Recomendador de revistas científicas mediante clasificación de texto

Imanol Benito Quintero Bermúdez

8 de enero de 2026

Resumen

Este trabajo presenta un sistema inteligente para recomendar la revista científica más adecuada a la que enviar un manuscrito a partir de su título, resumen (abstract) y palabras clave. El recomendador se entrena como un problema de clasificación multiclase usando un corpus de artículos publicados entre 2020 y 2024 en cuatro revistas de Elsevier (revistas 1, 2, 3 y 5), construido a partir de metadatos obtenidos de ScienceDirect. La metodología compara dos enfoques: (i) una aproximación clásica basada en procesamiento de lenguaje natural con representación TF-IDF y modelos supervisados implementados con Scikit-learn, y (ii) una aproximación conexionista implementada en PyTorch mediante una red neuronal simple con capa de embedding, agregación por media (mean pooling) y un perceptrón multicapa para predecir la revista. Para evaluar los modelos se emplea partición estratificada de entrenamiento y prueba y se reportan métricas de clasificación (accuracy, F1 macro y matriz de confusión), poniendo especial atención al desbalance entre clases. Los resultados muestran que el enfoque clásico con TF-IDF y un clasificador lineal ofrece un rendimiento sólido como línea base y proporciona un punto de comparación claro frente al modelo neuronal, además de permitir un análisis interpretable de los errores más frecuentes entre revistas temáticamente próximas.

Figura 1: ejemplos de las revistas científicas más importantes.

Índice

1. Introducción

1.1. Contexto y motivación

En un contexto de creciente globalización, donde la información y los resultados de investigación se difunden de forma prácticamente inmediata, resulta evidente el aumento exponencial de la producción científica, especialmente desde la segunda mitad del siglo XX. Diversos estudios estiman que la producción científica mundial ha crecido a un ritmo medio del 7,4 % anual, superando ampliamente el crecimiento del PIB global durante el mismo periodo. Asimismo, en 2022 se publicaron más de 5,14 millones de artículos académicos, incluyendo revisiones y actas de congresos, una cifra que continúa incrementándose año tras año. Este aumento sostenido en la tasa de producción científica ha sido reconocido desde hace décadas [8] [10] [2] [4] [9] [3]. Entre las causas que explican esta tendencia se encuentran el incremento del número de investigadores a nivel mundial [6], el consecuente aumento de descubrimientos susceptibles de ser comunicados a la comunidad científica, a la sociedad y a futuras generaciones [1], así como la creciente presión institucional para publicar, por ejemplo, como requisito para acceder o mantener posiciones postdoctorales y permanentes en el ámbito académico [7]. Como resultado, el volumen cada vez mayor de publicaciones ha conducido a una especialización creciente, dado que resulta prácticamente imposible, tanto para los científicos como para el público general, mantenerse al día con la totalidad de su área principal de conocimiento, y mucho menos con los avances en disciplinas afines o distantes [5].

En la actualidad existen miles de revistas científicas y, dentro de un mismo campo, decenas de opciones posibles para enviar artículos. Esta elección no depende solo de indicadores de calidad, sino también del encaje temático: publicar en una revista donde la mayoría de trabajos tratan temas cercanos aumenta la probabilidad de llegar a la audiencia adecuada y mejora la visibilidad del trabajo. Además, el proceso de selección suele hacerse de forma manual (revisando objetivos y alcance, números recientes o recomendaciones informales), lo que consume tiempo y puede introducir sesgos o decisiones poco sistemáticas.

Ante este escenario, resulta útil plantear un sistema automático que, a partir del contenido textual básico del manuscrito (título, resumen y palabras clave), sugiera la revista más compatible. Este tipo de herramienta puede servir como apoyo a la decisión: no reemplaza el criterio del autor, pero proporciona una recomendación rápida y reproducible basada en patrones aprendidos de publicaciones previas.

1.2. Objetivo del sistema

El objetivo de este trabajo es diseñar e implementar un modelo recomendador de revistas científicas formulado como un problema de clasificación de texto. El sistema recibe como entrada el título, el resumen (abstract) y las palabras clave de un artículo, y devuelve como salida la revista recomendada entre un conjunto de cuatro revistas candidatas (1, 2, 3 y 5). Para ello, se entrena un modelo supervisado utilizando artículos ya publicados en dichas revistas, de modo que el modelo aprenda a asociar patrones lingüísticos y terminología con cada revista.

1.3. Contribuciones

La primera contribución de este trabajo es la construcción de un corpus de artículos publicados entre 2020 y 2024 para las revistas 1, 2, 3 y 5. Para ello se recopilan metadatos textuales (título,

resumen y palabras clave) y se realiza un proceso de limpieza y normalización que permite disponer de un conjunto de datos homogéneo y apto para experimentación y evaluación.

En segundo lugar, se desarrolla un punto de partida sólido de la aproximación clásica mediante representación TF-IDF (matriz término-documento) y el entrenamiento de modelos de clasificación lineales en Scikit-learn. El objetivo de este escenario es establecer un punto de referencia robusto y reproducible, seleccionando el mejor modelo en función de la validación y las métricas definidas.

Como tercera aportación, se implementa una aproximación conexionista en PyTorch orientada a la clasificación de texto, con el fin de comparar un paradigma de aprendizaje profundo frente al enfoque clásico. Este modelo se entrena y evalúa con el mismo conjunto de datos y bajo el mismo protocolo experimental, para asegurar una comparación justa entre enfoques.

Además, se define un protocolo de evaluación reproducible que incluye partición estratificada de los datos, métricas por clase, F1 macro y el análisis mediante matrices de confusión, complementado con un análisis de errores. Este diseño permite no solo medir el rendimiento global, sino también identificar patrones de confusión entre revistas y comprender mejor el comportamiento del sistema.

Por último, se realiza una comparación cuantitativa y cualitativa entre los enfoques clásico y conexionista, discutiendo resultados, ventajas y limitaciones de cada propuesta. Esta discusión sirve también para proponer líneas de mejora y trabajo futuro, orientadas a incrementar el rendimiento y la aplicabilidad del recomendador en escenarios reales.

2. Datos

2.1. Fuente y criterios de recopilación

Para la implementación del recomendador se utiliza un conjunto de revistas de la editorial Elsevier, tomando como corpus los artículos publicados entre 2020 y 2024 en las revistas seleccionadas (revistas 1, 2, 3 y 5). Los datos se obtienen a través del portal ScienceDirect, accesible mediante la biblioteca de la universidad, y se recopilan los campos textuales necesarios para el problema: título, resumen (*abstract*) y palabras clave. La obtención de los registros puede realizarse exportando las referencias de cada número en formato RIS o BibTeX incluyendo el *abstract*, y posteriormente transformándolas a un formato estructurado para su procesamiento automático.

2.2. Estructura del conjunto de datos

El conjunto de datos final se organiza a nivel de artículo, de modo que cada instancia corresponde a un documento y su etiqueta es la revista de publicación. La estructura utilizada incluye, como mínimo, los siguientes campos: `journal_id` (identificador de la revista), `journal_name` (nombre), `year` (año de publicación), `title`, `abstract`, `keywords`, y un campo `text` construido por concatenación de `title`, `abstract` y `keywords` para alimentar los modelos de clasificación. Adicionalmente, se conservan identificadores como `doi` y el enlace `link` cuando están disponibles, ya que facilitan la trazabilidad y la detección de duplicados.

En total, el corpus procesado contiene 17 468 artículos distribuidos en cuatro clases (revistas 1, 2, 3 y 5), con una distribución desbalanceada entre revistas, aspecto que se tiene en cuenta en la evaluación mediante métricas por clase y F1 macro.

