Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Clasificación Inferida Mediante Datos: Año de Publicación de un TFG

Autor: Celia Hirt Quiroga
Tutor(a): Juan Antonio Fernández del Pozo

Madrid, Junio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Clasificación Inferida Mediante Datos: Año de Publicación de un TFG

Junio 2025

Autor: Celia Hirt Quiroga
Tutor: Juan Antonio Fernández del Pozo
Departamento de Inteligencia Artificial
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

Este Trabajo de Fin de Grado tiene como objetivo desarrollar un sistema capaz de inferir el año de publicación de un Trabajo de Fin de Grado (TFG) a partir únicamente de su resumen textual. La propuesta se enmarca dentro del ámbito del procesamiento del lenguaje natural (PLN) y la clasificación supervisada, y busca explorar la viabilidad de utilizar información textual limitada para extraer patrones temporales útiles en entornos académicos.

La base de datos utilizada se construyó de forma automática mediante técnicas de *web scraping* aplicadas al Archivo Digital de la Universidad Politécnica de Madrid (AD-UPM), seleccionando TFGs pertenecientes a cuatro titulaciones distintas y distribuidos equitativamente entre los años 2016 y 2024. Cada TFG fue representado mediante su resumen, y se aplicó un preprocesamiento exhaustivo que incluyó conversión a minúsculas, eliminación de caracteres no alfabéticos, tokenización, eliminación de *stopwords*, lematización y *stemming*.

En cuanto a la representación vectorial, se emplearon dos enfoques: TF-IDF y Word2Vec. Los resúmenes fueron posteriormente clasificados utilizando cuatro modelos supervisados: K-Nearest Neighbors, Naive Bayes (Multinomial y Gaussiano), Random Forest y XGBoost. Cada modelo fue evaluado en combinación con las dos técnicas de representación y ambas variantes de preprocesamiento léxico.

Los experimentos se realizaron con dos esquemas de etiquetas: uno de nueve clases (una por cada año entre 2016 y 2024) y otro reducido a tres clases agrupadas: 2016–2018, 2019–2021 y 2022–2024. Los resultados mostraron que la clasificación con tres etiquetas produjo métricas significativamente superiores. La mejor configuración fue la combinación de Random Forest, TF-IDF y *stemming*, con un F1 macro de 0.4321. En el caso de las nueve etiquetas, el mejor rendimiento lo ofreció XGBoost con TF-IDF y lematización, alcanzando un F1 macro de 0.1527 y un error medio en la predicción de 2.67 años.

Además del análisis de resultados, el trabajo reflexiona sobre la influencia del preprocesamiento, la representación semántica y la granularidad de la clasificación en el rendimiento de los modelos. También se aborda el impacto potencial del sistema en distintos contextos —personal, empresarial, social, económico, medioambiental y cultural—, y se identifican varios Objetivos de Desarrollo Sostenible (ODS) con los que el proyecto se alinea, como el ODS 4 (Educación de calidad) y el ODS 9 (Industria, innovación e infraestructura).

Finalmente, se proponen futuras líneas de trabajo, como el uso de técnicas no supervisadas como el *clustering*, la ampliación del corpus y la inclusión de más campos textuales. El trabajo concluye demostrando que es posible inferir información temporal relevante a partir de textos breves como los resúmenes de TFGs, utilizando herramientas accesibles y eficientes.

Abstract

This Final Degree Project aims to develop a system capable of inferring the year of publication of a Final Degree Project (TFG) based solely on its textual abstract. The proposal falls within the scope of natural language processing (NLP) and supervised classification, and seeks to explore the feasibility of using limited textual information to extract useful temporal patterns in academic contexts.

The dataset was automatically built using *web scraping* techniques applied to the Digital Archive of the Universidad Politécnica de Madrid (AD-UPM), selecting TFGs from four different degree programs, evenly distributed between the years 2016 and 2024. Each TFG was represented by its abstract, and underwent thorough preprocessing that included lowercasing, removal of non-alphabetic characters, tokenization, stopword removal, lemmatization, and stemming.

Two vector representation techniques were applied: TF-IDF and Word2Vec. The abstracts were then classified using four supervised models: K-Nearest Neighbors, Naive Bayes (Multinomial and Gaussian), Random Forest, and XGBoost. Each model was evaluated in combination with the two vectorization strategies and both lexical preprocessing variants.

Experiments were conducted with two labeling schemes: one with nine classes (one for each year from 2016 to 2024), and a simplified one with three grouped classes: 2016–2018, 2019–2021, and 2022–2024. The results showed that the three-class classification yielded significantly better performance. The best configuration combined Random Forest, TF-IDF, and stemming, achieving a macro F1 score of 0.4321. In the nine-class scenario, the best result came from XGBoost with TF-IDF and lemmatization, reaching a macro F1 score of 0.1527 and an average prediction error of 2.67 years.

In addition to the results analysis, the work discusses the impact of preprocessing, semantic representation, and label granularity on model performance. It also addresses the potential impact of the system in various contexts—personal, business, social, economic, environmental, and cultural—and identifies several Sustainable Development Goals (SDGs) aligned with the project, such as SDG 4 (Quality Education) and SDG 9 (Industry, Innovation and Infrastructure).

Finally, future lines of work are proposed, including the use of unsupervised techniques such as clustering, expanding the corpus, and incorporating additional textual fields. The work concludes by demonstrating that it is indeed possible to infer relevant temporal information from short texts such as TFG

abstracts using accessible and efficient tools.

