	animales	11.10	miwu	10.96	cine
6.77	animal	4.88	ciencia	10.87	pelicula
5.19	leon	4.78	agua	7.96	peliculas
5.09	cocodrilo	3.48	hielo	4.93	actor
4.95	aguila	3.29	cuerpo	4.54	actores
4.83	serpiente	3.23	la luna	4.53	peli
4.56	salvaje	3.18	quimica	4.09	escena
4.41	naturaleza	3.13	platino	4.08	alejo
4.29	animalitos	3.01	liquido	3.43	del cine
4.05	pez	2.91	pelon	3.38	chucky
Deportes		Música		Noticias	
Peso	Token	Peso	Token	Peso	Token
6.49	futbol	11.27	cancion	8.30	trump
5.87	equipo	7.96	canciones	8.08	venezuela
5.66	america	7.60	musica	7.23	maduro
5.59	gol	5.89	tema	6.28	presidente
5.39	chivas	5.81	cazzu	5.35	pueblo
5.11	portero	4.88	temazo	5.14	gobierno
5.10	partido	4.87	canta	5.13	estrategias
5.07	seleccion	4.57	voz	5.08	noticias
5.00	jugadores	4.53	chuy	4.65	pais
4.86	messi	4.08	romeo	4.16	dios
Tecnología		Vehículos		Viajes	
Peso	Token	Peso	Token	Peso	Token
8.24	tecnologia	7.02	autos	7.39	viaje
6.45	iphone	6.84	auto	5.44	viajar
6.12	telefono	6.63	carro	5.16	lugares
5.99	celular	6.56	vehiculos	4.88	ir
5.33	productos	6.52	coche	4.54	luisito
4.66	pantalla	6.44	motor	4.48	pais
4.65	pro	6.03	toyota	4.36	alquimia
4.60	china	5.61	vehiculo	4.28	conocer
4.34	apple	5.27	carros	4.20	hermoso
4.11	samsung	5.04	coches	4.12	nico

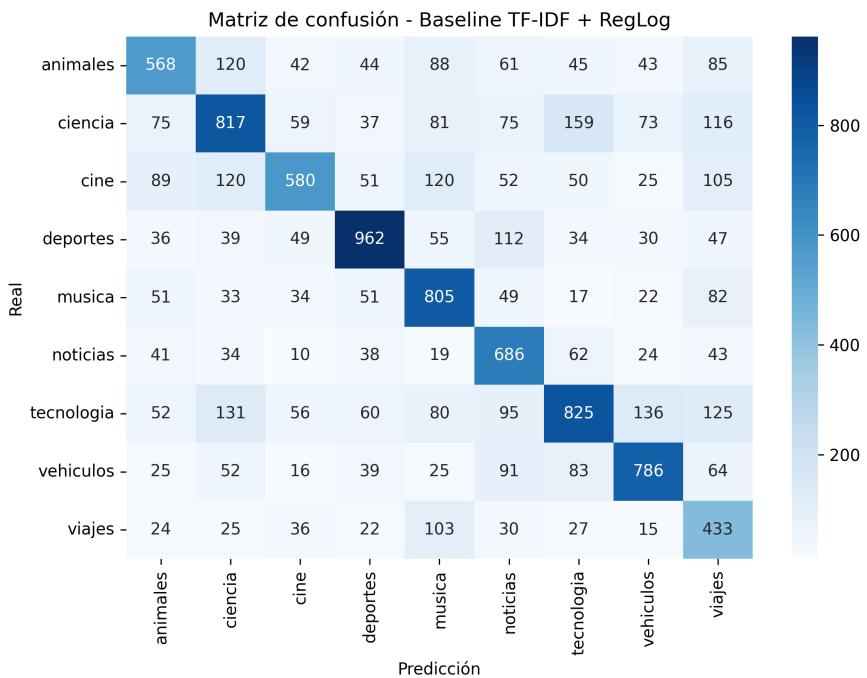


Figura 5.1: Matriz de confusión para el modelo *baseline* TF-IDF + Regresión Logística

indicar la presencia de ruido o una correlación errónea derivada de una coocurrencia frecuente pero no semánticamente representativa. Este fenómeno podría contribuir a la menor capacidad del modelo para generalizar en dichas clases.

La matriz de confusión aporta una visión más fina sobre cómo se distribuyen los errores entre clases. Se observa que la mayoría de las categorías presentan una diagonal dominante, lo cual indica que el modelo tiende a acertar más de lo que falla. Sin embargo, hay clases con una notable dispersión de errores: ciencia muestra muchas predicciones incorrectas hacia tecnología y viajes, y cine hacia música y viajes, lo cual sugiere cierto solapamiento temático o limitaciones del modelo para discriminar entre vocabularios parcialmente coincidentes. En viajes, por ejemplo, los errores están muy repartidos, lo que puede explicarse por la alta ambigüedad léxica y el uso de términos genéricos como ir, hermoso o país, que también aparecen en otras categorías. Aunque el modelo logra captar patrones léxicos claros en clases bien definidas, sufre ante temáticas menos delimitadas o léxicamente más difusas.

La dificultad de este modelo para discriminar clases solapadas, incluso con el uso de bigramas, es una consecuencia directa de su naturaleza como modelo lineal: al basar sus decisiones en una combinación lineal de la frecuencia ponderada de los términos (TF-IDF), el modelo es intrínsecamente incapaz de capturar las interacciones no lineales y las dependencias complejas que definen el contexto semántico de un comentario. Esto significa que, si bien puede detectar palabras clave (por ejemplo, "fútbol" en deportes), no puede comprender el significado de una palabra basándose en el resto de la frase, un

límite que los modelos *Transformer* superan mediante la atención contextualizada [1]. Por lo tanto, el rendimiento moderado de este *baseline* establece una referencia clara: la clasificación temática de comentarios de YouTube requiere un modelo con capacidad para manejar la ambigüedad y el contexto.

Resultados del modelo *zero-shot*

Para el modelo *zero-shot* analizamos el comportamiento global, por clase y en términos de distribución de errores a través de las métricas presentadas en la tabla 5.3. La figura 5.2 muestra la matriz de confusión resultante.

Tabla 5.3: Métricas de rendimiento del modelo *zero-shot*

Clase	Precision (%)	Recall (%)	F1-Score (%)
Animales	38.695	61.679	47.555
Ciencia	69.000	13.874	23.103
Cine	37.567	41.443	39.410
Deportes	60.736	44.795	51.561
Música	54.881	54.545	54.713
Noticias	34.457	33.124	33.777
Tecnología	57.174	50.064	53.383
Vehículos	68.445	64.098	66.200
Viajes	23.392	59.021	33.505
Macro avg	49.372	46.960	44.801
Weighted avg	51.910	45.687	45.251
Accuracy		45.687	

A nivel global, el rendimiento del modelo es bastante limitado, con una accuracy del 45,69% y una F1 macro de 44,80%. Aunque algunas clases alcanzan valores aceptables, como vehículos ($F1 = 0.66$), deportes ($F1 = 0.51$) o música ($F1 = 0.54$), en otras categorías el desempeño es claramente deficiente. Destaca negativamente la clase ciencia, con una F1 de apenas 0.23, debida a una fuerte caída en el recall (13.87%). También viajes y noticias presentan puntuaciones F1 cercanas a 0.33, lo que indica que el modelo tiene dificultades para identificar correctamente estas temáticas. Estas cifras reflejan una alta variabilidad entre clases, probablemente influida por la ambigüedad del lenguaje, la longitud de los comentarios o la similitud entre descripciones hipotéticas.

La matriz de confusión proporciona información adicional sobre la naturaleza de los errores cometidos. Se observa una fuerte tendencia a la dispersión de las predicciones, especialmente en clases como ciencia, que recibe asignaciones significativas en casi todas las etiquetas posibles, y viajes, que presenta un patrón de confusión claro, con un número elevado de instancias clasificadas erróneamente como música, cine y animales.

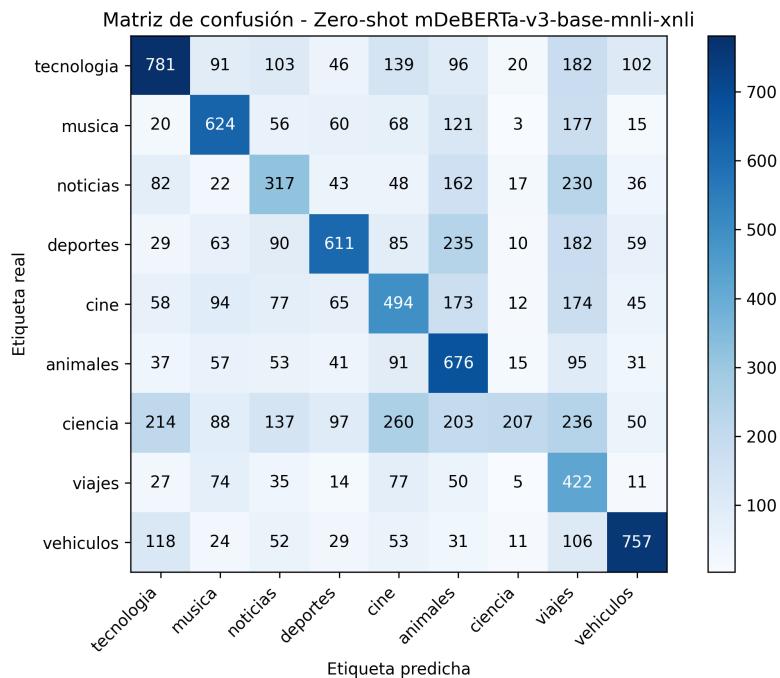


Figura 5.2: Matriz de confusión para el modelo *zero-shot*

Esto sugiere que el modelo *zero-shot* no consigue establecer separaciones claras entre las clases cuando el vocabulario empleado en los comentarios es genérico, ambivalente o poco alineado con las hipótesis formuladas. Por otro lado, se detecta un comportamiento más consistente en clases como vehículos y tecnología, donde se observa una acumulación más clara en la diagonal, lo que puede deberse a una mayor especificidad léxica en los textos de esas categorías.