2.3. Preprocesado y limpieza

El preprocesado se orienta a asegurar la consistencia del corpus y a preparar la entrada textual de los modelos. En primer lugar, se filtran los registros incompletos eliminando aquellos artículos sin título o sin resumen, ya que el sistema debe basarse en información textual suficiente para poder recomendar una revista. En segundo lugar, se realiza una deduplicación para evitar que un mismo artículo aparezca varias veces en el entrenamiento: cuando está disponible se usa el DOI como identificador único, y en caso contrario se emplea una clave basada en (`title`, `year`, `journal_id`). Finalmente, se normalizan los tipos de datos (por ejemplo, el año a formato numérico y las palabras clave a una cadena de texto) y se genera el campo `text` que actúa como entrada común a los diferentes enfoques (clásico y conexionista).

2.4. Organización y generación del dataset

Los datos en bruto (*raw*) se organizaron en ficheros JSON siguiendo una estructura jerárquica por revista y año, con un fichero por cada par (revista, año) en el intervalo 2020–2024. Cada fichero contiene una lista de artículos con metadatos textuales (título, resumen y palabras clave), además de identificadores como DOI y enlace cuando están disponibles, lo que facilita la trazabilidad del corpus.

Posteriormente, se desarrolló un script de preprocesado (`build_dataset.py`) para unificar todos los JSON en un único conjunto de datos tabular (*processed*) en formato CSV (`dataset.csv`). Este script normaliza tipos (por ejemplo, el año), transforma las palabras clave a una cadena de texto y

construye un campo `text` concatenando título, resumen y palabras clave, que actúa como entrada común a los modelos. Adicionalmente, se aplica una limpieza mínima consistente en: (i) filtrado de registros sin título o sin resumen, y (ii) deduplicación preferente por DOI y, en su ausencia, por la tupla (título, año, revista).

2.5. Análisis descriptivo

Dado que el problema se formula como clasificación multiclase, resulta relevante analizar la distribución del corpus por revista y por año. Este análisis permite detectar desbalance entre clases, lo cual puede afectar a métricas agregadas como la *accuracy* y hace recomendable reportar métricas por clase y métricas balanceadas como F1 macro en la evaluación. Asimismo, la distribución temporal (2020–2024) permite comprobar que el conjunto de datos cubre de forma razonable el intervalo definido en el enunciado y reduce el riesgo de sesgos derivados de usar un único año de publicaciones.

Para cada revista, se observa un desbalance notable entre clases (cuadro 1), especialmente por la mayor representación de la revista 3, por lo que se reportan métricas por clase y F1 macro además de la *accuracy*.

Cuadro 1: Distribución del corpus por revista (clase).

Revista (journal_id)	**Nº artículos**	**% del total**
1 (Applied Ergonomics)	1083	6.20
2 (Neural Networks)	2331	13.35
3 (Expert Systems with Applications)	9589	54.90
5 (Pattern Recognition)	4465	25.55
Total	**17468**	**100.00**

3. Metodología

3.1. Planteamiento como problema de clasificación

El sistema recomendador se formula como un problema de clasificación supervisada multiclase. Cada instancia corresponde a un artículo representado por sus campos textuales (`title`, `abstract` y `keywords`) y la etiqueta (`journal_id`) identifica la revista de publicación entre el conjunto de candidatas (revistas 1, 2, 3 y 5). De este modo, dada la información textual de un nuevo manuscrito (título, resumen y palabras clave), el modelo predice la clase más probable, que se interpreta como la revista recomendada para su envío.

3.2. Metodología de proyecto (CRISP-DM)

El desarrollo del recomendador se ha guiado por la metodología CRISP-DM, siguiendo un ciclo iterativo desde la comprensión del problema hasta la evaluación de resultados y la preparación de entregables.

En la fase de *comprensión del negocio* se definió el objetivo principal del proyecto: recomendar la revista más adecuada a partir del título, el resumen y las palabras clave de un manuscrito, restringiendo la recomendación a las revistas 1, 2, 3 y 5.

En la fase de *comprensión de los datos* se recopiló y revisó el corpus de artículos (2020–2024) y se analizó su distribución por clase, en la que se detectó un desbalance entre revistas.

En la fase de *preparación de los datos* se unificaron los ficheros en bruto y se generó un dataset tabular único, aplicando limpieza mínima (filtrado de registros incompletos y deduplicación) y construyendo el campo `text` como entrada común a los modelos.

En la fase de *modelado* se implementaron dos aproximaciones: una clásica basada en representación TF-IDF y modelos de clasificación en Scikit-learn, y una conexionista basada en una red neuronal implementada en PyTorch.

En la fase de *evaluación* se compararon los modelos bajo el mismo protocolo experimental (partición estratificada y métricas por clase), analizando matrices de confusión y errores para interpretar los resultados.

Finalmente, en la fase de *despliegue* se empaquetaron los artefactos del proyecto (datasets procesados, modelos entrenados, notebooks y scripts) junto con la memoria y la presentación para su entrega y defensa.

3.3. Estructura del proyecto y reproducibilidad (TDSP)

Para facilitar la reproducibilidad y el mantenimiento, el proyecto se organizó siguiendo una estructura inspirada en TDSP, separando datos, código, modelos, notebooks y documentación.

La estructura principal del repositorio es la siguiente:

En particular, `data/raw` contiene los ficheros en bruto organizados por revista y año, mientras que `data/processed` almacena el dataset tabular final utilizado por los modelos. Los modelos entrenados y sus artefactos se guardan en `models/` (separando Scikit-learn y PyTorch), y los resultados intermedios (métricas, errores y figuras) se depositan en `docs/`. Los notebooks documentan el proceso exploratorio y de experimentación, y los scripts en `src/` permiten ejecutar el pipeline de forma no interactiva.