Tabla de contenidos

Capítulo 1

Introducción

La minería de textos es una disciplina dentro del ámbito del análisis de datos y la inteligencia artificial que tiene como objetivo la extracción de conocimiento útil a partir de grandes volúmenes de texto no estructurado. A diferencia de los datos numéricos o categóricos, los textos presentan una estructura compleja, ambigua y contextual, lo que hace necesario aplicar técnicas específicas para su procesamiento, comprensión y análisis [1]. Esta área ha adquirido una gran relevancia en los últimos años gracias al aumento exponencial de la información textual disponible en internet, medios de comunicación, bases de datos científicas y entornos organizativos.

Una de las tareas más comunes dentro de la minería de textos es la *clasificación automática de documentos*, que consiste en asignar a cada documento una o varias etiquetas o categorías predefinidas. Esta técnica se utiliza ampliamente en diversos contextos: desde la detección de spam en correos electrónicos hasta la categorización temática de noticias, la clasificación de reseñas de productos según su polaridad, o la organización de archivos legales y administrativos [2]. Gracias a los avances en procesamiento de lenguaje natural (PLN) y aprendizaje automático, los sistemas actuales pueden lograr niveles de precisión comparables —o incluso superiores— al juicio humano en tareas específicas, especialmente cuando se dispone de grandes volúmenes de datos etiquetados y bien preparados [3].

1.1. Definición del trabajo

El presente Trabajo de Fin de Grado se enmarca en esta línea de investigación y tiene como objetivo explorar el uso de técnicas de minería de texto para resolver un problema concreto en el ámbito académico: predecir el año de publicación de Trabajos Fin de Grado (TFGs) a partir de su resumen. La motivación de este enfoque surge del hecho de que, aunque los TFGs disponibles en el Archivo Digital de la Universidad Politécnica de Madrid (AD-UPM) incluyen información sobre su año de publicación, dicha variable no ha sido tradicionalmente explorada desde una perspectiva lingüística o temática. Se plantea así la hipótesis de que el contenido textual de un TFG contiene información implícita que puede reflejar

su contexto temporal: vocabulario técnico asociado a tecnologías emergentes, temas de actualidad investigadora, enfoques metodológicos característicos de ciertos periodos, o incluso expresiones lingüísticas influenciadas por los cambios en la forma de redactar en el ámbito universitario. Si estas diferencias son suficientemente marcadas, un modelo de clasificación entrenado adecuadamente podría aprender a identificar patrones que permitan inferir el año aproximado de publicación de un documento sin necesidad de consultarlo explícitamente [4].

Este tipo de clasificación temporal de documentos presenta además retos interesantes desde el punto de vista metodológico. En primer lugar, no se trata de una clasificación por tema, sino por una variable contextual como es el tiempo, lo que obliga a tratar con clases posiblemente difusas o solapadas entre sí. En segundo lugar, el número de clases (años) puede ser elevado, lo cual convierte el problema en una clasificación multiclase con riesgo de desbalanceo si no se controla adecuadamente la distribución de la muestra. Finalmente, el proceso completo requiere la adquisición previa de datos textuales de calidad, lo cual implica implementar mecanismos de extracción, limpieza y normalización de los textos que no siempre están disponibles en origen de forma estructurada.

El alcance del proyecto abarca por tanto dos líneas principales y complementarias. Por un lado, la implementación de un sistema de adquisición de datos que permita recopilar, desde el AD-UPM, una muestra significativa de TFGs de diferentes titulaciones y años, extrayendo información clave como el título, resumen, año de publicación, grado, departamento y escuela. Por otro lado, el desarrollo de un modelo de clasificación supervisada capaz de aprender de esos datos y realizar predicciones sobre el año de publicación a partir del texto disponible. Ambas líneas se abordarán de forma integrada, ya que la calidad y representatividad de los datos será un factor clave para la eficacia del modelo.

1.2. Descripción del problema y planteamiento del trabajo

En esta sección se expone el contexto en el que se enmarca este trabajo, así como la motivación que justifica su desarrollo. Se describe el problema de investigación, su formulación como tarea de clasificación automática, y los supuestos y limitaciones que condicionan su alcance. El objetivo es establecer las bases conceptuales y prácticas sobre las que se construye el sistema propuesto.

1.2.1. Justificación y contexto

El crecimiento exponencial de documentos académicos en repositorios institucionales ha generado la necesidad de herramientas que permitan organizar, analizar y extraer valor de estos textos de manera automática. En particular, los Trabajos Fin de Grado (TFG) contienen información valiosa para detectar tendencias temáticas, líneas de investigación emergentes o patrones institucionales. Sin embargo, su clasificación temporal —por ejemplo, según el año de publicación— puede resultar difícil de automatizar debido a la naturaleza abierta y variable del lenguaje utilizado en los resúmenes de dichos trabajos [2].

Esta dificultad se ve acentuada por la distribución desigual de TFGs publicados a lo largo de los años, como puede observarse en la Figura 1.1, donde se representa gráficamente la cantidad de trabajos disponibles por año dentro del corpus analizado. Dicha visualización ha sido elaborada por la autora a partir de los datos del AD-UPM.