En conjunto, estos resultados exponen tanto las posibilidades como las limitaciones de la inferencia *zero-shot* en tareas de clasificación temática sobre texto informal. El modelo es capaz de capturar patrones útiles en clases con señales semánticas bien definidas y términos discriminativos frecuentes, pero falla sistemáticamente cuando se enfrenta a categorías menos concretas, con solapamientos léxicos o con expresiones ambiguas. Esto deja en evidencia la dependencia crítica de esta estrategia respecto a la formulación de las hipótesis y a la alineación entre el lenguaje del conjunto de textos original y el conjunto de textos de entrenamiento del modelo. El desafío no es la capacidad inherente del *Transformer* para entender el lenguaje, sino su incapacidad para aplicar su conocimiento previo al dominio informal, ambiguo y conciso de los comentarios de YouTube sin ajuste supervisado.

5.2.2. Resultados del modelo *Transformer* basado en DistilBERT

La tabla 5.4 muestra las métricas cuantitativas globales y por conjunto del dataset, mientras que las figuras 5.3 y 5.4 muestran la distribución de errores a través de la matriz de confusión y los valores ROC-AUC por clase, respectivamente.

Tabla 5.4: Métricas de rendimiento del modelo *Transformer* basado en DistilBERT

Clase	Precision (%)	Recall (%)	F1-Score (%)
Animales	67.296	58.577	62.634
Ciencia	66.490	58.914	62.473
Cine	59.207	52.601	55.709
Deportes	76.570	73.314	74.906
Música	57.388	76.049	65.414
Noticias	63.900	69.175	66.433
Tecnología	67.386	54.038	59.979
Vehículos	67.326	72.058	69.611
Viajes	43.303	60.140	50.351
Macro avg	63.207	63.874	63.057
Weighted avg	64.515	63.583	63.604
Accuracy			63.583

Desde el punto de vista numérico, los resultados reflejan una mejora notable aunque moderada en las métricas de evaluación sobre el conjunto de test. El modelo alcanza una accuracy del 63,58% y una F1 macro de 63,06%, lo que indica un rendimiento global sólido en un escenario multiclase equilibrado. La consistencia entre las métricas en validación y test sugiere una generalización razonablemente estable, con diferencias menores entre ambos conjuntos. Además, la pérdida de validación (`eval_loss`) y de test se mantienen en torno a 1.31–1.32, lo que apunta a un aprendizaje sostenido sin signos de sobreajuste severo. Estos valores permiten anticipar una capacidad del modelo para capturar patrones complejos en los datos, incluso sin un preprocesamiento extremadamente restrictivo.

La matriz de confusión confirma esta impresión, mostrando una diagonal bien definida en la mayoría de las clases, lo que evidencia una fuerte tendencia del modelo a acertar en las predicciones. Las clases como deportes, noticias y música presentan especialmente buenos niveles de acierto, con concentraciones marcadas en sus respectivas celdas diagonales. No obstante, aún persisten errores dispersos en categorías como ciencia, cine o viajes, donde se observan confusiones hacia clases temáticamente cercanas. Esto sugiere que, aunque el modelo ha aprendido a identificar señales específicas en la mayoría de los casos, algunas temáticas siguen resultando difíciles de separar por la similitud en el vocabulario empleado por los usuarios.

El análisis de las curvas ROC-AUC por clase permite evaluar la capacidad del modelo

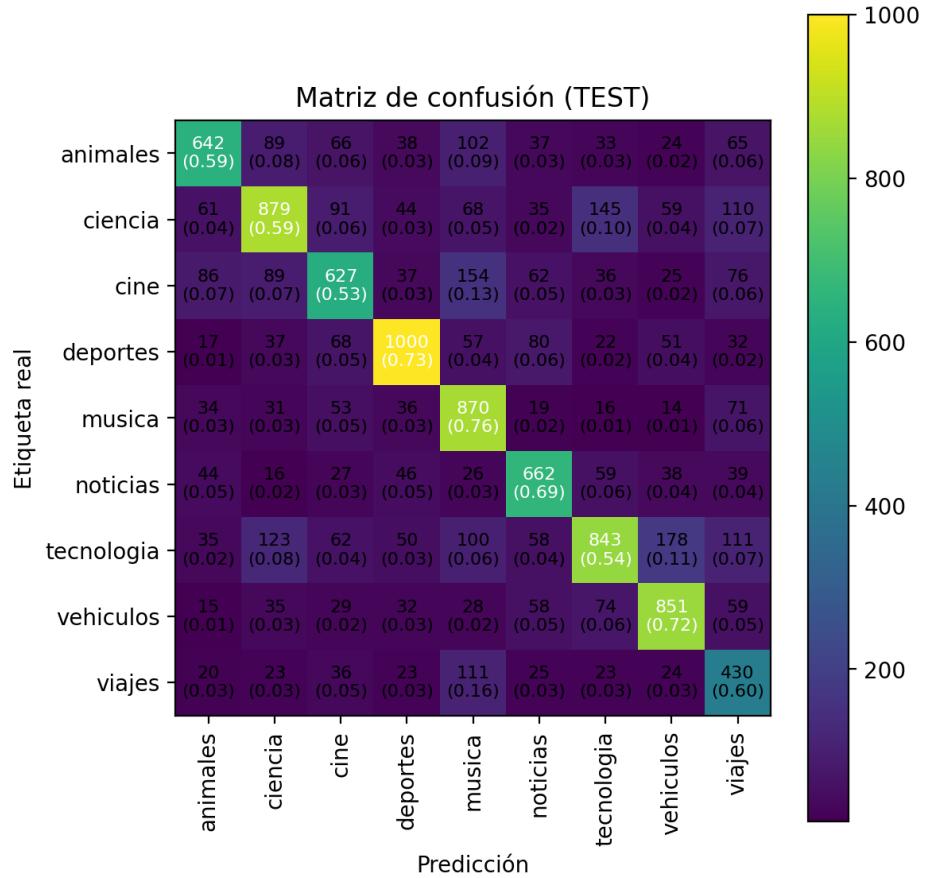


Figura 5.3: Matriz de confusión para el modelo *Transformer* basado en DistilBERT

para discriminar entre categorías de forma binaria. El valor macro promedio (0.9073) y el micro promedio (0.9069) confirman una sensibilidad alta generalizada. A nivel individual, destacan deportes (0.9436), música (0.9387) y vehículos (0.9326) como las clases más fácilmente separables, mientras que cine (0.8618) y tecnología (0.8799) presentan valores algo más bajos, aunque aún dentro de un rango alto. Esta información refuerza la idea de que el modelo logra establecer fronteras de decisión claras en la mayoría de los casos, siendo menos preciso únicamente en temáticas donde el solapamiento semántico o la ambigüedad contextual dificultan la separación entre clases.

Antes de avanzar con el análisis comparativo de los resultados de los distintos modelos, cabe mencionar que además de DistilBERT, que es multilingüe, se entrenó un modelo basado en BETO, un modelo BERT de tamaño similar entrenado sobre un corpus masivo de textos en español [32]. La tabla 5.5 compara los resultados de entrenamiento de ambos.

Observamos un F1 macro similar, pero con una pérdida notablemente superior en BETO (2.30 vs 1.32). Esto se puede atribuir a que la especialización en español estándar

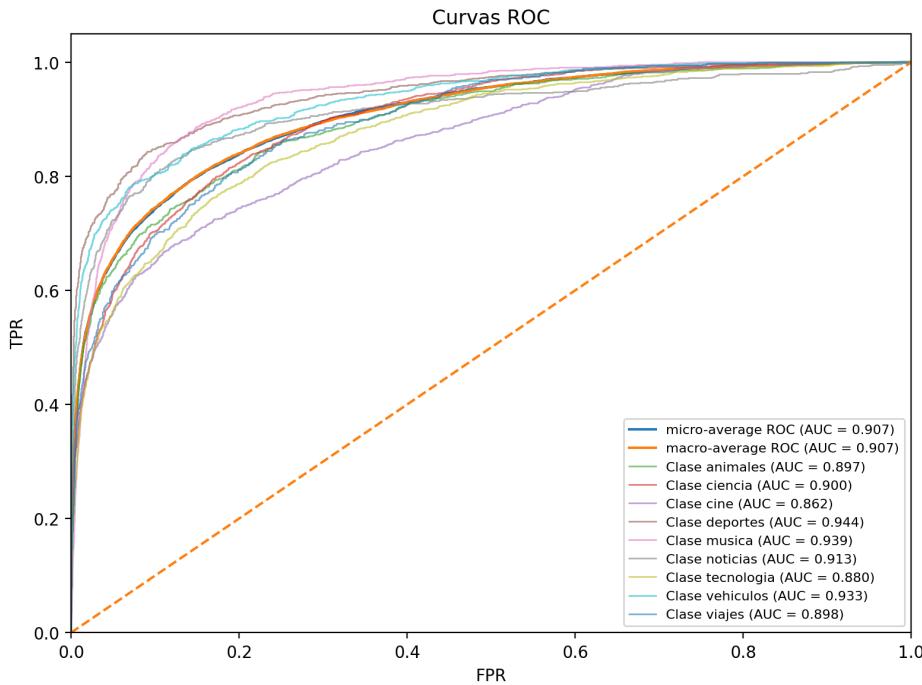


Figura 5.4: Curvas ROC-AUC para el modelo *Transformer* basado en DistilBERT

de BETO no se alinea con el lenguaje altamente informal, lleno de jerga y neologismos, propio de los comentarios de YouTube. BETO fue entrenado sobre un corpus de textos en español relativamente formales (Wikipedia, transcripciones de conferencias, libros) [32], corpus que quizá no captura la variedad lingüística, a menudo globalizada de los textos publicados en YouTube. La arquitectura multilingüe de DistilBERT demuestra ser más flexible para capturar patrones de comunicación digital global, superando la rigidez léxica de un modelo entrenado en un corpus masivo pero formal del español.