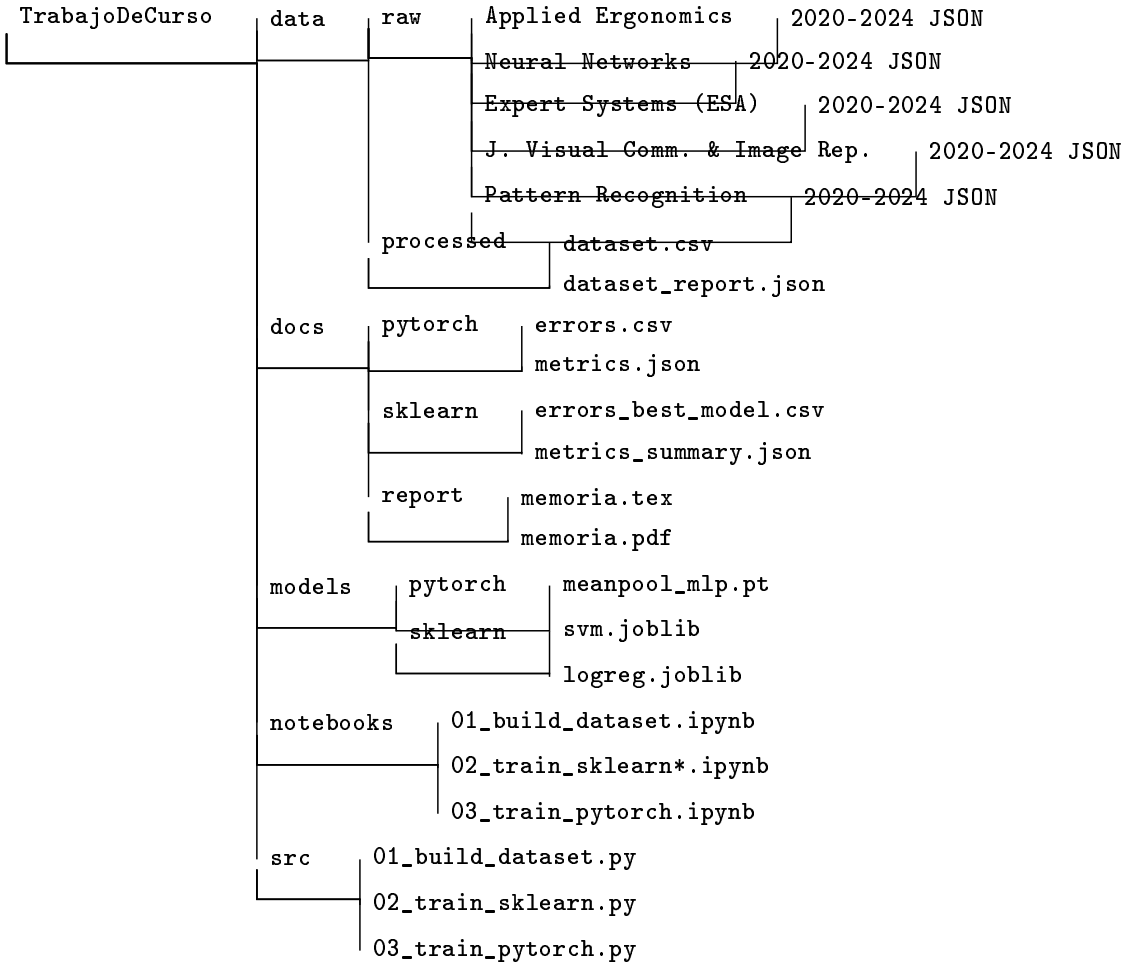

Figura 2: Estructura general del repositorio del proyecto

3.4. Protocolo de evaluación

Para evaluar el rendimiento de los modelos se utiliza una partición estratificada del conjunto de datos en entrenamiento y prueba, de forma que se preserva la proporción de artículos por revista en ambos subconjuntos. La comparación de modelos se realiza manteniendo fija la partición (misma semilla) para garantizar la reproducibilidad. Se reportan métricas globales (*accuracy*) y métricas por clase (precisión, *recall* y F1), junto con F1 macro y matrices de confusión para analizar los patrones de error entre revistas, especialmente relevantes dada la existencia de desbalance entre clases.

3.5. Métricas de evaluación

Para comparar de forma objetiva los enfoques clásico y conexionista, se emplean métricas estándar de clasificación multiclase que permiten evaluar tanto el rendimiento global como el rendimiento por clase. En particular, se reportan *accuracy*, *precision*, *recall* y *F1-score*, además del *support* (número de ejemplos) de cada clase.

3.5.1. Accuracy

La *accuracy* mide la proporción de ejemplos correctamente clasificados sobre el total. Aunque es una métrica fácil de interpretar, puede resultar poco informativa cuando el conjunto está desbalanceado, ya que un modelo puede obtener alta *accuracy* prediciendo bien solo las clases mayoritarias.

3.5.2. Precision, recall y F1-score

Para una clase c, la *precision* cuantifica qué fracción de las predicciones como c son correctas, mientras que el *recall* mide qué fracción de los ejemplos reales de c se recuperan correctamente. El *F1-score* es la media armónica entre *precision* y *recall*, y resume el compromiso entre ambos tipos de error (falsos positivos y falsos negativos). Estas métricas se reportan por clase, lo que permite detectar clases problemáticas incluso si la *accuracy* global es alta.

3.5.3. Promedios macro y ponderado

Además de las métricas por clase, se reporta **F1 macro** (promedio no ponderado del F1 de cada clase), que asigna el mismo peso a todas las clases y es especialmente útil para evaluar escenarios con desbalance. Como complemento, se reporta el promedio ponderado (*weighted avg*), que pondera cada clase por su *support* y suele reflejar mejor el rendimiento global cuando las clases tienen frecuencias muy diferentes. En este proyecto se prioriza F1 macro como criterio de selección del mejor modelo, y se usa el promedio ponderado como información adicional.

3.5.4. Matriz de confusión

La matriz de confusión cruza etiquetas reales y predicciones para cuantificar qué clases tienden a confundirse entre sí. Este análisis es clave para interpretar los errores del modelo y justificar decisiones de modelado (por ejemplo, necesidad de balanceo o de representaciones más expresivas).

3.6. Aproximación clásica (Scikit-learn)

La aproximación clásica se basa en transformar los documentos en una representación vectorial dispersa y entrenar clasificadores lineales tradicionales sobre dicha matriz. Para asegurar reproducibilidad y evitar fugas de información, se recomienda encapsular todo el flujo (vectorización → modelo) en un `Pipeline` de Scikit-learn, de forma que el `fit` del vectorizador se realice únicamente con el conjunto de entrenamiento en cada experimento. Además, se prioriza el uso de modelos eficientes para alta dimensionalidad y matrices dispersas (por ejemplo, `LinearSVC` o modelos lineales entrenados con SGD), dado que suelen ofrecer un buen compromiso entre rendimiento y coste computacional en clasificación de textos.

3.6.1. Representación del texto: TF-IDF

El texto de entrada se construye concatenando el título, resumen y palabras clave en un único campo (`text`), manteniendo un preprocesado consistente para todos los modelos. A partir de este campo se genera una matriz término-documento usando `TfidfVectorizer`, que produce directamente la matriz TF-IDF (equivalente a `CountVectorizer` + `TfidfTransformer`). En la práctica, se exploran parámetros actuales habituales en clasificación de texto: *n-grams* (por ejemplo, `ngram_range=(1,2)`), control de vocabulario con `min_df`/`max_df` para filtrar términos extremadamente raros o muy frecuentes, y *stopwords* para reducir ruido.

3.6.2. Modelos evaluados

Sobre la representación TF-IDF se entrenan clasificadores lineales robustos para texto, destacando `LinearSVC` (SVM lineal) por su buen rendimiento y compatibilidad con entradas dispersas. Adicionalmente, se mantiene la regresión logística multinomial y Naive Bayes como comparación, ya que permiten contrastar modelos discriminativos y generativos en el mismo espacio TF-IDF.

Para tratar el desbalance entre revistas, se emplea ponderación de clases (`class_weight="balanced"`) en los modelos que lo soportan, con el objetivo de penalizar más los errores en clases minoritarias.

Además de los modelos lineales, se incorpora un clasificador SVM con kernel RBF (`SVC(kernel=rbf)`) como variante no lineal sobre TF-IDF. Este modelo puede capturar fronteras de decisión más complejas, aunque su coste computacional es significativamente mayor que `LinearSVC`, especialmente en conjuntos de datos grandes; por ello se evalúa como referencia de capacidad, manteniendo el mismo protocolo y partición para una comparación justa.

3.6.3. Selección del mejor modelo

La selección del modelo final se realiza priorizando F1 macro, ya que es más informativa que *accuracy* bajo desbalance al promediar el rendimiento por clase de forma no ponderada. Como criterio complementario, se revisa el rendimiento por clase y la matriz de confusión para detectar confusiones sistemáticas (por ejemplo, entre revistas cercanas temáticamente) y justificar la elección final en términos de interpretabilidad. Finalmente, el modelo clásico seleccionado corresponde a `SVC` con kernel RBF y ponderación de clases (`class_weight="balanced"`), ya que obtiene el mayor F1 macro en test (0.7703) dentro de los enfoques evaluados. Esta elección se adopta reconociendo el mayor coste de entrenamiento del kernel RBF, por lo que se justifica su uso como mejor rendimiento clásico frente a alternativas más eficientes como `LinearSVC`.

3.7. Aproximación conexionista (PyTorch)

La aproximación conexionista se implementa mediante una red neuronal en PyTorch para resolver la misma tarea de clasificación multiclase, aprendiendo representaciones densas (*embeddings*) a partir de secuencias de tokens. Para asegurar una comparación justa con la aproximación clásica, se mantiene la misma partición *train/test* y se reportan las mismas métricas (accuracy, F1 macro y análisis por clase). Como actualización metodológica, se enfatiza el control de reproducibilidad (semillas) y el uso de *early stopping* para limitar sobreajuste, tal como se aplica en el cuaderno PyTorch.