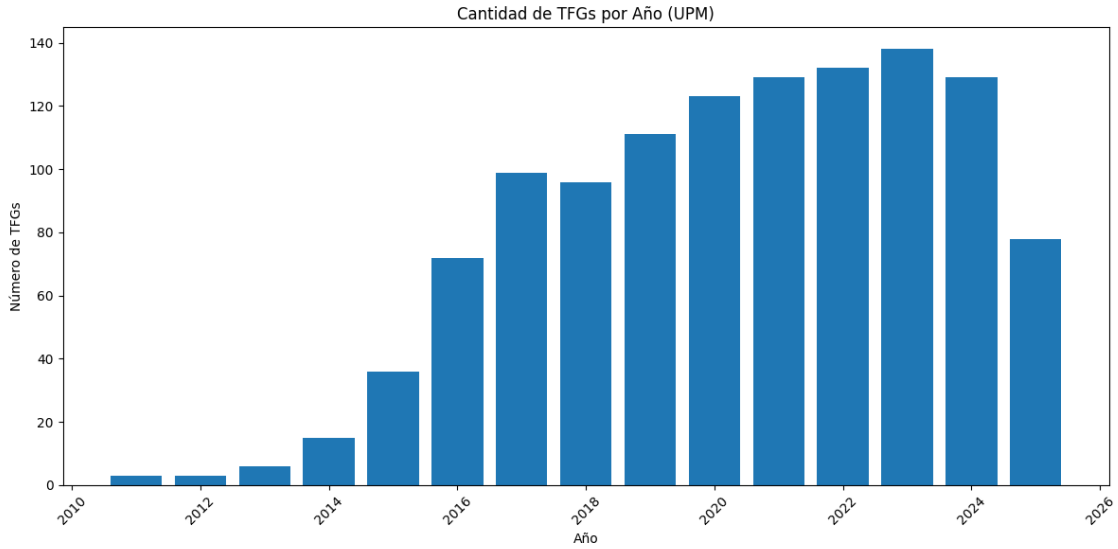

Figura 1.1: Distribución de Trabajos Fin de Grado por año en el conjunto de datos analizado.

Este trabajo se enmarca en la aplicación de técnicas de minería de texto y aprendizaje automático al análisis documental académico. El objetivo principal es desarrollar un sistema capaz de predecir el año de publicación de un TFG a partir de su resumen, utilizando únicamente dicha información textual y sin apoyarse en metadatos explícitos como la fecha [5].

1.3. Formulación del problema

Se plantea un problema de clasificación supervisada multiclase, donde a cada entrada textual (resumen) se le debe asignar una etiqueta correspondiente a un año concreto. Para ello, se requiere:

- Preprocesar los textos para reducir ruido lingüístico y normalizar el contenido.
- Representar los documentos mediante técnicas vectoriales que capturen información relevante.
- Entrenar modelos de clasificación que aprendan patrones discriminativos entre documentos de distintos años.
- Evaluar el rendimiento del sistema con métricas estándar del aprendizaje automático [3].

Dado que los textos no incluyen de forma explícita la fecha de publicación, se espera que el modelo sea capaz de identificar de forma implícita características lingüísticas, temáticas o estilísticas que correlacionen con el año de publicación.

La solución propuesta se estructura en torno a un flujo de procesamiento que recoge estas cuatro tareas fundamentales, como se resume en la Figura 1.2. Este esquema representa el *pipeline* completo del sistema desarrollado.

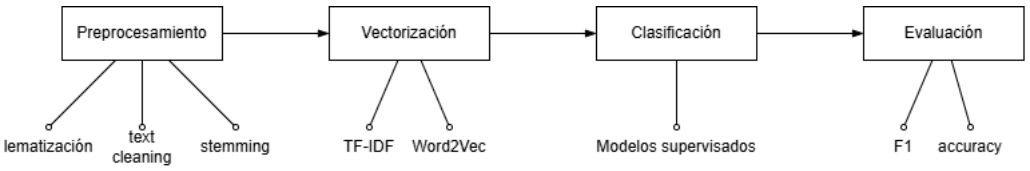

Figura 1.2: Flujo general del sistema de clasificación propuesto. Elaboración propia.

Los detalles técnicos de cada una de estas etapas pueden consultarse en los Anexos A–E, donde se documenta la implementación de cada uno de los pasos.

1.4. Supuestos y limitaciones

El sistema parte de algunos supuestos clave:

- Los resúmenes contienen suficiente información para inferir la fecha de publicación.
- Existe cierta evolución temporal en el estilo, el vocabulario o los temas tratados, que puede ser detectada por modelos computacionales.
- Los datos disponibles están correctamente etiquetados y segmentados por año, y son representativos del conjunto global de TFGs.

Entre las principales limitaciones destacan el desbalance de clases (años con más o menos trabajos), la ambigüedad semántica inherente a ciertos textos, y la dificultad de generalizar a otros dominios o instituciones con distintos estilos de redacción o temáticas.

1.5. Objetivos

Desde un punto de vista práctico, el trabajo tiene como finalidad demostrar la aplicabilidad de técnicas de minería de texto sobre datos reales del entorno universitario, utilizando herramientas actuales de procesamiento de lenguaje natural y aprendizaje automático. Más allá del objetivo técnico, este enfoque abre la puerta a nuevas formas de exploración y organización de los repositorios institucionales, fomentando el uso de la inteligencia artificial para enriquecer el acceso al conocimiento académico.

En base a este planteamiento, el trabajo persigue dos objetivos fundamentales que estructuran tanto la parte técnica como metodológica del proyecto.

- El **primer objetivo** es desarrollar un modelo de clasificación supervisada que sea capaz de predecir el año de publicación de un TFG a partir de una selección de contenidos en los documentos. Para ello, será necesario explorar y comparar distintas técnicas de representación de texto —como TF-IDF o modelos de lenguaje basados en *embeddings* [5]—, así como entrenar y evaluar diferentes algoritmos de clasificación multiclase, tales como máquinas de vectores de soporte (SVM), Naive Bayes o redes neuronales ligeras. El enfoque se centrará en maximizar la precisión del modelo y en analizar cómo varía su rendimiento en función de la calidad y características de los datos textuales utilizados.
- El **segundo objetivo** del trabajo consiste en garantizar que los datos sobre los que se construye el modelo sean adecuados y representativos. Esto implica implementar un sistema de adquisición automatizada de TFGs desde el Archivo Digital de la UPM, que permita extraer de manera sistemática los títulos, resúmenes y demás campos relevantes. Dado que esta información no siempre se presenta de forma homogénea, será necesario aplicar técnicas de limpieza, normalización y almacenamiento estructurado para disponer de un conjunto de datos coherente, reutilizable y de calidad suficiente para su análisis posterior.