Tabla 5.5: Resultados de entrenamiento de DistilBERT y BETO

Conjunto	Modelo	Loss	Accuracy (%)	F1 macro (%)
Train	DistilBERT	0.5830648933		
	BETO	0.7751340401		
Validación	DistilBERT	1.3107496500	62.241	62.099
	BETO	2.2575161457	63.389	63.505
Test	DistilBERT	1.3269534111	63.583	63.057
	BETO	2.3075048923	62.732	62.597

5.2.3. Análisis comparativo de resultados

Tras analizar de manera aislada el rendimiento de cada modelo, pasamos al análisis comparativo de los tres enfoques, con el objetivo de validar la ganancia de rendimiento que justifica la complejidad computacional de la arquitectura final.

La tabla 5.6 resume las métricas globales clave obtenidas por cada modelo sobre el conjunto de test.

Tabla 5.6: Comparativa de métricas globales de rendimiento

Modelo	Accuracy (Test)	F1 Macro (Test)
<i>Baseline</i> TF-IDF + Regresión Logística	60.387	59.895
<i>Baseline</i> Zero-shot (<i>mDeBERTa-v3</i>)	45.687	44.801
<i>Transformer</i> DistilBERT (Ajustado)	63.583	63.057

Los resultados globales demuestran que el modelo *Transformer* ajustado es el que obtiene el rendimiento más sólido y equilibrado, con un F1 macro de 63.06% y una precisión general del 63,583%. Este desempeño supera moderadamente al *baseline* TF-IDF (F1 macro 59,895%) y, de forma más contundente, al modelo *zero-shot* (F1 macro 44,801%).

La diferencia más significativa se observa entre los dos enfoques basados en *Transformers*. El modelo *zero-shot* demuestra tener una capacidad de generalización muy limitada en el dominio específico de los comentarios de YouTube. Su bajo F1 macro y la fuerte dispersión de predicciones en la matriz de confusión confirman que, a pesar de usar una arquitectura potente (*mDeBERTa-v3*), el modelo es incapaz de aplicar eficazmente su conocimiento general del lenguaje sin alineación al dominio. Su dependencia de la formulación de hipótesis no se traduce bien a la jerga y ambigüedad del contenido de los comentarios de YouTube. En contraste, el ajuste supervisado de DistilBERT permitió al modelo aprender representaciones contextuales específicas del lenguaje de YouTube, lo que resultó en una mejora absoluta de casi 20 puntos porcentuales en F1 macro (0.63 vs 0.44), justificando plenamente el proceso de *fine-tuning*.

Al comparar el *baseline* TF-IDF con el *Transformer* ajustado observamos un éxito más moderado, lo cual dificulta la justificación del *Transformer* como método estrictamente superior. Sin embargo, el TF-IDF fracasa sistemáticamente en categorías con alto solapamiento léxico o que dependen del contexto para su interpretación. Su incapacidad para capturar interacciones no lineales y dependencias complejas resulta en una pobre capacidad de discriminación en viajes (F1 0.47), por ejemplo. El modelo TF-IDF, siendo lineal, basa su éxito en la aparición de palabras clave altamente discriminativas (como "futbol" en deportes o "pelicula" en cine). Esto le permite obtener resultados fuertes en clases con vocabulario muy específico, como deportes (F1 0.72) y vehículos (F1 0.67), pero falla en caso contrario.

El *Transformer* DistilBERT, gracias a su mecanismo de atención, mitiga estos pro-

blemas al entender el significado de los tokens en función del resto de la frase. Su ventaja más crucial es la generalización más estable a través de todas las clases. Por ejemplo, en Ciencia (donde el TF-IDF tiene F1 0.57 y *zero-shot* F1 0.23), DistilBERT alcanza F1 0.62, demostrando una mejor captura de la semántica compleja.

A pesar de la mejora general, el análisis de las matrices de confusión y las métricas por clase revela que la clase viajes sigue siendo el principal punto débil para todos los enfoques. El solapamiento semántico y la ambigüedad inherente al vocabulario utilizado en los comentarios de viajes (términos genéricos como "hermoso", "ir", "pais") persisten como el principal desafío. Las curvas ROC-AUC, si bien altas en promedio (macro 0.9073), también sitúan a viajes y cine entre las clases con menor discriminación. Esta persistencia de errores, incluso en el modelo más avanzado, justifica el siguiente paso del análisis: profundizar en la naturaleza de estos fallos y utilizar herramientas de interpretabilidad para entender por qué el modelo DistilBERT no logra mejorar las fronteras de decisión en estos casos de alta ambigüedad semántica.

5.3. Análisis de errores

El análisis de rendimiento y la evaluación de métricas de la sección anterior confirmaron la superioridad del modelo *Transformer* ajustado sobre los *baselines*. Sin embargo, para obtener una comprensión profunda del comportamiento del modelo y de las dificultades intrínsecas de la tarea de clasificación, es fundamental (y mucho más informativo) examinar los fallos cometidos. Este análisis de errores nos permitirá caracterizar los patrones de error específicos de cada arquitectura, justificando por qué la contextualización (*Transformer*) corrige ciertos fallos del modelo léxico-lineal (TF-IDF) y, a su vez, identificando los límites de la mejora conseguida, especialmente en las clases temáticamente ambiguas.

5.3.1. Análisis cuantitativo

Con el propósito de extraer los errores de una manera accesible y legible, implementamos post-desarrollo los scripts `filtrar_errores` y `comparacion_errores`. Estos scripts sencillos utilizan los CSV de predicciones realizadas sobre el conjunto de test para extraer los errores cometidos por cada modelo, y comparan ambos conjuntos de errores para dividirlos en tres grupos: aquellos realizados por un modelo pero no el otro, viceversa, y aquellos realizados por ambos. La tabla 5.7 muestra el desglose exacto.

El modelo *Transformer* comete menos errores en total, lo que reafirma la mejora cuantitativa global observada en las métricas (F1 macro 0.63 vs 0.58). Sin embargo, el hallazgo más significativo es el elevado número de errores compartidos: 3029 comentarios, que representan más del 71% de los fallos del *baseline*. Este porcentaje sugiere que la mayor parte de la dificultad en la clasificación no se debe a las limitaciones de una arquitectura específica (lineal vs. contextual), sino a la ambigüedad semántica intrínseca y la dificultad para asignar una etiqueta clara a un comentario, independientemente

Tabla 5.7: Desglose de errores entre modelo *baseline* y modelo *Transformer*

Concepto	Recuento	Porcentaje
Total errores modelo <i>baseline</i>	4239	—
Total errores modelo <i>Transformer</i>	3897	—
Errores solo en <i>baseline</i>	1210	28.54%
Errores solo en <i>Transformer</i>	868	22.27%
Errores mutuos (compartidos)	3029	71.46%

de la potencia del modelo. Estos son los casos de solapamiento léxico o comentarios sin contexto temático fuerte, tal como se había anticipado en el análisis de las matrices de confusión. La verdadera diferencia entre modelos reside en los errores no compartidos. El *Transformer* corrigió 1210 errores que el *baseline* cometió, mientras que el *baseline* corrigió 868 errores del *Transformer*.

5.3.2. Análisis cualitativo

Pasemos a examinar los ejemplos en los tres grupos definidos para caracterizar los patrones de fallo. En esta sección usaremos unos pocos ejemplos especialmente representativos para guiar el análisis sin hacerlo demasiado extenso. Estos ejemplos provienen de un conjunto mayor de ejemplos de errores seleccionados, estando este conjunto completo en el Anexo A. Es muy recomendable echar un vistazo a todos los ejemplos para obtener una mejor visión de los patrones analizados en esta sección.

Errores del *baseline* corregidos por el *Transformer*

Estos 1210 errores son la clave para justificar la inversión en la arquitectura *Transformer* y el proceso de *fine-tuning*. Los fallos del *baseline* TF-IDF provienen de su naturaleza lineal, incapaz de entender el significado de una palabra basándose en el resto de la frase:

- **Corrección de polisemia y ambigüedad léxica:** El *Transformer* logra desambiguar términos que son sobreponderados por el TF-IDF en una clase incorrecta. Por ejemplo, el comentario:

"el gato de schrodinger lo llevo escuchando desde que era pequeno"

El *baseline* lo predice como animales, probablemente por el alto peso de la palabra "gato" en esa clase, mientras que el *Transformer* lo clasifica correctamente como ciencia al contextualizar "gato" con "schrodinger". De manera similar, en "en baterias y tecn... muy bonito el twingo", el *baseline* se queda en tecnología por "baterías" y falla, mientras que el *Transformer* acierta en vehículos por "twingo" y el contexto del comentario.

- **Detección de contexto invertido o sarcasmo:** El modelo lineal no capta la intención. El comentario *"no estas viendo una territory sino una chinitory jajaja"* fue clasificado erróneamente por el *baseline* como cine, mientras que el *Transformer* lo sitúa correctamente en vehículos, interpretando la expresión como una referencia a un tipo de coche con una connotación satírica, lo cual es inaccesible a un modelo que solo procesa n-gramas.