3.7.1. Tokenización y vocabulario

El texto `text` se tokeniza y se construye un vocabulario usando únicamente el conjunto de entrenamiento, mapeando cada token a un índice entero. Se reservan identificadores especiales para `<pad>` (relleno) y `<unk>` (tokens desconocidos), y se limita el tamaño del vocabulario con umbrales tipo `min_freq` y `max_vocab`. Para entrenar por lotes, las secuencias se truncan a una longitud máxima (por ejemplo, 256) y se rellenan con padding en una *collate function* que también calcula las longitudes reales para enmascarar correctamente el padding durante el *pooling*.

Como mejora habitual, conviene fijar explícitamente `padding_idx` en `nn.Embedding` para que el embedding del padding no se actualice (gradiente cero) y no introduzca ruido en la representación promedio.

3.7.2. Arquitectura del modelo

Se emplea una arquitectura eficiente tipo *embedding* + *mean pooling* + MLP (figura 3), adecuada como estado inicial neuronal para textos: `nn.Embedding` transforma IDs en vectores densos, se promedian los embeddings (ignorando `<pad>` con una máscara), y una MLP genera los *logits* multiclase. En la implementación, el *mean pooling* se realiza sumando embeddings y dividiendo por la longitud efectiva, lo que permite manejar batches con longitudes variables. La predicción final se obtiene seleccionando la clase con mayor logit (equivalente a arg máx tras *softmax* a efectos de clasificación), manteniendo cuatro clases (revistas 1, 2, 3 y 5 tras el mapeo).

3.7.3. Entrenamiento

El entrenamiento minimiza `CrossEntropyLoss` y optimiza con Adam (por ejemplo, `lr` $= 10^{-3}$), monitorizando *loss* y *accuracy* en train y test por época. Para regularizar, se aplica *dropout* en la MLP y se usa *early stopping* con una *patience* (por ejemplo, 10 épocas sin mejora) para detener el entrenamiento cuando no mejora el rendimiento en test. Para abordar desbalance, se prueba el uso de pesos de clase en `CrossEntropyLoss(weight=...)` calculados inversamente a la frecuencia por clase, comparando resultados con y sin balanceo.

En los resultados del cuaderno, el modelo sin balanceo logra la mejor *accuracy* (0.7705) y un F1 macro de 0.7438, mientras que el balanceo mantiene F1 macro prácticamente igual pero reduce *accuracy* (0.7570), por lo que se selecciona el modelo sin balanceo como final dentro de PyTorch.

Finalmente, se reportan *precision/recall/F1* por clase y se analiza la matriz de confusión, donde destacan confusiones entre revistas 2, 3 y 5, coherentes con solapamiento temático en el corpus.

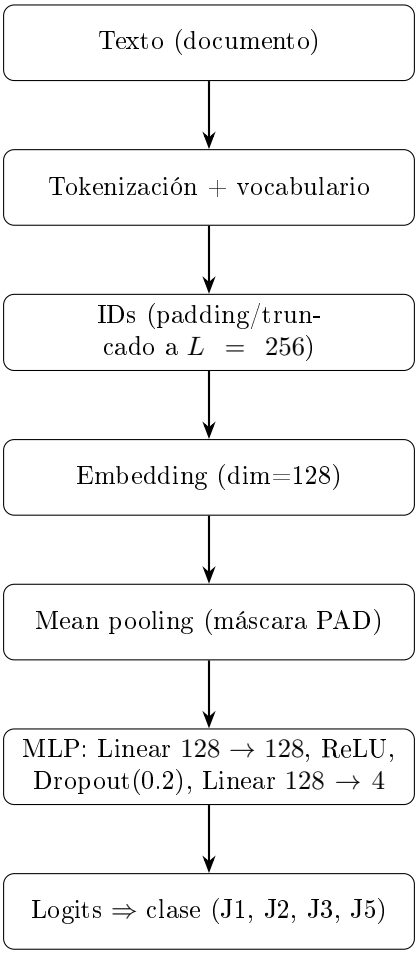

Figura 3: Arquitectura del enfoque conexionista.

4. Diseño experimental

4.1. Particiones de datos

El conjunto de datos se divide en subconjuntos de entrenamiento y prueba mediante una partición estratificada, preservando la proporción de artículos por revista en ambos subconjuntos. Se trata de un diseño que busca garantizar que todas las clases estén representadas durante el entrenamiento y que la evaluación sea comparable entre modelos, especialmente en presencia de desbalance entre clases. Para asegurar reproducibilidad, se fija una semilla (`random_state`) en el proceso de particionado y se mantiene la misma partición en todos los experimentos, tanto en la aproximación clásica como la conexionista.

4.2. Hiperparámetros

En la aproximación clásica, se emplea una representación TF-IDF (*Term Frequency-Inverse Document Frequency*) y se analizan parámetros habituales en clasificación de texto. En particular, se consideran la normalización a minúsculas, eliminación de *stopwords*, uso de unigramas y bigramas (`ngram_range=(1,2)`), y filtrado de términos raros o demasiado frecuentes mediante `min_df` y `max_df`. Para los clasificadores se prueban modelos lineales (regresión logística y SVM lineal) y, como referencia, Naive Bayes multinomial; además, en modelos que lo permiten se evalúa el uso de ponderación de clases (`class_weight="balanced"`) como mecanismo para mitigar el desbalance.

En la aproximación conexionista, los hiperparámetros principales incluyen la longitud máxima de secuencia, el tamaño del vocabulario, la dimensión del *embedding*, el tamaño de la capa oculta del MLP, el tamaño de lote (*batch size*), la tasa de aprendizaje y el número de épocas de entrenamiento. El entrenamiento se realiza optimizando la pérdida de entropía cruzada con un optimizador basado en gradiente, y se monitoriza el rendimiento en el conjunto de prueba con las mismas métricas que la aproximación clásica para permitir una comparación directa.

5. Resultados

5.1. Resultados de la aproximación clásica (SKLearn)

El corpus procesado contiene 17 468 artículos distribuidos en cuatro clases (revistas 1, 2, 3 y 5). El conjunto se dividió en entrenamiento y prueba mediante partición estratificada, obteniéndose 13 974 instancias para entrenamiento y 3 494 para prueba.

Se evaluaron tres modelos sobre la misma representación TF-IDF (regresión logística, SVM lineal y Naive Bayes multinomial). El cuadro 2 resume los resultados obtenidos *sin* ponderación de clases.

Cuadro 2: Resultados en test de la aproximación clásica (TF-IDF + clasificador) sin ponderación de clases.

Modelo	**Accuracy**	**F1 macro**
Logistic Regression	0.77	0.74
Linear SVM	0.78	0.77
MultinomialNB	0.58	0.28

El mejor rendimiento sin ponderación de clases se obtuvo con SVM lineal (F1 macro = 0.767), por lo que se tomó como punto de referencia de la aproximación clásica. Naive Bayes multinomial mostró un rendimiento claramente inferior, lo que sugiere que, para este corpus, la frontera lineal aprendida por SVM sobre TF-IDF captura mejor las diferencias entre revistas que el supuesto generativo de independencia condicional de NB.

Adicionalmente, se evaluaron dos estrategias para mitigar el desbalance del corpus: (i) la ponderación de clases mediante `class_weight="balanced"` y (ii) el sobremuestreo sintético de las clases minoritarias mediante SMOTE, aplicado únicamente durante el entrenamiento para no introducir fuga de información. En la figura 4 se comparan los modelos en distintos escenarios (sin balanceo, con `class_weight`, con SMOTE y el enfoque híbrido), observándose mejoras moderadas en F1 macro respecto al caso sin balanceo. En conjunto, el mejor rendimiento dentro de los modelos lineales evaluados se obtiene con `linear_svm_balanced`, con una mejora ligera frente a su versión sin balanceo, mientras que SMOTE logra resultados comparables según el clasificador.