Ambos objetivos están interconectados y se abordarán de forma conjunta a lo largo del proyecto. La calidad del modelo dependerá directamente de la calidad de los datos de entrada, por lo que se dedicará una atención especial a todo el proceso previo de adquisición y preparación, que forma parte integral de la solución propuesta.

1.6. Descripción del documento

El resto del documento se organiza de forma progresiva. En primer lugar, se presentan los trabajos previos que contextualizan el proyecto y justifican la elección metodológica. A continuación, se detallan los fundamentos teóricos y las herramientas utilizadas, seguidos del diseño general del sistema y la estrategia de adquisición de datos. Posteriormente, se describen las fases de análisis, preprocesamiento y codificación de texto, así como la construcción y evaluación del modelo de clasificación. Finalmente, se analiza el impacto del trabajo en relación con los Objetivos de Desarrollo Sostenible, y se exponen las conclusiones y posibles líneas de desarrollo futuro.

Capítulo 2

Trabajos previos

Antes de abordar el diseño y desarrollo de un sistema de clasificación de documentos basado en determinadas secciones de su contenido textual, resulta imprescindible revisar los antecedentes teóricos y técnicos que fundamentan este tipo de proyectos [2]. Esta sección tiene como objetivo contextualizar el trabajo en el marco de la literatura existente y presentar los enfoques metodológicos que han sido empleados con éxito en problemas similares. Se analizan los conceptos fundamentales de la minería de texto y su aplicación en la clasificación documental, prestando especial atención a los retos específicos que plantea el tratamiento de texto en lenguaje natural [1].

Asimismo, se revisan distintas estrategias de clasificación supervisada aplicadas en el ámbito del procesamiento de lenguaje natural (PLN), destacando sus características, ventajas y limitaciones [3]. Finalmente, se examina el estado del arte en tareas de clasificación temporal de documentos, ya que este trabajo no pretende clasificar por tema o categoría, sino por una variable contextual como es el año de publicación, lo que añade una dimensión analítica adicional [4].

El repaso de estos enfoques previos no solo permite justificar las decisiones tomadas en la implementación posterior, sino que también sirve para identificar herramientas, buenas prácticas y problemas comunes que deben ser tenidos en cuenta en el desarrollo del presente proyecto.

2.1. Minería de textos

2.1.1. Definición y contexto

La minería de texto (*text mining*) es una rama interdisciplinar que combina técnicas de procesamiento de lenguaje natural (PLN), recuperación de información, aprendizaje automático y estadística con el objetivo de descubrir patrones, relaciones y conocimiento útil a partir de grandes volúmenes de datos textuales no estructurados [2]. A diferencia de los datos estructurados que se almacenan en bases de datos convencionales, el texto libre presenta ambigüedad, redundancia y una considerable variabilidad lingüística, lo que requiere un tratamiento

específico antes de ser analizado computacionalmente [1].

Esta disciplina ha adquirido una gran relevancia en las últimas décadas debido al crecimiento exponencial de la información textual disponible en múltiples ámbitos: redes sociales, medios de comunicación digitales, documentos académicos, registros clínicos, opiniones de usuarios, correos electrónicos, entre otros [5]. Frente a esta sobreabundancia de datos textuales, la minería de textos permite automatizar procesos de análisis que tradicionalmente dependían de la lectura humana, facilitando así tareas como la categorización de contenidos, el resumen automático, la detección de opiniones o el descubrimiento de temas latentes.

Como se muestra en la Figura 2.1, el proceso de minería de textos suele seguir un flujo estructurado que abarca desde la adquisición del texto bruto hasta su análisis mediante modelos computacionales, pasando por etapas clave como la limpieza, transformación y representación del texto.

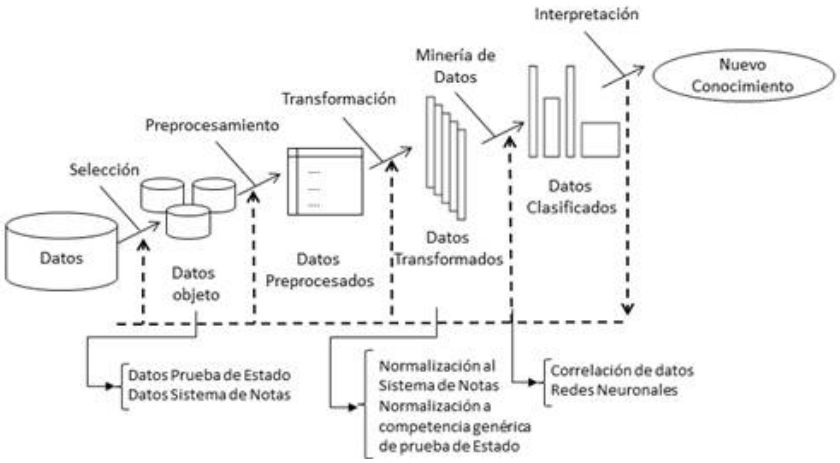

Figura 2.1: Flujo general de un sistema de minería de textos. Fuente: [6]

En el contexto del presente trabajo, la minería de texto se emplea como marco metodológico general que engloba las técnicas necesarias para convertir el texto libre —en este caso, los resúmenes de Trabajos Fin de Grado (TFGs)— en representaciones numéricas susceptibles de ser procesadas por algoritmos de clasificación automática [3]. A partir de dichas representaciones, se busca inferir información no explícita en los textos, como el año de publicación, basándose en patrones lingüísticos detectables a lo largo del tiempo.