Los fallos únicos del *baseline* demuestran que el *Transformer* ajustado utiliza con éxito la información contextual, minimizando los errores provocados por la dependencia de palabras clave directas y la incapacidad de resolver la ambigüedad inherente al lenguaje informal de YouTube.

Errores del *Transformer* corregidos por el *baseline*

Estos 868 errores son especialmente interesantes porque revelan las limitaciones introducidas por la complejidad de la arquitectura y el proceso de *fine-tuning*. El *Transformer*, al aprender patrones más complejos, corre el riesgo de sobreajustarse a correlaciones falsas o volverse excesivamente sensible a jerga de nicho que no fue robustamente representada en el entrenamiento:

- **Sobre-sensibilidad a jerga o especificaciones técnicas:** El *Transformer* puede interpretar un token técnico como demasiado informativo, ignorando el contexto general que el TF-IDF sí logra captar. Por ejemplo, el comentario *"yo tengo la asus tuf f15 i7 13620h rtx 4060 16gb ram"*, que es claramente tecnología, el *Transformer* lo predice como vehículos. Posiblemente, el modelo aprendió que secuencias alfanuméricas complejas están a veces correlacionadas con marcas de vehículos (un tipo de jerga), mientras que el *baseline* se adhiere a términos más tradicionales y correctamente ponderados como *"ram"* y *"rtx"* en tecnología.
- **Confusión temática sutil por proximidad de *embeddings*:** El *fine-tuning* tiende a agrupar semánticamente clases que están relacionadas. Esto suele ser beneficioso, pero a veces causa errores sutiles. El *Transformer* confunde *"transistor cuantico... qbits"*, que es ciencia, con tecnología. Esto sugiere que la frontera de decisión entre las representaciones internas de ambas clases se ha vuelto demasiado estrecha en el espacio de *embeddings* del modelo, un error de sobre-refinamiento que no comete el *baseline* TF-IDF (que se basa en tokens disjuntos).

Errores mutuos

Finalmente, los 3029 errores compartidos son la evidencia de los límites superiores de la tarea con el dataset actual, independientemente del modelo. Estos fallos no son correctibles solo con una mejor arquitectura o más datos, sino que reflejan una ambigüedad fundamental en la etiqueta real o en la naturaleza del comentario.

- **Ambigüedad temática extrema:** Comentarios genéricos que no contienen palabras clave ni contexto temático. Ejemplos como *"excelente video gracias"* o *"saludos desde salta argentina"* son intrínsecamente difíciles de clasificar, ya que podrían aparecer en vídeos de cualquier categoría, confirmando la presencia de ruido residual semántico en el dataset.
- **Clase viajes como punto ciego:** Las métricas ya nos permitieron identificar viajes como la clase más problemática, teniendo el peor F1-score en el *Transformer* (F1 0.50). Los ejemplos de errores mutuos confirman que los comentarios de viajes se confunden constantemente con música y cine, incluso cuando contienen palabras genéricas esperadas: *"buena musica la escucho mientras estoy en el bus"* (se clasifica como musica, pertenece a viajes). La imposibilidad de resolver esta ambigüedad es el principal desafío persistente.
- **Contenido extenso y multitema:** Comentarios muy largos que mezclan temas o son relatos personales. Por ejemplo, discusiones largas sobre hardware/consolas etiquetadas como noticias, o el comentario técnico sobre tanques etiquetado como cine. En estos casos, la longitud (que excede el comentario promedio) y la mezcla de temas desbordan tanto la representación dispersa como la contextual.

El análisis de errores concluye que la implementación del *Transformer* fue exitosa en mitigar los fallos causados por la ambigüedad léxica y contextual (problemas que el TF-IDF no puede resolver). El modelo ajustado logra una mejor capacidad de generalización. Aun así, es evidente que hay un límite del rendimiento intrínseco a la tarea, no establecido por la potencia de ninguno de los modelos, sino por la complejidad y ambigüedad contextual de los comentarios de YouTube.

Errores cometidos por otros modelos (*zero-shot* y BETO)

Antes de pasar al análisis de interpretabilidad, merece la pena echar un vistazo a los errores cometidos por los otros modelos *Transformer* que hemos entrenado. La tabla 5.8 muestra la cantidad de errores cometidos por los modelos *zero-shot* y BETO previamente mencionados, y cuántos de ellos corrige el *Transformer* basado en DistilBERT.

Tabla 5.8: Errores de *zero-shot* y BETO corregidos por DistilBERT

Concepto	Recuento	Porcentaje
Total errores modelo <i>zero-shot</i>	5812	—
Total errores modelo BETO	3988	—
Errores solo en <i>zero-shot</i> (corregidos)	2654	45.66%
Errores solo en BETO (corregidos)	929	23.29%

Un par de ejemplos de errores cometidos por el modelo *zero-shot* y corregidos por DistilBERT:

- *"lo del olwas on display no funciona en los 23 ultra (americano no estoy seguro si en los demas paises si funciona)"* pred=animales, label=tecnología
- *"nunca he usado llave dinamometrica y nunca he tenido ningun p...ta llegar a media vuelta antes de partir. nunca me falla jaja"* pred=viajes, label=vehículos

En los ejemplos vemos como el modelo *zero-shot* no está alineado con nuestro espacio de etiquetas ni con la jerga real del dataset, así que toma atajos superficiales y falla ante comentarios con mucho ruido. Por ejemplo, en *"lo del always on display no funciona en los 23 ultra..."* el modelo *zero-shot* no asocia anglicismos, faltas y nombres de modelo con tecnología y acaba prediciendo una clase irrelevante; y en *"nunca he usado llave dinamométrica..."* confunde verbos de movimiento (*"llegar"*, *"vuelta"*) con viajes, ignorando que *"llave dinamométrica"* es un marcador claro de vehículos/mecánica. En cambio, el fine-tuning de DistilBERT permite aprender estas correlaciones específicas del dominio y, por tanto, mejora la clasificación al adaptar el modelo a cómo se expresa realmente la información en nuestros comentarios.

Un par de ejemplos de errores cometidos por BETO y corregidos por DistilBERT:

- *"que humilde lewandoski,veia mucho rapido y furioso"* pred=cine, label=deportes
- *"chris coghlan no piso la tercera base, pero nadie se dio cuenta, por tal motivo la carrera fue valida"* pred=cine, label=deportes

En el primer ejemplo vemos como BETO probablemente utiliza *"rapido y furioso"* como señal de la clase cine. Una hipótesis razonable de por qué DistilBERT multilingüe es más robusto en este caso es que este ha visto más nombres propios internacionales y variantes en su preentrenamiento. Similarmente, en el segundo ejemplo quizás BETO falla debido a la presencia del nombre *"chris coghlan"*, muy anglosajón, y por lo tanto un tipo de nombre peor representado en un corpus de textos en español. Puede que influya también que el deporte en cuestión sea béisbol, que de nuevo estaría poco representado en un corpus español en comparación con uno multilingüe.

5.4. Interpretabilidad del modelo *Transformer*

Los fallos en viajes y las confusiones sutiles entre clases vecinas, incluso en el modelo más avanzado, requieren una explicación que vaya más allá del resultado final (la predicción). Es necesario saber qué partes de un comentario ambiguo están impulsando la predicción errónea y cómo el modelo de atención está relacionando los tokens para entender realmente las limitaciones del modelo (o las intrínsecas a la tarea). Por lo tanto, el siguiente paso del análisis es utilizar técnicas de interpretabilidad, como SHAP y BertViz, para abrir la *"caja negra"* del *Transformer* y estudiar el mecanismo interno de sus decisiones en los casos de éxito (donde corrigió al *baseline*) y en los casos de fallo (los errores mutuos o los errores únicos).

El modelo *baseline* TF-IDF + Regresión Logística es básicamente transparente, ya que sus decisiones se basan en una combinación lineal de pesos léxicos (los coeficientes de la regresión aplicados a los pesos TF-IDF). Sin embargo, el *Transformer* es una "caja negra" cuyas predicciones emergen de interacciones altamente no lineales entre miles de parámetros y múltiples capas de atención, lo que no nos permite interpretar directamente con factores lingüísticos claros. Por ello, recurrimos a estas técnicas para obtener una explicación de los fallos y aciertos, utilizando precisamente los patrones de error y los casos críticos (fallos únicos y mutuos) identificados en la sección anterior.

5.4.1. Explicabilidad externa: SHAP

Análisis global de tokens con mayor contribución a la clasificación

La tabla 5.9 muestra la contribución media global, positiva o negativa, de los tokens más contribuyentes en el conjunto de test para la predicción de cada una de las nueve clases en las que se clasifican los comentarios. Esta primera visión de la "caja negra" del *Transformer* ya nos permite hacer una serie de observaciones muy interesantes.

La primera observación es que los tokens más influyentes, tanto positiva como negativamente, son a menudo subpalabras o fragmentos (**jes**, **ura**, **hon**, **cum**). Esto se debe a que DistilBERT utiliza un tokenizador basado en subpalabras, lo que significa que el modelo aprende a asignar importancia a partes de las palabras y no solo a la palabra completa, haciendo que la interpretación directa sea más compleja que en el caso del *baseline* TF-IDF. El *Transformer* ajustado opera bajo un principio de exclusión contextual (valores negativos fuertes) además de la inclusión léxica, ya que el éxito en la clasificación se basa no solo en la presencia de tokens relevantes, sino en la ausencia de tokens pertenecientes a clases competidoras fuertes.