La figura 8 desglosa el efecto del balanceo por clase y muestra que el mayor beneficio se concentra en la revista 2, que es la clase con peor separabilidad, a costa de una degradación pequeña en la revista 1. Este comportamiento es coherente con el objetivo de estas técnicas, que buscan reducir el sesgo hacia la clase mayoritaria y mejorar el rendimiento en clases minoritarias o más difíciles.

Finalmente, la figura 5, junto con el cuadro 3, presentan las métricas por clase (precisión, recall y F1) y permiten interpretar el impacto del balanceo más allá de las métricas agregadas. Se observa que la revista 1 mantiene un rendimiento muy alto (F1 cercano a 0.96) en la mayoría de configuraciones, mientras que la revista 2 continúa siendo la clase más difícil, con errores concentrados en confusiones con las revistas 3 y 5, lo que justifica priorizar F1 macro como criterio principal de selección.

Cuadro 3: Resultados en test de la aproximación clásica (TF-IDF) bajo distintas estrategias de desbalance.

Modelo	**Estrategia**	**Accuracy**	**F1 macro**
Logistic Regression	Sin balanceo	0.77	0.7432
Linear SVM	Sin balanceo	0.78	0.7671
MultinomialNB	Sin balanceo	0.58	0.2785
Logistic Regression	class_weight	0.76	0.7609
Linear SVM	class_weight	0.78	0.7690
MultinomialNB	class_weight	0.58	0.2785
Logistic Regression	SMOTE	0.77	0.7685
Linear SVM	SMOTE	0.78	0.7678
MultinomialNB	SMOTE	0.74	0.7420
Logistic Regression	SMOTE + class_weight	0.77	0.7685
Linear SVM	SMOTE + class_weight	0.78	0.7678
MultinomialNB	SMOTE + class_weight	0.74	0.7420

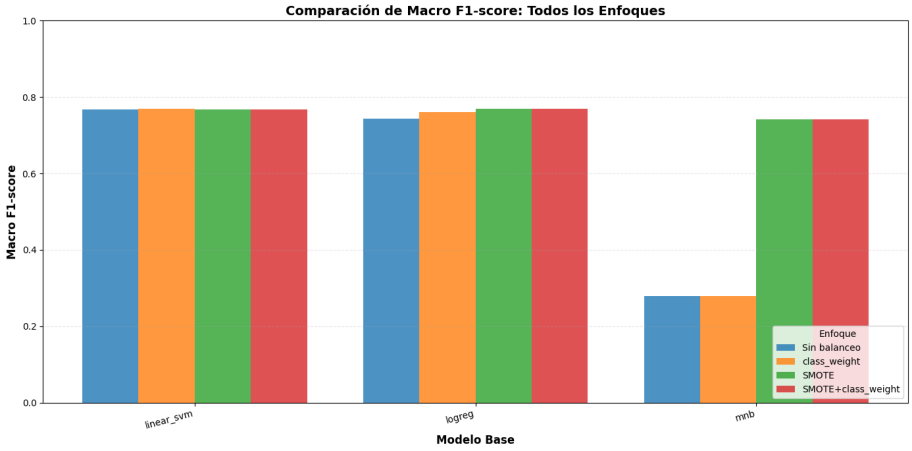

Figura 4: Comparación de modelos mediante F1 macro en test, incluyendo variantes con ponderación de clases (`class_weight="balanced"`) y SMOTE.

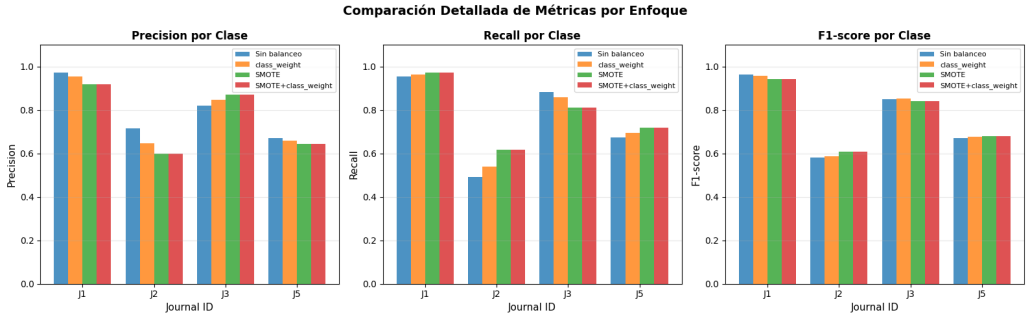

Figura 5: Métricas por clase (precisión, recall y F1) del mejor modelo clásico en test.

Para analizar en detalle los errores se emplean las matrices de confusión del modelo seleccionado sin balanceo (figura 6) y con balanceo (figura 7). En ambas se observa que las principales confusiones se producen entre las revistas 2, 3 y 5: una fracción importante de artículos de la revista 2 se clasifica como 3 o 5, y también se observa confusión frecuente entre 5 y 3. Por el contrario, la revista 1 se clasifica de forma muy fiable, lo que sugiere que su vocabulario es más distintivo respecto al resto.

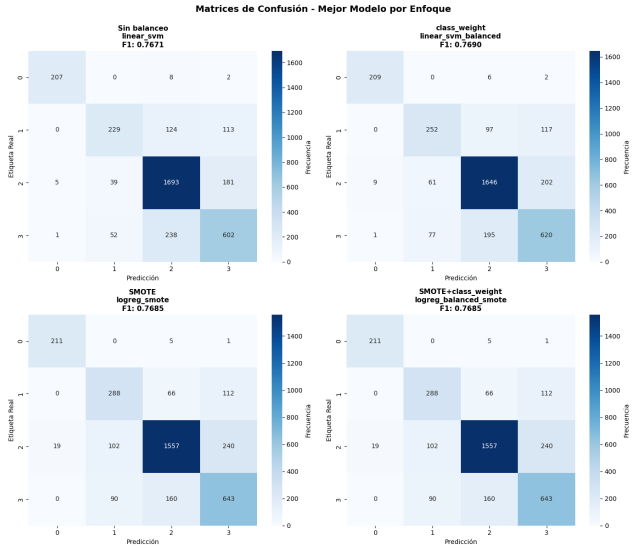

Figura 6: Matriz de confusión del mejor modelo clásico sin balanceo (TF-IDF + LinearSVC) en el conjunto de prueba.

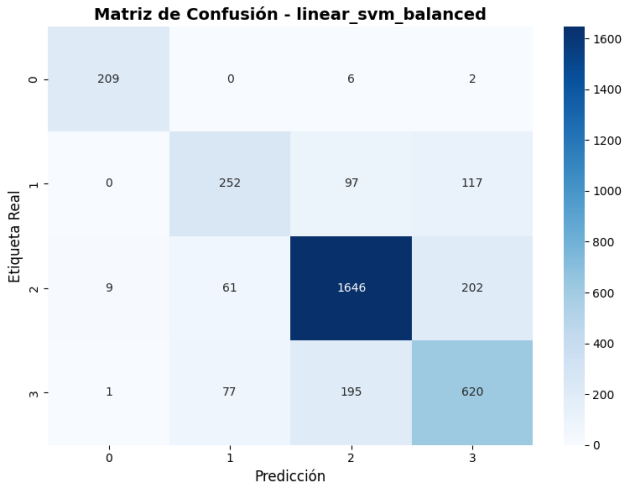

Figura 7: Matriz de confusión del mejor modelo clásico con balanceo (TF-IDF + LinearSVC, `class_weight="balanced"`) en el conjunto de prueba.