2.1.2. Características del texto como fuente de datos

A diferencia de los datos estructurados que se almacenan en bases de datos relacionales y presentan un formato homogéneo (tablas, categorías, números), el texto libre es inherentemente desestructurado [7]. Esto lo convierte en una fuente de información rica, pero también compleja de procesar. Comprender las particularidades del texto como dato es esencial para diseñar un sistema de

análisis eficaz, especialmente en tareas como la clasificación documental. Esta diferencia entre ambos tipos de datos se ilustra en la Figura 2.2.

Figura 2.2: Comparativa entre datos estructurados y no estructurados. Fuente: [8]

Una primera característica distintiva del texto es su ambigüedad. Muchas palabras tienen significados múltiples que solo pueden desambiguarse a partir del contexto, lo que complica su interpretación automática [1]. A esto se suma la polisemia, la sinonimia, y la variabilidad léxica, que hacen que una misma idea pueda expresarse de múltiples formas diferentes.

El lenguaje natural también está sujeto a fenómenos de redundancia, informalidad, errores ortográficos, variaciones regionales o uso de tecnicismos, todo lo cual introduce ruido en los datos [5]. Este tipo de ruido puede impactar negativamente en la calidad de los modelos si no se aplican técnicas adecuadas de limpieza y normalización durante el preprocesamiento.

Otra característica clave es la alta dimensionalidad de los datos textuales una vez que son representados numéricamente. Por ejemplo, al utilizar técnicas como Bag-of-Words o TF-IDF, cada término del vocabulario se convierte en una dimensión del espacio vectorial, lo que puede dar lugar a representaciones con decenas de miles de variables. Esto hace necesario aplicar técnicas de selección o reducción de características para evitar problemas de sobreajuste y mejorar la eficiencia computacional [2].

Además, los textos tienen una estructura interna compleja: contienen sintaxis, relaciones semánticas, dependencias gramaticales, énfasis pragmáticos, etc., que pueden ser explotados por modelos más avanzados como los basados en redes neuronales o modelos de lenguaje preentrenados. Sin embargo, acceder a esa información requiere una fase de análisis lingüístico más profundo, que incluye tareas como el etiquetado gramatical, la lematización o el reconocimiento de entidades [1].

Por último, es importante destacar que el texto como fuente de datos suele venir acompañado de metadatos (autor, año, fecha, título, etc.) que pueden ser uti-

lizados como variables adicionales en los modelos. En el contexto del presente trabajo, solo se analizan los textos de entrada (resúmenes de TFGs), aunque la clasificación se centra específicamente en el año de publicación inferido a partir del contenido textual.

2.1.3. Flujo general de un sistema de minería de textos

Un sistema de minería de texto se compone de varias fases sucesivas que permiten transformar datos textuales no estructurados en información útil y estructurada para su análisis. Estas fases constituyen un flujo de trabajo que comienza con la recolección de los documentos y culmina en la obtención de resultados interpretables y, en muchos casos, visualizables. Comprender este flujo es esencial para planificar adecuadamente las tareas que conforman un proyecto de minería de textos [9].

La Figura 2.3 muestra de forma esquemática las principales etapas que componen este flujo, desde la adquisición de documentos hasta la evaluación final de resultados.

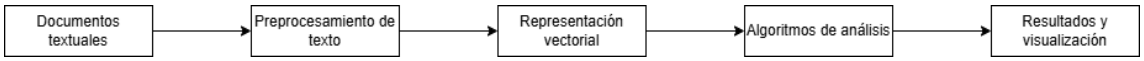

Figura 2.3: Diagrama del flujo general de un sistema de minería de textos

En primer lugar, se lleva a cabo la **selección y adquisición de documentos**, que consiste en recopilar los textos relevantes para el problema planteado. En el contexto de este trabajo, esta fase se traduce en la extracción de los resúmenes y años de TFGs, junto con otros metadatos, desde el repositorio institucional de la UPM, mediante técnicas de *web scraping*.

A continuación, se realiza el **preprocesamiento de texto**, una etapa crítica que implica la limpieza del contenido textual. Esto incluye la eliminación de caracteres no deseados, conversión a minúsculas, tokenización, lematización, y eliminación de palabras vacías (*stopwords*) [5], entre otras operaciones. El objetivo es reducir el ruido y normalizar el texto para facilitar su posterior análisis.

La siguiente fase es la **transformación y representación**, en la que los textos se convierten en vectores numéricos que puedan ser interpretados por los algoritmos de aprendizaje automático. Entre las técnicas más comunes se encuentran Bag-of-Words, TF-IDF y los modelos de *embeddings* (como Word2Vec o BERT).

Posteriormente, tiene lugar el **análisis propiamente dicho**, que puede incluir tareas como clasificación, agrupamiento, extracción de relaciones, resumen automático o detección de temas. En este trabajo, el objetivo principal es la clasificación multiclase por año de publicación.

Finalmente, se produce la fase de **interpretación y evaluación de resultados**, donde se aplican métricas cuantitativas y herramientas visuales para analizar el rendimiento del sistema, validar su eficacia y extraer conclusiones útiles a partir del modelo entrenado. En este punto, cobra especial relevancia la explicabilidad del clasificador, especialmente en modelos complejos. Métodos como

las explicaciones contrafactuales o las técnicas basadas en relevancia local y evidencia permiten aumentar la confianza en el sistema, facilitando su adopción en entornos sensibles [10, 11].