El análisis SHAP global confirma y explica el alto rendimiento cuantitativo observado en las clases más sólidas del modelo (deportes F1 0.75 y música F1 0.65):

- **Música:** La clase música está dominada por tokens con contribuciones positivas excepcionalmente altas, como **temas** (0.60), **canto** (0.58), **cantar** (0.40) y **musica** (0.30). Esto sugiere que el modelo ha internalizado un léxico musical muy robusto y específico. Los principales tokens negativos son términos asociados a deportes (**equipo**, **jugador**) y términos de entusiasmo genérico (**jes**, **hon**, **ja**).
- **Deportes:** Muestra una separación temática casi binaria. Los tokens positivos son extremadamente fuertes (**equipo** 0.80, **deporte** 0.58, **jugador** 0.25), mientras que los tokens negativos son, abrumadoramente, el léxico de música (**sound**, **musica**, **tema**, **cantante**, **canto**).

El análisis también nos permite ofrecer una explicación de por qué la clase viajes fue el punto débil del *Transformer* (F1 0.50):

- **Baja capacidad de anclaje positivo:** Los tokens con mayor contribución positiva en viajes (**arriva, conocer, pais, horas**) tienen valores SHAP relativamente bajos (máximo 0.075). Esto contrasta fuertemente con los valores de 0.80 en deportes o 0.60 en música.
- **Alta interferencia negativa:** Los tokens que empujan la predicción lejos de viajes son extremadamente fuertes, destacando **deporte** (-0.270) y **canto** (-0.155).

Esto confirma la hipótesis del análisis de errores: los comentarios de viajes son altamente susceptibles a ser mal clasificados porque su léxico temático positivo es demasiado débil y es fácilmente interferido por los vocabularios dominantes y mejor definidos de las categorías más fuertes. El modelo tiene que "luchar" mucho para encontrar evidencia positiva en viajes y es muy sensible a cualquier rastro léxico de otras clases.

El análisis también confirma patrones que subyacen a las confusiones temáticas observadas en la matriz de confusión:

- **Ciencia y Cine:** Ambas clases tienen una mezcla de términos positivos abstractos o fragmentados (**ura, principio, real** en ciencia; **dra, anima, final, historia** en cine). Esto sugiere que la clasificación no depende de un léxico técnico obvio, sino de un contexto narrativo o explicativo, haciéndolas vulnerables al solapamiento léxico entre sí o hacia noticias.
- **Tecnología y Vehículos:** Observamos una presencia positiva de tokens relacionados con capacidad o funcionalidad en ambas (**potencia, ens, loc, cualquier**). Esto explica la dificultad sutil que tiene el *Transformer* para separarlas (un error de "sobre-refinamiento" de *embeddings* cercanos), como se notó en los errores únicos del *Transformer* (por ejemplo, confundir hardware con vehículos).

Una última observación que podemos hacer es la de tokens que contribuyen positivamente a múltiples clases, lo que indica que el *Transformer* los utiliza como marcadores de interacción genérica dentro del dominio de YouTube, en lugar de marcadores temáticos. Por ejemplo, el fragmento **ja** (risas) es positivo en animales, ciencia, viajes y tecnología. De manera similar, **gracias** es positivo en viajes y tecnología, confirmando que los comentarios de agradecimiento genérico siguen teniendo una leve contribución positiva en el *Transformer*, aunque con valores bajos.

Análisis de ejemplos específicos

Además de analizar los valores SHAP globales calculados sobre todo el conjunto de test, analizar ejemplos específicos nos permite validar, mediante la explicabilidad local de SHAP, los patrones de comportamiento del Transformer ya establecidos en el análisis de errores y en el análisis global SHAP. Para ello utilizaremos dos comentarios estratégicos: un error de ambigüedad léxica corregido por el *Transformer*, y un error mutuo que ilustra los límites intrínsecos de la tarea.

El comentario *"el gato de schrodinger lo llevo escuchando desde que era pequeno"* fue clasificado correctamente como ciencia por el *Transformer*, superando el error del *baseline* TF-IDF, que lo clasificó como animales debido al alto peso léxico del término *"gato"*. El análisis SHAP local (la figura 5.5 muestra la explicación SHAP tanto para la clase correcta, (arriba) como para la incorrecta predecida por el *baseline* (abajo)) demuestra cómo la contextualización del *Transformer* opera activamente para corregir este fallo basado en polisemía.



Figura 5.5: Explicación SHAP para comentario clasificado correctamente por el *Transformer* e incorrectamente por el *baseline*

La explicación para la clase correcta muestra una alta confianza en la clase ciencia ($f(x) = 0.71$). Los tokens clave para impulsar la predicción positiva son los fragmentos que componen el término *"schrodinger"* (*rod*, *inger*, etc.) junto con tokens de contexto amplio como *desde* y *pequeno*. Esto demuestra que el modelo utiliza el conocimiento de que *"gato"* es un término científico en este contexto. Por otro lado, si nos fijamos en la explicación para la clase con la que el *baseline* tenía su confusión, animales, vemos que la confianza en esta clase es extremadamente baja ($f(x) = 0.02$). Crucialmente, el token *ga* (fragmento de *"gato"*) no solo no impulsa la predicción, sino que aporta una contribución negativa neta (empuje a la izquierda). Estas explicaciones validan que el *Transformer* actúa como un "lingüista que entiende el contexto", ya que el mecanismo de atención de DistilBERT permitió que el contexto (*"schrodinger"*) suprimiera activamente el peso léxico de la palabra *"gato"* en su significado animal, evitando así la clasificación errónea que era inevitable para el modelo lineal TF-IDF.

El comentario *"buena musica la escucho mientras estoy en el bus"* fue clasificado erróneamente como música por ambos modelos, representando un error mutuo característico de las dificultades intrínsecas de la clase viajes, a la que pertenece el comentario. El análisis SHAP local (la figura 5.6 muestra la explicación SHAP tanto para la clase correcta (arriba) como para la incorrecta predecida por el *Transformer* (abajo)) confirma la hipótesis de la interferencia de clases fuertes establecida en el análisis global.

La explicación para la clase música muestra una confianza dominante en esta ($f(x) = 0.82$). El token *musica* proporciona una contribución positiva masiva hacia esta clase, y fragmentos como *buena* y *escucho* amplifican el léxico musical. En contraste, el token



Figura 5.6: Explicación SHAP para comentario clasificado erróneamente por ambos modelos

bus (la ancla temática de viajes) tiene una contribución negativa muy pequeña. Por otro lado, la explicación para la clase viajes muestra que la confianza en la clase es casi nula ($f(x) = 0.15$). A pesar de la presencia de **bus** y la preposición **en** (marcador de ubicación/viaje), estos tokens ofrecen una contribución positiva débil. Por el contrario, el token **musica** es el principal impulsor de la predicción lejos de viajes (contribución negativa). Este ejemplo valida que la debilidad léxica de la clase viajes (cuyos tokens positivos son globalmente bajos, máx. 0.075) la hace incapaz de competir con la fuerza de la señal de clases dominantes como música. El *Transformer*, aunque contextual, no puede superar la ambigüedad inherente cuando una señal temática fuerte, aunque secundaria al contexto ("escucho música mientras estoy en el bus"), se encuentra con un ancla contextual débil para la clase real. Este problema que nos encontramos con esta clase es indicativo de una limitación mayor, asociada a la naturaleza de la tarea y no tanto a las características de la arquitectura *Transformer*.

5.4.2. Visualización interna: BertViz

Para complementar la explicabilidad externa de SHAP, aportaremos una visualización interna directa de los mecanismos de atención del *Transformer* mediante BertViz. Utilizaremos la vista *model*, que proporciona una perspectiva global, visualizando simultáneamente todas las capas y cabezas de atención, y permitiendo identificar qué capas o cabezas son más activas o relevantes para el comentario analizado en el proceso de construcción de la decisión. Una vez escogidas estas capas/cabezas, la vista *head* nos servirá para visualizar las relaciones explícitas entre tokens de entrada y salida (útil para observar patrones lingüísticos concretos como dependencias sintácticas o concordancias locales).

El comentario seleccionado para este análisis interno es "*no estas viendo una territory sino una chinitory jajaja*", un caso especialmente interesante de detección de contexto invertido y jerga. El modelo *Transformer* lo clasificó correctamente como vehículos, superando el error del *baseline* que predijo cine. Analizar este ejemplo nos permitirá ver cómo la atención resuelve la negación y el sarcasmo que son invisibles para la Regresión Logística.