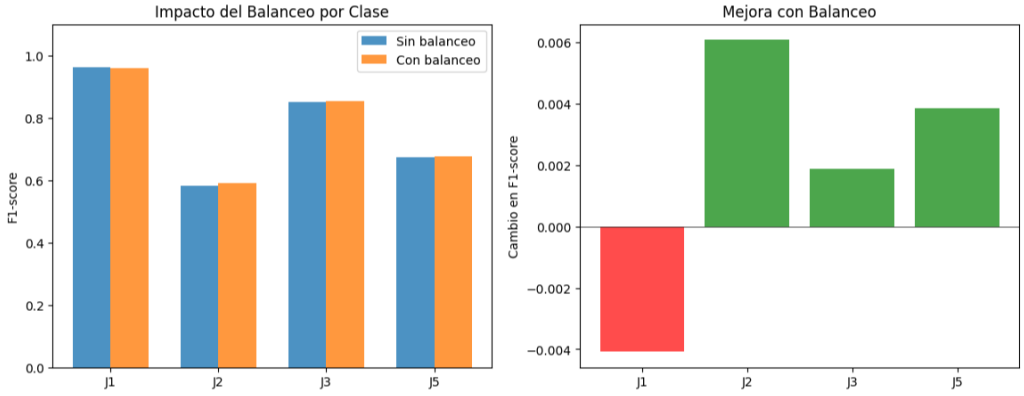

Figura 8: Impacto del balanceo por clase: comparación de F1 por clase (sin vs con balanceo) y mejora absoluta por clase.

Como experimento adicional sobre la representación TF-IDF, se evaluó un SVM no lineal mediante `SVC` con kernel RBF. Dado su mayor coste computacional, se advierte que el entrenamiento puede prolongarse varios minutos u horas en función del hardware y del tamaño del conjunto de entrenamiento. En los ensayos realizados, la variante sin ponderación (`svc_rbf`) obtuvo un F1 macro de 0.7427, mientras que al incorporar ponderación de clases (`svc_rbf_balanced`,

`class_weight="balanced"`) se alcanzó el mejor rendimiento global dentro de los enfoques clásicos, con F1 macro de 0.7703 y *accuracy* de 0.77. Por clase, el modelo mantiene un rendimiento muy alto en la revista 1 (F1 = 0.95) y confirma que la revista 2 sigue siendo la más difícil (F1 = 0.61), con confusiones principalmente hacia las revistas 3 y 5; en consecuencia, `svc_rbf_balanced` se considera el mejor modelo clásico según F1 macro, aunque alternativas lineales como `linear_svm_balanced` quedan muy próximas en rendimiento con un coste de entrenamiento sustancialmente menor.

5.2. Resultados de la aproximación conexionista (PyTorch)

La aproximación conexionista se implementó en PyTorch mediante un modelo de clasificación de texto basado en *embeddings* aprendidos, agregación por promedio (*mean pooling*) y un MLP. El entrenamiento se realizó en GPU, con secuencias truncadas a longitud máxima 256 y un vocabulario de tamaño 29 174 tokens (incluyendo `<pad>` y `<unk>`).

Modelo base (sin balanceo). El modelo sin ponderación de clases alcanzó su mejor rendimiento en la época 4, con *accuracy* en test de 0.7705 y F1 macro de 0.7438. Las Figuras 9 y 10 muestran la evolución del entrenamiento, mientras que la Figura 11 resume precisión, recall y F1 por revista. Para analizar los errores, se utiliza la matriz de confusión (Figura 12) y una visualización adicional centrada en errores (Figura 13), donde se observa que las confusiones más frecuentes se concentran entre las revistas 2, 3 y 5.

Modelo con balanceo de clases. Dado el desbalance del corpus, se evaluó una variante equivalente a `class_weight="balanced"` de Scikit-learn, introduciendo pesos de clase en la función de pérdida (`CrossEntropyLoss`). Los pesos se calcularon inversamente proporcionales a la frecuencia de cada clase en entrenamiento, obteniéndose: 4.03 (revista 1), 1.87 (revista 2), 0.46 (revista 3) y 0.98 (revista 5). El modelo balanceado alcanzó un *accuracy* máximo de 0.7570 y un F1 macro de 0.7435, mostrando una mejora en la revista 2 (+0.024 en F1) pero una ligera degradación en el resto de revistas. La Figura 14 resume este impacto por clase.

Selección del modelo final. En este caso, el balanceo no aporta una mejora global en F1 macro y reduce el *accuracy* respecto al modelo base. Por tanto, el modelo seleccionado para guardar y reportar como resultado final de la aproximación conexionista es el modelo *sin* balanceo, manteniendo el balanceo como experimento complementario orientado a mejorar el rendimiento de la clase minoritaria (revista 2).

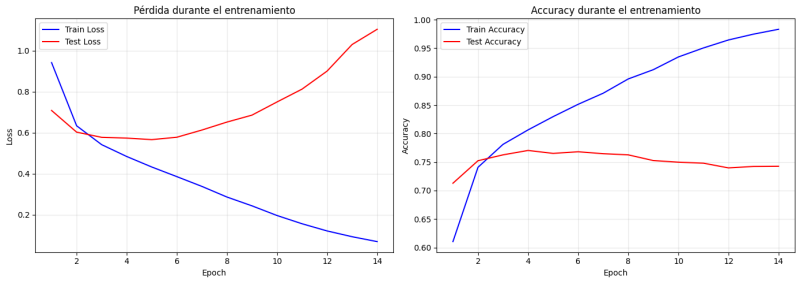

Figura 9: Evolución de pérdida y *accuracy* durante el entrenamiento del modelo PyTorch (sin balanceo).

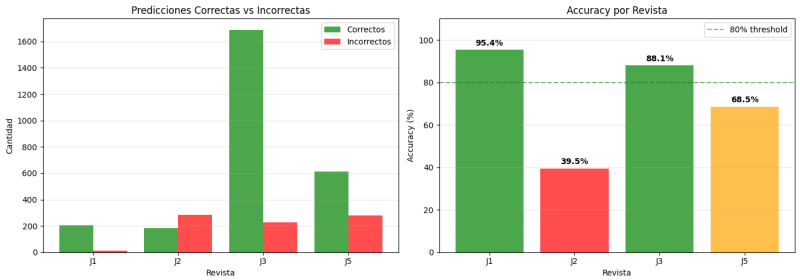

Figura 10: Evolución del *accuracy* y selección del mejor punto de entrenamiento (*early stopping*) del modelo PyTorch (sin balanceo).

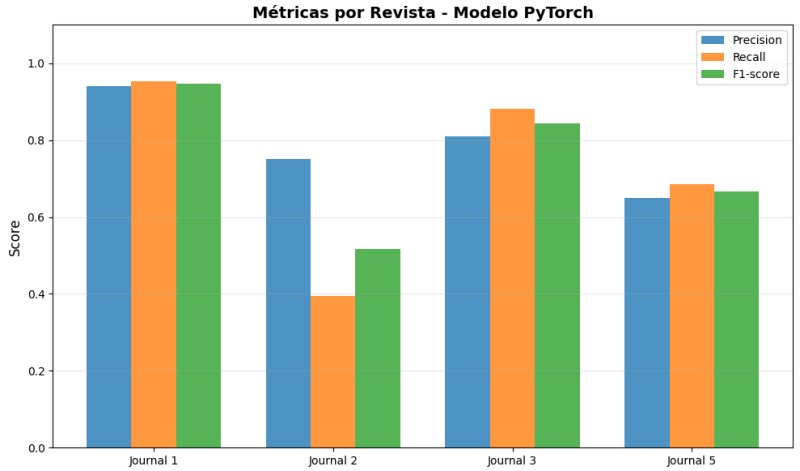

Figura 11: Métricas por clase (precisión, recall y F1) del modelo PyTorch seleccionado (sin balanceo) en el conjunto de prueba.