2.1.4. Dificultades y retos habituales

Aunque la minería de textos ha avanzado considerablemente en los últimos años gracias al desarrollo de nuevas herramientas y algoritmos, sigue presentando una serie de dificultades técnicas y metodológicas que deben ser tenidas en cuenta al abordar cualquier proyecto. Estos retos afectan tanto al procesamiento inicial del texto como a la fiabilidad de los modelos generados y a la interpretación de los resultados.

Una de las principales dificultades radica en la **ambigüedad y variabilidad del lenguaje natural**. A diferencia de los datos estructurados, el texto contiene múltiples formas de expresar una misma idea, lo que genera sinónimos, polisemia, ambigüedad contextual y dependencias gramaticales complejas. Esto obliga a utilizar técnicas avanzadas de preprocesamiento y modelos lingüísticos capaces de capturar el significado más allá de las palabras individuales [1].

Otro reto importante es la **alta dimensionalidad del espacio de representación**. Cuando los documentos se transforman en vectores numéricos —por ejemplo, mediante TF-IDF o Bag-of-Words—, se generan espacios con miles de dimensiones (tantas como palabras distintas). Esto puede llevar a problemas de sobreajuste y dificultades computacionales si no se aplican mecanismos de reducción o selección de características [2].

Además, es común enfrentarse a la **escasez o desbalanceo de datos etiquetados**. En tareas supervisadas como la clasificación, disponer de un conjunto equilibrado y representativo de ejemplos por cada clase es fundamental. Sin embargo, en muchos casos reales —como el presente trabajo— los datos no están etiquetados de forma explícita o hay años con muy pocos documentos, lo que puede afectar negativamente al rendimiento del modelo y requerir técnicas específicas para mitigar el desbalanceo, como el muestreo estratificado o el ajuste de pesos en el entrenamiento [12].

La **calidad del texto** también es un factor crítico. Los documentos pueden presentar errores tipográficos, estructuras gramaticales deficientes, uso de abreviaturas o formatos inconsistentes, especialmente si han sido generados en distintos contextos o por distintos autores. Esto complica el proceso de limpieza y normalización previo al análisis [5].

Por otro lado, los **problemas éticos y legales** asociados al uso de ciertos datos también representan un reto. Aunque los datos textuales disponibles públicamente, como los del AD-UPM, pueden ser utilizados con fines académicos, es importante garantizar el cumplimiento de las normativas de uso y respetar la confidencialidad de los autores si se tratara de textos sensibles.

Por último, cabe mencionar el reto de la **interpretabilidad**. Muchos de los modelos más avanzados, especialmente los basados en redes neuronales profundas o

transformadores, funcionan como "cajas negras", lo que dificulta explicar cómo se ha llegado a una determinada clasificación [13]. Esto puede limitar su aplicación en entornos donde la transparencia del proceso es un requisito, como en ámbitos legales o educativos.

Todos estos retos se encuentran, en mayor o menor medida, presentes en el desarrollo del presente Trabajo de Fin de Grado. Abordarlos adecuadamente resulta esencial para garantizar la validez del sistema propuesto y su posible reutilización futura en contextos similares.

2.2. Técnicas de clasificación supervisada

La clasificación supervisada constituye una de las aproximaciones más utilizadas en tareas de aprendizaje automático aplicadas al procesamiento de textos. En esta sección se revisan sus fundamentos teóricos, así como algunos de los modelos clásicos empleados en la literatura para abordar este tipo de problemas.

2.2.1. Concepto y principios fundamentales

La clasificación supervisada es una técnica de aprendizaje automático cuyo objetivo es asignar una o varias etiquetas a una instancia de datos, a partir de un conjunto de ejemplos previamente etiquetados [14]. En este paradigma, el modelo aprende una función de decisión a partir de un conjunto de entrenamiento, compuesto por pares (x_i, y_i), donde x_i representa una instancia (por ejemplo, un documento de texto vectorizado) y y_i es su clase asociada.

El término "supervisado" hace referencia al hecho de que el modelo recibe una guía explícita durante el aprendizaje, basada en los ejemplos proporcionados. Esto contrasta con otros enfoques como el *aprendizaje no supervisado*, donde se utilizan técnicas como el análisis de conglomerados o *clustering* para identificar estructuras ocultas en los datos [15], o el *aprendizaje por refuerzo*, basado en la toma de decisiones secuenciales mediante recompensas y penalizaciones [16]. En el caso concreto de la clasificación de documentos, esta guía se traduce en un corpus de textos anotados con la categoría que les corresponde: temática, polaridad, nivel educativo, autoría, año de publicación, etc.

El proceso de clasificación supervisada implica varias etapas fundamentales. En primer lugar, es necesario preparar un conjunto de datos de entrenamiento de calidad, que sea representativo del problema y que esté adecuadamente equilibrado en cuanto a las clases. A continuación, los documentos deben ser transformados en una representación numérica, generalmente mediante técnicas de vectorización como TF-IDF o modelos de *embeddings* [5]. Sobre esta representación se entrena un modelo, que puede ser más o menos complejo según la naturaleza del problema.

Durante el entrenamiento, el algoritmo ajusta sus parámetros internos para minimizar una función de pérdida [3] que mide la discrepancia entre las predicciones y las clases reales. Una vez entrenado, el modelo se evalúa sobre un

conjunto de validación o prueba para estimar su capacidad de generalización, es decir, su rendimiento ante textos no vistos previamente.