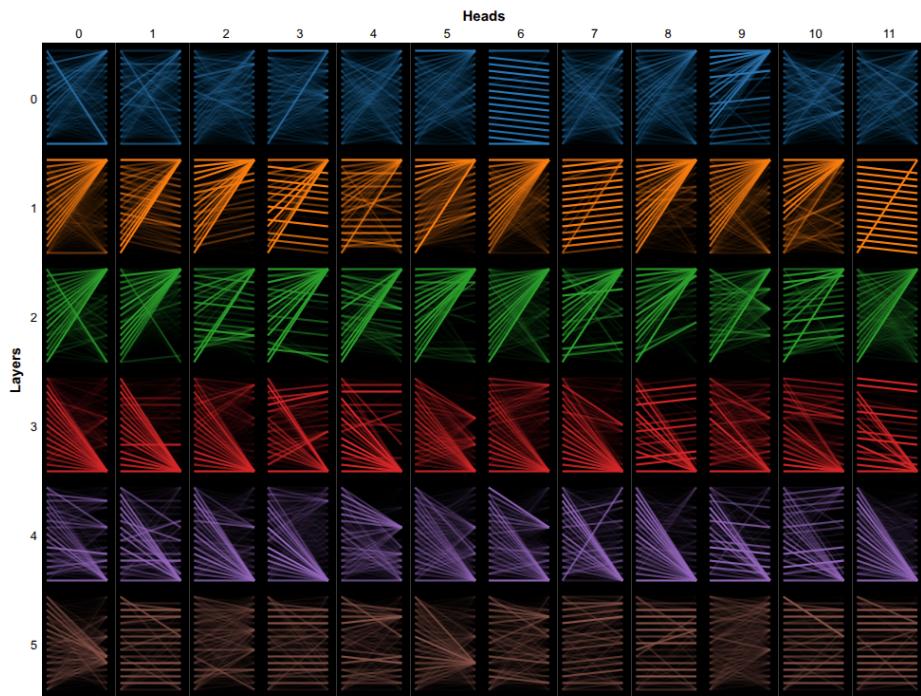


Figura 5.7: Vista *model* para comentario sarcástico

La Figura 5.7 (vista *model*) revela la distribución de los pesos de atención a lo largo de las seis capas del modelo DistilBERT. Observamos una clara transición en el comportamiento de la atención a medida que la información avanza. Las capas inferiores (0 y 1) presentan patrones de atención altamente difusos y distribuidos, lo cual es típico de la fase inicial donde el modelo captura dependencias sintácticas y co-ocurrencias locales sin una focalización semántica profunda. En contraste, las capas superiores (4 y 5) muestran patrones de atención notablemente más focalizados y selectivos, con líneas de atención concentradas en un número menor de tokens. Esta concentración indica que estas capas profundas son las responsables de extraer las señales temáticas y contextuales decisivas para la clasificación final.

Tras analizar la atención en varias de las cabezas en distintas capas, hemos encontrado ejemplos esclarecedores de cómo el *Transformer* resuelve la clasificación del comentario. La figura 5.8 revela el mecanismo exacto para interpretar la jerga "chinitory". Esta cabeza en la capa 5 (profunda) ejecuta dos funciones críticas: reconstruye la subpalabra, mostrando fuertes enlaces internos entre los fragmentos del neologismo (chinitory) y los fragmentos del término original (territory), asegurando que la invención conserva la representación contextual del vehículo Territory, y también valida el contraste, ya que se observa una atención focalizada que vincula los tokens temáticos (chinitory y territory) con el token **sino**, confirmando que esta cabeza está interpretando la relación de contraste gramatical ("no esto sino aquello").

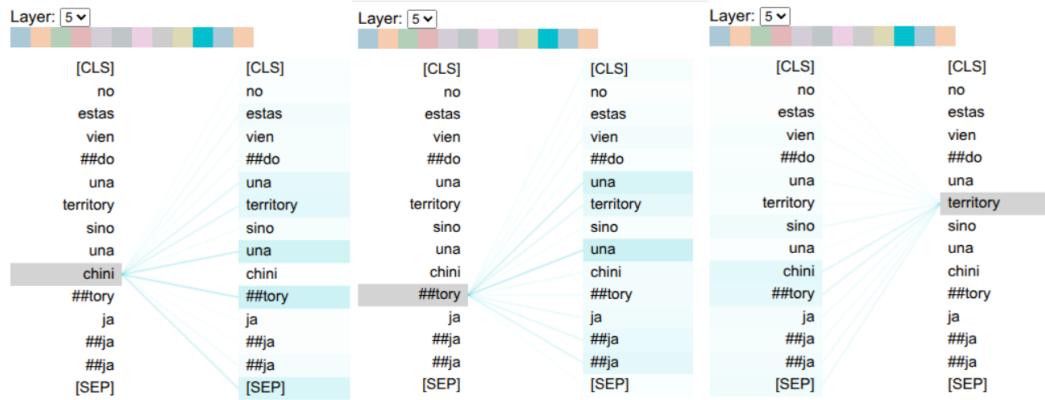


Figura 5.8: Vista *head* de la novena cabeza, última capa para comentario sarcástico

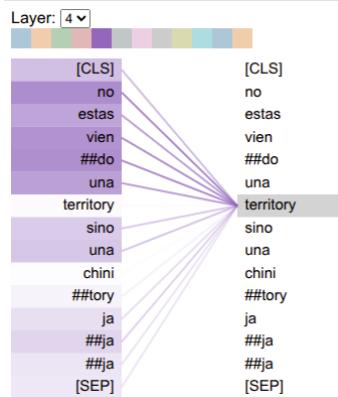


Figura 5.9: Vista *head* de la cuarta cabeza, penúltima capa para comentario sarcástico

En la figura 5.9 vemos cómo el token **territory** recibe una atención masiva de los tokens de entrada, crucialmente, proveniente de los elementos que definen la negación y el contexto verbal: el token de negación **no**, y los fragmentos verbales **vien** y **##do** (que componen "viendo"). Esta concentración de atención demuestra que esta cabeza en particular está dedicada a vincular la acción de la frase ("no estás viendo") al objeto principal de la negación/contraste ("territory"). Este proceso de anclaje es un paso intermedio que permite al modelo confirmar que la frase trata semánticamente sobre el vehículo (**territory**), y preservar el modificador contextual (**no**) junto con el sujeto para que las capas superiores (capa 5) puedan utilizar esa información y resolver el contraste explícito introducido por el token **sino** (como se observó en el análisis de la capa 5, cabeza 9). Sin esta dependencia fuerte en la capa 4, el modelo podría haber perdido la conexión entre la negación y el sujeto, dificultando la corrección del error del modelo lineal.

La figura 5.10 ilustra la consolidación final de la decisión. Esta cabeza se especializa en conectar la representación contextualizada de la frase con el token de clasificación

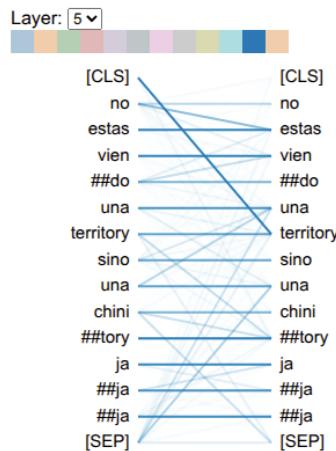


Figura 5.10: Vista *head* de la décima cabeza, última capa para comentario sarcástico

final, [CLS] (el vector utilizado para la predicción). Observamos una fuerte atención que sale del token [CLS] y se dirige al token **territory**, así como atención recíproca.

Este vistazo a la atención interna del modelo a la hora de clasificar un comentario semánticamente complejo (humor, negación) demuestra que el *Transformer* no clasifica el comentario basándose simplemente en palabras clave individuales (como haría el baseline), sino que utiliza cabezas de atención en capas profundas (Capas 4 y 5) para realizar tareas lingüísticas complejas, como enlazar la negación y el contraste con el sujeto principal y reconstruir el significado de neologismos basados en jerga humorística.

Tabla 5.9: Top 10 tokens con contribución SHAP (positiva/negativa) para cada clase

Animales		Ciencia		Cine		Deportes		Música	
Positivos (Top-10)	Negativos (Top-10)								
jes	0.075	ura	0.105	dra	0.245	equipo	0.80	temas	0.60
per	0.058	principio	0.023	anima	0.170	deporte	0.58	canto	0.58
ter	0.052	loc	0.013	final	0.150	jugador	0.25	cantar	0.40
ro	0.041	hon	0.011	jan	0.135	narra	0.25	temi	0.35
nur	0.036	ja	0.010	ada	0.125	juego	0.095	cantante	0.33
sa	0.034	real	0.010	ale	0.102	chim	0.070	musica	0.30
apa	0.033	ntarios	0.009	arte	0.075	potencia	0.065	tema	0.28
ee	0.031	del	0.009	pta	0.070	cha	0.060	cantando	0.26
hon	0.029	gg	0.008	horas	0.070	chi	0.058	sound	0.26
dri	0.027	ada	0.008	stra	0.068	ilidad	0.055	cum	0.25
Noticias		Tecnología		Vehículos		Viajes			
Positivos (Top-10)	Negativos (Top-10)								
droga	0.095	jes	0.052	jugador	0.052	arriva	0.075		
potencia	0.080	j	0.032	jes	0.027	conocer	0.070		
dieron	0.055	ens	0.031	hon	0.021	mir	0.058		
parce	0.032	hon	0.030	potencia	0.020	frase	0.056		
deros	0.025	ada	0.027	maestro	0.019	horas	0.052		
bak	0.020	uu	0.025	ntarios	0.018	textos	0.051		
mede	0.019	arriva	0.024	cualquier	0.017	pais	0.049		
maestro	0.018	loc	0.021	uu	0.016	10	0.045		
ros	0.017	ja	0.021	ura	0.016	narra	0.043		
chim	0.016	gg	0.019	ens	0.014	jes	0.040		
Noticias		Tecnología		Vehículos		Viajes			
Positivos (Top-10)	Negativos (Top-10)								
cantar	-0.145	pier	-0.145	temas	-0.055	deporte	-0.270		
equipo	-0.105	temas	-0.100	can	-0.048	canto	-0.155		
musica	-0.090	temi	-0.095	cantante	-0.030	temas	-0.060		
final	-0.060	cantante	-0.045	temi	-0.025	rom	-0.058		
rr	-0.040	cantando	-0.042	equipo	-0.023	cantando	-0.056		
temas	-0.038	canto	-0.040	musica	-0.022	equipo	-0.055		
bar	-0.036	can	-0.038	canto	-0.021	cantar	-0.050		
tema	-0.034	droga	-0.032	rape	-0.020	rr	-0.045		
cantante	-0.028	dra	-0.030	dra	-0.019	dra	-0.044		
canto	-0.027	equipo	-0.028	omb	-0.018	jugador	-0.043		

6. Conclusiones

En este trabajo nos hemos propuesto el objetivo de desarrollar un sistema de clasificación temática de comentarios de YouTube mediante técnicas avanzadas y estado del arte del Procesamiento del Lenguaje Natural y el *Machine Learning*. Para ello, hemos implementado un *pipeline* integral, empezando por la obtención automática de los comentarios a través de la API de YouTube, pasando por un riguroso proceso de filtrado basado en reglas y métricas estadísticas, y acabando con el entrenamiento de un modelo basado en la arquitectura *Transformer*. El resultado es un modelo funcional basado en DistilBERT multilingüe, capaz de trabajar con el lenguaje informal y ambiguo de los comentarios. Además de ese modelo, hemos entrenado un modelo *baseline* basado en la combinación clásica de TF-IDF para la representación del texto y Regresión Logística, y también hemos utilizado un modelo *zero-shot*. A partir de ellos hemos realizado un análisis comparativo tanto cuantitativo (métricas) como cualitativo (fijándonos en los errores cometidos y utilizando técnicas de interpretabilidad).