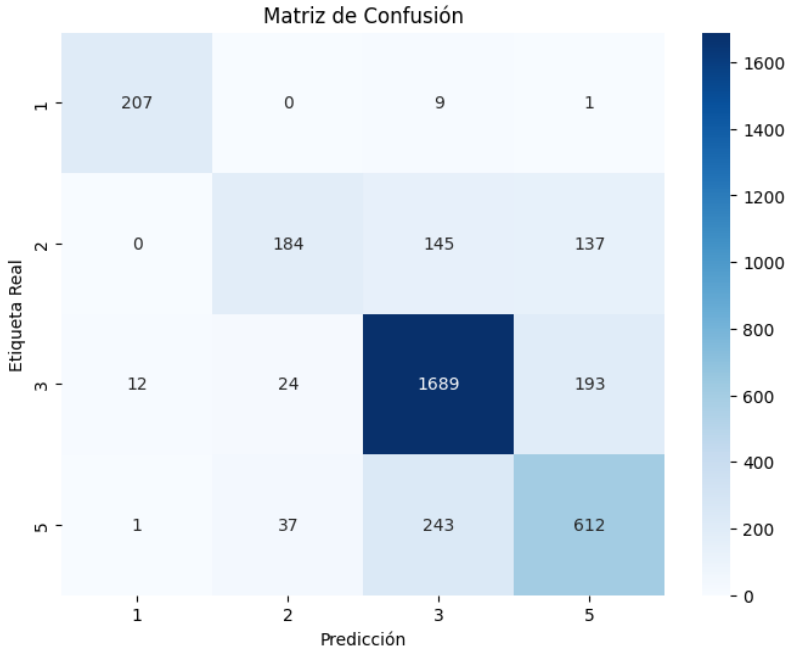

Figura 12: Matriz de confusión del modelo PyTorch seleccionado (sin balanceo) en el conjunto de prueba.

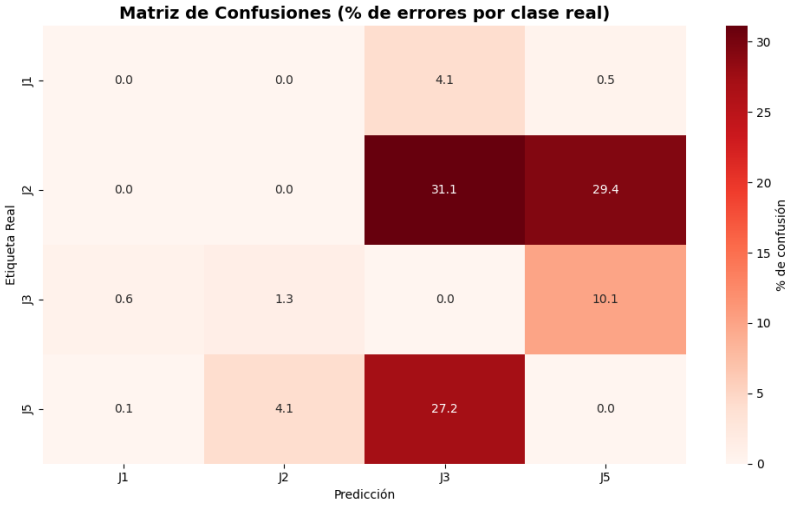

Figura 13: Visualización de errores a partir de la matriz de confusión del modelo PyTorch (sin balanceo), destacando las confusiones más frecuentes entre revistas.

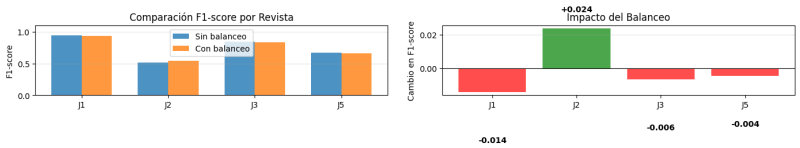

Figura 14: Impacto del balanceo en PyTorch: comparación de F1 por clase entre el modelo sin balanceo y el modelo con pesos de clase, y cambio absoluto por clase.

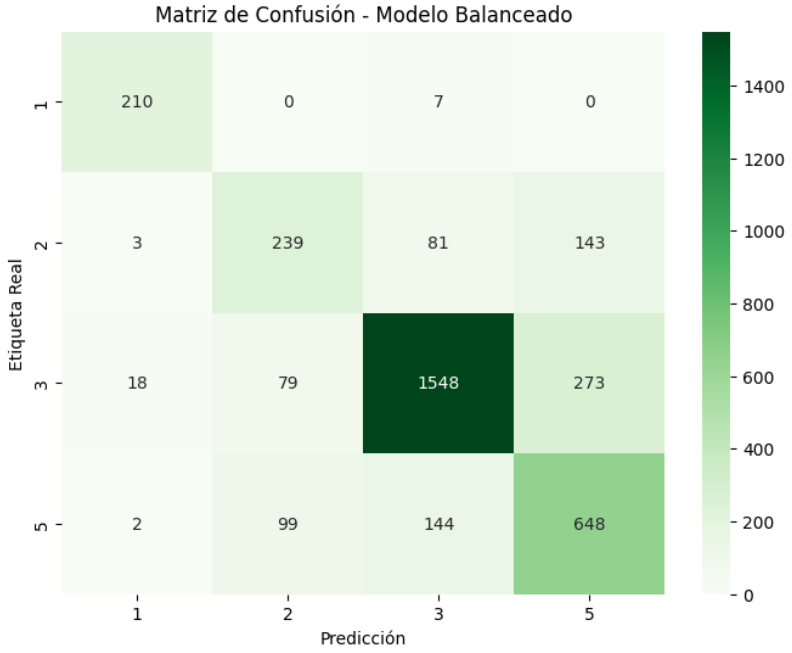

Figura 15: Matriz de confusión del modelo PyTorch con balanceo (pérdida ponderada por clase) en el conjunto de prueba.

Cuadro 4: Resultados en test de la aproximación conexionista (PyTorch) con y sin balanceo de clases.

Modelo	**Accuracy**	**F1 macro**
PyTorch (sin balanceo)	0.7705	0.7438
PyTorch (con balanceo en la pérdida)	0.7570	0.7435

5.3. Comparación entre enfoques

Los dos enfoques obtienen una *accuracy* muy similar en test: el mejor modelo clásico (`TF-IDF + LinearSVC` con `class_weight="balanced"`) alcanza 0.78, mientras que el mejor modelo conexionista base (PyTorch, sin balanceo) alcanza 0.7705. Sin embargo, al comparar la métrica más robusta ante desbalance (F1 macro), el enfoque clásico resulta superior: F1 macro = 0.7690 (clásico) frente a 0.7438 (PyTorch sin balanceo).

En términos prácticos, el enfoque clásico (TF-IDF + SVM lineal) ofrece un punto de partida muy estable con un coste de entrenamiento bajo y fácil de reproducir, mientras que la red neuronal con *embeddings* y *mean pooling* no supera al clásico con la arquitectura simple utilizada.

Respecto al tratamiento del desbalance, en el enfoque clásico la ponderación de clases aporta una mejora ligera en F1 macro (por ejemplo, `linear_svm` 0.7671 → `linear_svm_balanced` 0.7690). En PyTorch, la ponderación de clases en `CrossEntropyLoss` mantiene el F1 macro prácticamente igual (0.7438 → 0.7435) y reduce la *accuracy* (0.7705 → 0.7570), aunque mejora la clase 2 (+0.024 F1) a costa de degradar ligeramente otras clases, por lo que se selecciona el modelo sin balanceo como final para este enfoque.

5.4. Análisis de errores

En ambos enfoques, las confusiones más relevantes se concentran entre las revistas 2, 3 y 5, lo que sugiere una solapación temática importante en el espacio de características (tanto en TF-IDF como en las representaciones aprendidas). En el modelo PyTorch sin balanceo, las confusiones más frecuentes en test son: revista 5 → 3 (243 errores, 27.2 % de J5), revista 3 → 5 (193 errores, 10.1 % de J3), revista 2 → 3 (145 errores, 31.1 % de J2) y revista 2 → 5 (137 errores, 29.4 % de J2). Este patrón indica que la revista 2 es la clase más difícil de separar, ya que una parte significativa de sus artículos se asigna a las revistas 3 y 5, y que existe también una confusión bidireccional notable entre 3 y 5.