La clasificación supervisada es especialmente adecuada para tareas donde se dispone de datos históricos etiquetados, y donde las clases son conocidas y bien definidas. En el contexto de este trabajo, se aplica para predecir el año de publicación de un Trabajo Fin de Grado (TFG) a partir de su resumen, formulando el problema como una clasificación multiclase. Cada clase representa un año específico, lo que añade complejidad adicional debido a la proximidad semántica entre clases adyacentes y al posible desbalance de ejemplos entre años.

2.2.2. Modelos clásicos de clasificación de texto

A lo largo del tiempo, se han desarrollado diversos algoritmos de clasificación supervisada que han demostrado ser especialmente eficaces en el tratamiento de texto. Estos modelos se han aplicado con éxito en tareas como clasificación temática de noticias, detección de spam, análisis de sentimientos [2] o, como en este caso, inferencia temporal a partir de contenido textual. A continuación, se describen los modelos clásicos más representativos.

Naive Bayes: se trata de un clasificador probabilístico muy simple que puede representarse como un modelo gráfico, y que opera bajo la hipótesis de independencia condicional entre las características del texto dado su clase. Aunque esta suposición rara vez se cumple en la práctica, el clasificador Naive Bayes suele ofrecer un rendimiento sorprendentemente alto en tareas de clasificación de texto, especialmente cuando se trabaja con representaciones de tipo Bag-of-Words o TF-IDF [17]. Su rapidez y simplicidad lo hacen ideal como línea base comparativa.

Máquinas de vectores de soporte (SVM): las SVM buscan encontrar el hiperplano que mejor separa las clases en un espacio de alta dimensión. Este tipo de modelos han mostrado excelentes resultados en clasificación de texto gracias a su capacidad para manejar grandes cantidades de características (palabras) sin perder eficiencia. Las SVM son particularmente efectivas cuando las clases son linealmente separables o casi separables en el espacio vectorial generado por la representación textual [18]. Un ejemplo gráfico de este tipo de separación se muestra en la Figura 2.4.

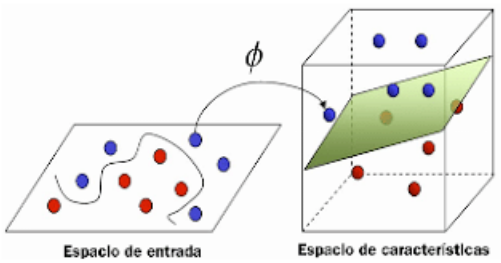

Figura 2.4: Máquina de vectores de soporte. Fuente: [19]

Árboles de decisión y Random Forests: los árboles de decisión construyen un modelo jerárquico basado en reglas de decisión simples sobre las características del texto. Su principal ventaja radica en la **interpretabilidad**, ya que permiten visualizar claramente cómo se toman las decisiones de clasificación. Si bien pueden ser menos efectivos que otros algoritmos con representaciones dispersas y de alta dimensionalidad, como es habitual en texto, los métodos basados en conjuntos como Random Forests o Gradient Boosted Trees han demostrado mejoras en estabilidad y rendimiento [20], especialmente cuando se combinan con técnicas de reducción de características. La Figura 2.5 muestra una representación esquemática del modelo.

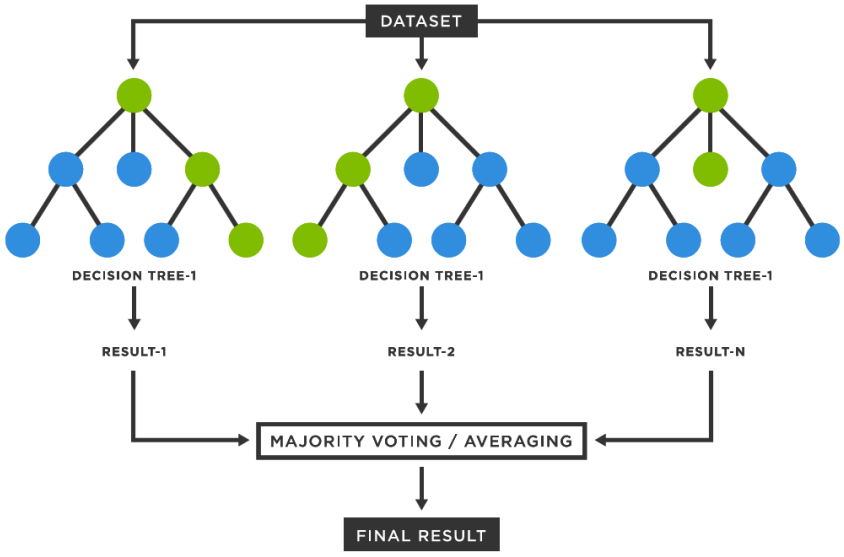

Figura 2.5: Algoritmo de Random Forest. Fuente: [21]

XGBoost (Extreme Gradient Boosting): es una técnica de ensamblado basada en árboles de decisión optimizados mediante el algoritmo de gradient boosting [22]. Se caracteriza por su eficiencia, capacidad de generalización y excelente rendimiento en competiciones de aprendizaje automático. XGBoost incorpora mecanismos de regularización [22], paralelización y manejo de valores ausentes, lo que lo hace especialmente robusto frente a sobreajuste y eficaz incluso con datos de alta dimensionalidad, como los representados por TF-IDF o embeddings. Por estas razones, se ha convertido en uno de los algoritmos más utilizados en tareas de clasificación complejas. La Figura 2.6 muestra de forma esquemática el funcionamiento general de este tipo de modelos mediante una secuencia de clasificadores encadenados.