El análisis comparativo nos ha permitido abordar la tarea de justificar el uso de un modelo computacionalmente costoso como lo es el *Transformer* frente a técnicas más sencillas. Por un lado, la mejora respecto al modelo *zero-shot* es evidente: observamos un aumento del F1 de casi 20 puntos porcentuales, justificando así plenamente el proceso de *fine-tuning* a la hora de usar estos modelos. Este es especialmente necesario cuando queremos aplicar el conocimiento general del lenguaje proveniente del pre-entrenamiento a un dominio específico y complejo como lo es los comentarios de YouTube. Por otro lado, el éxito del *Transformer* sobre el *baseline* TF-IDF es más moderado, con una mejora de menos de 4 puntos porcentuales. Sin embargo, el análisis de los errores confirma que el modelo lineal depende de palabras clave, mientras que el *Transformer*, gracias al mecanismo de atención, tiene capacidad para capturar relaciones contextuales y no lineales, lo cual resulta crítico para el éxito en dominios complejos como el digital.

Finalmente se realizó un análisis de interpretabilidad de nuestro modelo DistilBERT. Este se ha enfocado desde dos ángulos: interpretabilidad externa mediante el método SHAP, e interpretabilidad interna mediante la herramienta BertViz. El método de explicabilidad local SHAP nos ha permitido comprobar la importancia que atribuye el modelo *Transformer* a las palabras dentro de cada comentario, validando así que el peso de un token no depende únicamente de su frecuencia o características léxicas, sino del contexto en el que aparece dentro del texto. Pasando a la interpretabilidad interna, hemos generado una serie de visualizaciones internas de los mecanismos de atención del modelo mediante BertViz. Esto ha demostrado que el modelo no infiere al azar,

sino que entiende fenómenos como el sarcasmo, la negación y el uso de neologismos, lo cual se refleja directamente en la forma en la que se disparan las cabezas de atención en las múltiples capas del modelo, siendo las capas más profundas las encargadas de resolver las ambigüedades lingüísticas más complejas.

La mayor limitación que se ha encontrado a partir del análisis reside en el solapamiento semántico de las categorías definidas. La visualización PCA ya nos mostraba que el espacio vectorial no presentaba una separación clara y sencilla de aprender por un modelo, y esto se ve reflejado claramente en resultados como los de la clase viajes o errores de clasificación en los que el contenido del comentario era lo suficientemente ambigüo como para pertenecer razonablemente a más de una categoría. La alta proporción de errores cometidos tanto por el *Transformer* como por el modelo *baseline* sugiere que existe un límite superior a lo que podemos conseguir refinando más el entrenamiento o incluso con arquitecturas más potentes. Se trata más bien de un problema de ruido residual inevitable en los comentarios de YouTube, además de un enfoque seguramente subóptimo en cuanto a maximizar resultados se refiere.

Por lo tanto, este trabajo deja amplio espacio para evolucionar el sistema en un futuro. Más allá de simplemente probar con otros modelos o variar la configuración de entrenamiento, los resultados sugieren que merece más la pena trabajar en la parte de etiquetado y procesado de los datos. Por ejemplo, una posible mejora sería la de implementar un sistema de clasificación multietiqueta para gestionar la presencia de comentarios que traten varios temas a la vez. También sería interesante refinar el dataset de manera más específica para las clases más débiles, o incluir en este metadatos adicionales para mejorar el contexto del comentario. Por último, cabe plantearse el pivotar de un enfoque de clasificación supervisada a uno no supervisado, en el que se aproveche la potencia del mecanismo de atención para generar *embeddings* que separen en *clusters* los comentarios de manera automática, de manera que se genere un conjunto de temáticas emergente en vez de limitarnos a uno prefijado.

Bibliografía

- [1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30:5998–6008, 2017.
- [2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018. URL <https://arxiv.org/abs/1810.04805>.
- [3] Enja Kokalj, Blaž Škrlj, Nada Lavrač, Senja Pollak, and Marko Robnik-Šikonja. BERT meets shapley: Extending SHAP explanations to transformer-based classifiers. In Hannu Toivonen and Michele Boggia, editors, *Proceedings of the EACL Hackashop on News Media Content Analysis and Automated Report Generation*, pages 16–21, Online, April 2021. Association for Computational Linguistics. URL <https://aclanthology.org/2021.hackashop-1.3/>.
- [4] Jose Camacho-Collados and Mohammad Taher Pilehvar. On the role of text preprocessing in neural network architectures: An evaluation study on text categorization and sentiment analysis. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 40–46, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5406. URL <https://aclanthology.org/W18-5406/>.
- [5] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models*. 3rd edition, 2025. URL <https://web.stanford.edu/~jurafsky/slp3/>. Online manuscript released August 24, 2025.
- [6] Serkan Sarica and Jianxi Luo. Stopwords in technical language processing. *PLoS One*, 16(8):e0254937, August 2021. doi: 10.1371/journal.pone.0254937. URL <https://doi.org/10.1371/journal.pone.0254937>. Erratum published in PLoS One, 2024, 19(12): e0315195.
- [7] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, 2008. URL <https://nlp.stanford.edu/IR-book/>. Capítulo 6 utilizado como referencia.

- [8] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006. URL <https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf>. Capítulos 3 y 4 utilizados como referencia.
- [9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Regularization for Deep Learning*, chapter 7. MIT Press, 2016. URL https://mcube.lab.nycu.edu.tw/~cfung/docs/books/goodfellow2016deep_learning.pdf.
- [10] Daiki Nishigaki, Yuki Suzuki, Tomohiro Wataya, Kosuke Kita, Kazuki Yamagata, Junya Sato, Shoji Kido, and Noriyuki Tomiyama. Bert-based transfer learning in sentence-level anatomic classification of free-text radiology reports. *Radiology: Artificial Intelligence*, 5(2):e220097, 2023. doi: 10.1148/ryai.220097. URL <https://pmc.ncbi.nlm.nih.gov/articles/PMC10077075/>.
- [11] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013. URL <https://arxiv.org/abs/1301.3781>.
- [12] Bradlhy Luis Machado Medina, César Alonso Santillana Quirita, and Shar-melyn Violeta Bautista Luque. Classification of news categories using bert. *Innovation and Software*, 4(2):36–51, Sep. 2023. doi: 10.48168/innosoft.s12.a98. URL <https://n2t.net/ark:/42411/s12/a98>.
- [13] Joe Davison. Zero-shot learning: An overview. Joe Davison Blog, 2020. URL <https://joeddav.github.io/blog/2020/05/29/ZSL.html>.
- [14] Mohit Tuteja and Daniel González Juclà. Long text classification using transformers with paragraph selection strategies. In Daniel Preoțiuc-Pietro, Catalina Goanta, Ilias Chalkidis, Leslie Barrett, Gerasimos Spanakis, and Nikolaos Aletras, editors, *Proceedings of the Natural Legal Language Processing Workshop 2023*, pages 17–24, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.nllp-1.3. URL <https://aclanthology.org/2023.nllp-1.3/>.
- [15] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.
- [16] Google Developers. Youtube data api v3 documentation. Google Developers Documentation, 2024. URL <https://developers.google.com/youtube/v3/docs>.

- [17] Google Developers. Determining quota costs. Google Developers Documentation, 2024. URL https://developers.google.com/youtube/v3/determine_quota_cost.
- [18] Google Developers. Youtube data api: Search.list. Google Developers Documentation, 2024. URL <https://developers.google.com/youtube/v3/docs/search/list>.
- [19] Google Developers. Youtube data api: Videos.list. Google Developers Documentation, 2024. URL <https://developers.google.com/youtube/v3/docs/videos/list>.
- [20] Google Developers. Youtube data api: CommentThreads.list. Google Developers Documentation, 2024. URL <https://developers.google.com/youtube/v3/docs/commentThreads/list>.
- [21] Steven Bird, Ewan Klein, and Edward Loper. Natural language toolkit (nltk). NLTK Project Website, 2024. URL <https://www.nltk.org/>.
- [22] YData. Ydata profiling documentation. YData Documentation Website, 2024. URL <https://docs.profiling.ydata.ai/latest/>.
- [23] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: A review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, April 2016. doi: 10.1098/rsta.2015.0202. URL <https://doi.org/10.1098/rsta.2015.0202>.
- [24] Sentence Transformers. distiluse-base-multilingual-cased-v2 model card. Hugging Face Model Repository, 2024. URL <https://huggingface.co/sentence-transformers/distiluse-base-multilingual-cased-v2>.
- [25] Marco Lui and Timothy Baldwin. langid.py: An off-the-shelf language identification tool. In *Proceedings of the ACL 2012 System Demonstrations*. Association for Computational Linguistics, 2012.
- [26] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2nd edition, 2006.
- [27] scikit-learn developers. LogisticRegression — scikit-learn documentation (v1.8.0). https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.
- [28] Hugging Face. Zero-shot classification pipeline — transformers documentation, 2025. URL <https://huggingface.co/tasks/zero-shot-classification>. Consultado en línea.