El balanceo en PyTorch reduce parcialmente el sesgo hacia la clase mayoritaria en términos cualitativos (mejora de F1 en la clase 2), pero no resuelve la confusión estructural entre las revistas 2, 3 y 5, por lo que la mejora global es marginal.

Como línea de mejora, estos errores son compatibles con un escenario en el que la señal discriminativa depende de semántica más contextual que de palabras clave aisladas, por lo que podrían explorarse arquitecturas más expresivas (por ejemplo, pooling con atención o modelos preentrenados tipo Transformer) o enriquecimiento del texto (secciones adicionales del artículo) para aumentar separabilidad entre revistas cercanas.

6. Conclusiones

En este trabajo, el modelo conexionista basado en embeddings, mean pooling y una MLP ofrece un rendimiento global sólido, alcanzando un mejor accuracy de test de 0.7705 y un macro F1 cercano a 0.744, lo que sugiere que captura señales útiles del texto pese al desequilibrio entre revistas.

La estrategia de balanceo mediante pesos de clase no aporta una mejora global en macro F1, ya que prácticamente se mantiene, y, además, reduce ligeramente la exactitud general (de 0.7705 a 0.7570), por lo que no se justifica como elección final si el objetivo principal es maximizar el rendimiento agregado. No obstante, el análisis por clase muestra que el balanceo puede modificar el reparto de errores y la sensibilidad a clases minoritarias, por lo que su uso debería depender de si se prioriza el equilibrio entre clases frente a la accuracy global.

Finalmente, la matriz de confusión evidencia confusiones sistemáticas entre algunas revistas, especialmente entre Journal 5 y Journal 3, y también entre Journal 2 con Journal 3 y Journal 5, lo que indica solapamiento semántico.

Como líneas de mejora, conviene explorar tokenización y normalización más robustas, arquitecturas con mejor modelado, por ejemplo, modelos basados en transformadores, y estrategias específicas para reducir las confusiones dominantes.

Referencias

[1] P Thomas Carroll. American science transformed. *American Scientist*, 74(5):466–485, 1986.

[2] Derek John de Solla Price. Little science, big science. 1963.

[3] Bernadette Freedman. Growth and change in the world's biological literature as reflected in biosis publications. *Publishing Research Quarterly*, 11(3):61–79, 1995.

[4] Eugene Garfield. Economics and realpolitik of exponential information growth or journal selection aint easy, 1973.

[5] Ronald N Kostoff. Overcoming specialization. *BioScience*, 52(10):937–941, 2002.

[6] Christine Lamb. Open access publishing models: opportunity or threat to scholarly and academic publishers? *Learned publishing*, 17(2):143–150, 2004.

[7] Peter A Lawrence. The politics of publication. *Nature*, 422(6929):259–261, 2003.

[8] Harvey C Lehman. The exponential increase of man's cultural output. *Soc. F.*, 25:281, 1946.

[9] I Sengupta. The growth of biophysical literature. *Scientometrics*, 8(5-6):365–375, 1985.

[10] Paul Weiss. Knowledge: a growth process: Knowledge grows like organisms, with data serving as food to be assimilated, rather than merely stored. *Science*, 131(3415):1716–1719, 1960.

Glosario

Clasificación de documentos Tarea de aprendizaje supervisado que asigna a cada texto una etiqueta discreta (por ejemplo, `journalid`) a partir de sus características lingüísticas.

Enfoque clásico Estrategia basada en representaciones tipo bolsa de palabras (por ejemplo, TF-IDF) combinadas con clasificadores lineales como SVM/LinearSVC.

Aproximación conexionista Enfoque basado en redes neuronales que aprenden representaciones (embeddings) y una función de decisión a partir de datos, en este caso con PyTorch.

TF-IDF Representación numérica de texto que pondera términos por su frecuencia en el documento (TF) y su rareza en el corpus (IDF), típica en enfoques clásicos.

LinearSVC Clasificador lineal (SVM lineal) usado para clasificación multiclase; en el proyecto se usa como baseline clásico (incluyendo variantes con balanceo).

Desbalance de clases Situación en la que algunas clases tienen muchos más ejemplos que otras, lo que puede sesgar el entrenamiento y las métricas.

`class_weight` Mecanismo (en modelos de `sklearn`) para ponderar clases, penalizando más los errores en clases minoritarias.

Pesos de clase (*class weights*) Ponderaciones aplicadas a la función de pérdida (por ejemplo, en `CrossEntropyLoss`) para compensar el desbalance, calculadas inversamente a la frecuencia de cada clase.

Train/Test split estratificado División del dataset en entrenamiento y test manteniendo proporciones de clases similares en ambos conjuntos.

Tokenización Proceso de convertir texto en una secuencia de unidades (tokens); en el cuaderno PyTorch se aplica una tokenización simple basada en regex y minúsculas.

Vocabulario (*vocab*) Conjunto de tokens conocidos por el modelo, construido a partir del *train*; se suelen incluir tokens especiales como `PAD` y `UNK`.

`PAD` / **Padding** Token (y operación) para rellenar secuencias a una longitud común dentro de un batch, permitiendo computación vectorizada.

`UNK` Token usado para representar palabras fuera del vocabulario.

Embedding Capa que transforma IDs de tokens en vectores densos de dimensión fija, aprendidos durante el entrenamiento.

Mean pooling Estrategia de agregación que promedia los embeddings de una secuencia (ignorando padding mediante una máscara) para obtener una representación global del texto.

MLP (*Multi-Layer Perceptron*) Red feed-forward (capas lineales y activaciones) usada como clasificador final sobre la representación agregada del texto.

Logits Salida real-valuada del modelo antes de aplicar `softmax`; se usa como entrada de `CrossEntropyLoss`.

`CrossEntropyLoss` Función de pérdida estándar para clasificación multiclase en PyTorch; admite pesos de clase mediante el parámetro `weight`.

Adam Optimizador usado para actualizar parámetros del modelo neuronal a partir del gradiente de la pérdida.

Epoch Una pasada completa del entrenamiento sobre el conjunto de entrenamiento.

Early stopping Criterio para detener el entrenamiento si la métrica en validación/test no mejora durante un número de epochs (*patience*) para evitar sobreajuste.

Accuracy Proporción de predicciones correctas sobre el total de ejemplos evaluados.

Precision Para una clase, proporción de predicciones de esa clase que son correctas.

Recall Para una clase, proporción de ejemplos reales de esa clase que el modelo recupera correctamente.

F1-score Media armónica entre *precision* y *recall*; resume el equilibrio entre ambos.

Macro F1 Promedio no ponderado del F1 por clase (todas las clases pesan igual), útil cuando hay desbalance.

Weighted F1 Promedio del F1 por clase ponderado por el *support* (tamaño de clase).

Support Número de ejemplos reales de una clase en el conjunto evaluado.

Matriz de confusión Tabla que cruza etiqueta real y etiqueta predicha para analizar qué clases se confunden.

Batch / DataLoader Un *batch* es un subconjunto de ejemplos procesado en una iteración; `DataLoader` gestiona batches, shuffling y *collate*.

Collate function Función que construye un batch a partir de ejemplos individuales (incluye padding y cálculo de longitudes).

`class2idx` / `idx2class` Mapas para convertir etiquetas originales (por ejemplo, 1,2,3,5) a índices contiguos (0..N-1) y viceversa, necesarios para entrenar/evaluar en PyTorch.

Reproducibilidad (seed) Fijación de semillas aleatorias en `random`, `numpy` y `torch` para obtener resultados repetibles.