- [29] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019. URL <https://arxiv.org/abs/1910.01108>.
- [30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Sequence Modeling: Recurrent and Recursive Nets*, chapter 10. MIT Press, 2016. URL https://mcube.lab.nycu.edu.tw/~cfung/docs/books/goodfellow2016deep_learning.pdf.
- [31] Jesse Vig. Visualizing attention in transformer-based language representation models. arXiv preprint arXiv:1904.02679, 2019. URL <https://arxiv.org/abs/1904.02679>.
- [32] José Cañete, Gabriel Chaperon, Rodrigo Fuentes, Jou-Hui Ho, Hojin Kang, and Jorge Pérez. Spanish pre-trained bert model and evaluation data. In *PML4DC at ICLR 2020*, 2020.
- [33] Hugging Face. Sentence transformers documentation. Hugging Face Hub Documentation, 2024. URL <https://huggingface.co/docs/hub/en/sentence-transformers>.
- [34] Pummy Dhiman, Amandeep Kaur, Deepali Gupta, Sapna Juneja, Ali Nauman, and Ghulam Muhammad. Gbert: A hybrid deep learning model based on gpt-bert for fake news detection. *Helijon*, 10(16):e35865, 2024. ISSN 2405-8440. doi: <https://doi.org/10.1016/j.heliyon.2024.e35865>. URL <https://www.sciencedirect.com/science/article/pii/S2405844024118968>.

A. Ejemplos de errores por modelo

Este anexo recopila ejemplos informativos de comentarios erróneamente clasificados, ya sean errores cometidos por el *baseline* y corregidos por el *Transformer*, viceversa, o errores cometidos por ambos modelos. Dentro de cada uno de esos conjuntos, se agrupan los ejemplos por patrones identificados en el análisis de los errores.

Errores del *baseline* corregidos por el *Transformer*

Corrección de polisemia / ambigüedad léxica

- *el gato de schrodinger lo llevo escuchando desde que era pequeno* pred=animales, label=ciencia
- *monologo auto reflexivo...* pred=vehiculos, label=ciencia
- *raperos... sueltan una pista...* pred=deportes, label=musica
- *en baterias y tecn... muy bonito el twingo* pred=tecnologia, label=vehiculos
- *llave dinamometrica... hasta llegar a media vuelta* pred=viajes, label=vehiculos
- *se necesita un compas una escuadra y un cartabon* pred=vehiculos, label=ciencia

Expresiones idiomáticas / contexto invertido

- *no estas viendo una territory sino una chinitory jajaja* pred=cine, label=vehiculos
- *si claro... a nada que tomes una curva volcara* pred=ciencia, label=vehiculos
- *que espectacular jugada... tiempo ejecucion y distancia claro ok jajajajaj* pred=tecnologia, label=deportes
- *empezaron las excusa... esta novela continuara* pred=noticias, label=deportes
- *ironicamente diria que fue el viento... proyectil* pred=deportes, label=cine

Errores del *Transformer* corregidos por el *baseline*

Sobre-sensibilidad a jerga / tokens raros

- *nombre de la pelicula: misfit... disponible en hbo max... xdd* pred=**tecnologia**, label=cine
 - *yo tengo la asus tuf f15 i7 13620h rtx 4060 16gb ram* pred=**vehiculos**, label=tecnologia
 - *yo tengo jacks en chikengun pero com happymod* pred=viajes, label=tecnologia
 - *dtbm bro laborcaaaa zazzzz... saludos desde augusta ga* pred=viajes, label=tecnologia
 - *le han dado con de todo y seguia el infeliz xd* pred=deportes, label=cine

Confusión temática sutil (clases vecinas)

Ciencia → Tecnología

- *transistor cuantico... qbits*
 - *informacion sobre superconductores*
 - *como los algoritmos condicionan el consumo de contenido digital*

Tecnología ↔ Vehículos

- *una go pro me hace lo mismo y me ahorro una lana*
 - *que opinan del jac js6 me lo quiero comprar*
 - *una transmision dct en un chino es buena idea*

Cine ↔ Música

- *freddy sera un icono de la musica*
 - *esa escena quedo bella yo amo a kate y leo*

Deportes ↔ Noticias

- *que humilde ribery diciendo que son arregladas*
 - *que pasa con la fifa que verguenza*

Errores mutuos

Ambigüedad temática extrema

- *excelente video gracias*
- *excelente informacion gracias*
- *saludos desde salta argentina*
- *nuevo seguidor es mi primer video*

Clase viajes como clase problemática

- *excelente seleccion musical*
- *buenas musica la escucho mientras estoy en el bus*
- *la novela mejor deberia estar en netflix*
- *que hermosa cancion le dedico lucero a su hija*

Contenido multitema / ruido residual

- Comentario largo técnico sobre tanques: *(segun lo que se yo) el tiger se movio por que los sherman hicieron disparos del humo no podia parar, ya que si lo hacia lo iban a dejar cegados y rodearlo, me parece que lo que queria hacer fury era rodearlo lo mas que podia para no dar pelea frontalmente sino por atras o lateralmente, ahora aunque el fury si podia penetrar el tanque en esos tiempos era realmente dificil para el tiger y los sherman disparar en sus puntos de penetracion mas debiles. y sobre el disparo lateral aunque sii habia una posibilidad grande de destruirlo pero me parece que tambien hay una oportunidad de que falle pequena pero hay.. ahora la pelea entre el tanque fury y el tiger supongo que por la lenta movilidad del tiger era complicado hacer un disparo certero y mas anadirle su pesado canon, y donde estaba en su espalda estaba en una posicion dificil de mantener sobre todo el apunte ya que si vemos la mira del tanque cuando se apunta en cercano dispara muy diferente que de lejos, a lo mejor una combinacion de sucesos hizo que el tanque tiger no tuviera oportunidad. la verdad solo no hay que olvidar que es solo una pelicula y aunque algunos de sus datos o registros son erroneos no necesario ofender a una persona por quererla mucho y apreciarla ya que recuerda que es una de las pocas peliculas que nos dan una pelea epica pero con detalles. Clase real cine.*
- Texto religioso extenso: *tema ezequiel 18: el alma que pecare morira. vino palabra a mi de jehova: ?que pensais ,los que usan este refran: los padres comieron las*

uvas acidadas y los hijos sufren la dentera ?. vivo yo ,dice jehova el senor ,que nunca mas diras este refran . he aqui que todas las almas son mias ; como el alma del padre ,asi el alma del hijo es mia ; el alma que pecare,esa morira . el hombre justo que haga justicia que no comiere animales ,ni alzare los ojos alas imagenes (idolos) ni violare la mujer de su projimo ,ni llegare ala mujer menstruosa . ni oprimier a ninguno ; que al deudor devolviere su prenda ,que no robe ,y diere pan al ambr,cubre al desnudo ,no presta a interes ni toma a usura ,hiciere juicio verdadero alos hombres ,caminare guardando mis mandamientos y ordenanzas rectamente , este es justo y vivira dice el senor. Clase real **noticias**.

- Discusión de hardware/consolas en vídeos etiquetados como Noticias: *chicos es una pc plug and play, potencia de pc con movilidad de laptop, para llevar de viaje, para meterla en tu living en una tele, para ir llevandola del comedor a tu pieza. no es el producto que ustedes estan esperando, le dan mucho hype a algo que ustedes no van a usar. no es tan dificil de entender el concepto, pero ustedes escuchan lo que quieren escuchar.* Clase real **noticias**.
- Relatos autobiográficos largos sin términos temáticos claros: *mi hermano y mi papa trabajaron desde chicos, mi papa mas chico y cuando tienen dias libres o es feriado, siguen trabajando, mi hermano se aburre en mi casa por no saber que hacer, y mi papa se aburria en los fines de semana,trabajaba hasta los feriados, jamas los entendi, lo que si es que ostentaban todo lo q compraban y no podian ahorrar mucho dinero, yo por ejemplo me decian como hacia para ahorrar dinero mientras ellos no podian,yo siempre tuve metas sobre el dinero,en que gastarlo, sin embargo ellos lo gastaban en estupideces, lo gastaban mas en autos, me acuerdo que mi papa se invento un verso terrible de como habia obtenido un auto para impresionar a sus amigos y yo les dije que era de mi hermano, mi papa teniendo casi 65 anos, cuando naces cn ese cerebro jamas cambias, y otra cosa, no sienten empatia,no hablan,no mantienen conversacion, en navidades estaban callados,no podian entablar vinculos, mi hermano gano sus noviecititas por sus autos y darles dinero, mi papa lo mismo, pero de tan autoritarios que eran por tenerlo, se creian mas y terminaban dejandolos, mi papa se murio con una interesada con 2 hijos y mi hermano solo sigue a los 50 anos, tuvo como 7 novias y todas las dejaron, y siempre o lo echaban de la casa de los padres o le decian que era mala persona, pero q va a cambiar una persona a esa edad, a mi nunca me intereso el dinero hasta grande con una enfermedad, lo unico q me interesa el dinero es para videojuegos y nada mas,debe ser por eso la capacidad de ahorrar, estoy satisfecho y dejo de gastar,me acuerdo que a los 12 anos queria la snes y me iba a trabajar con mi papa,pero porque tenia un objetivo, estos 2 infelices trabajan porque no tienen otra cosa que hacer, mi papa se aburria en casa lo mismo mi hermano, en sus vacaciones no sale de su dormitorio a pesar de tener un coche de 30 mil dolares.* Clase real: **ciencia**