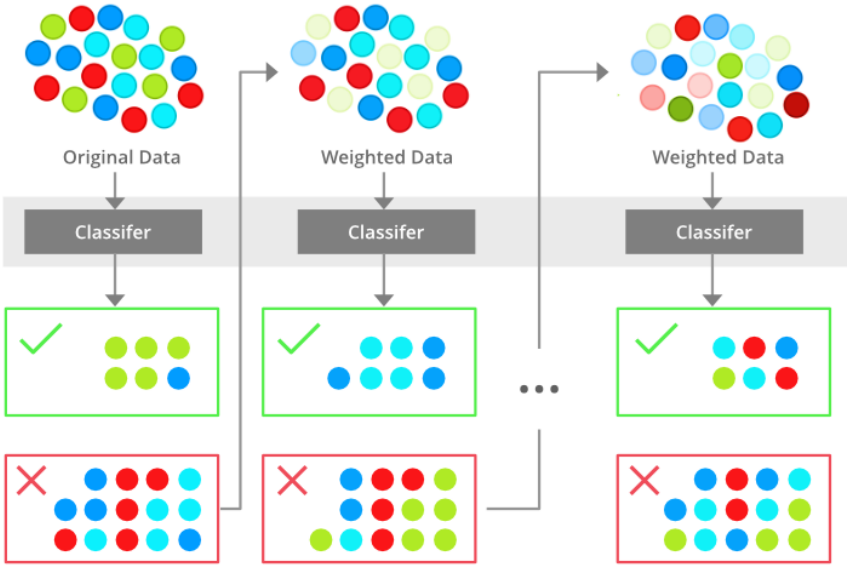

Figura 2.6: Funcionamiento general del algoritmo XGBoost. Fuente: [23]

K-Nearest Neighbors (KNN): este algoritmo asigna a un nuevo documento la clase más común entre sus k vecinos más cercanos en el espacio vectorial. Aunque es intuitivo y fácil de implementar, su rendimiento se ve afectado en espacios de alta dimensionalidad y su coste computacional es elevado para grandes volúmenes de datos. Por este motivo, se utiliza más frecuentemente como referencia teórica [24] que como solución práctica en contextos reales. Un ejemplo visual del algoritmo puede verse en la Figura 2.7.

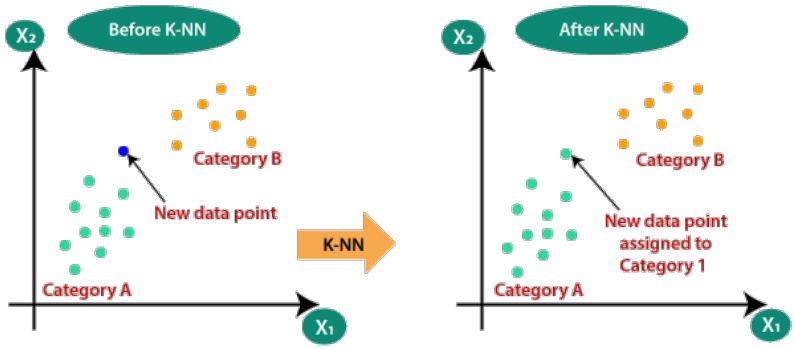

Figura 2.7: Ejemplo del algoritmo de K-Nearest Neighbors. Fuente: [25]

La elección del modelo dependerá de múltiples factores, como la cantidad y calidad de los datos disponibles, la distribución de las clases, la representación elegida del texto y los requisitos computacionales del entorno. En este Trabajo de Fin de Grado se evaluarán algunos de estos modelos sobre un corpus de resúmenes de TFGs, comparando su rendimiento en la tarea de predicción del año de publicación.

2.3. Clasificación temporal de documentos

La clasificación de documentos según su dimensión temporal, es decir, su datación o pertenencia a un periodo concreto, ha recibido una atención creciente en los últimos años dentro del ámbito del procesamiento de lenguaje natural. A diferencia de las tareas más habituales como la clasificación temática, el análisis de sentimientos o la detección de entidades, la clasificación temporal plantea un conjunto distinto de retos y se basa en la hipótesis de que el lenguaje, el estilo y el contenido de los textos reflejan, de forma implícita, el momento en que fueron redactados.

Uno de los primeros enfoques en este campo consistió en la datación de documentos históricos o literarios a partir de su estilo lingüístico. Por ejemplo, de Jong et al. [26] propusieron modelos de lenguaje con sensibilidad temporal para analizar textos históricos, mientras que Kanhabua y Nejdl [27] investigaron la identificación de intervalos temporales relevantes en búsquedas con conciencia del tiempo. En ambos casos, se emplearon métodos probabilísticos basados en la frecuencia de términos y patrones lingüísticos para estimar la fecha de creación de los textos. Más recientemente, el uso de modelos de aprendizaje automático ha permitido aplicar estas ideas a grandes colecciones de noticias, publicaciones académicas o entradas en redes sociales.

En el ámbito académico, algunos estudios han utilizado la datación automática de artículos científicos o tesis como herramienta para el análisis de la evolución de una disciplina. La hipótesis común en estos trabajos es que el lenguaje científico cambia con el tiempo, ya sea por la aparición de nuevos términos técnicos, la popularidad de ciertas metodologías o la evolución temática de las áreas de investigación. De hecho, investigaciones recientes han demostrado que incluso en textos breves como los títulos o los resúmenes es posible detectar señales temporales suficientes para predecir su fecha con una precisión razonable [4].

Los enfoques metodológicos utilizados en clasificación temporal de documentos incluyen modelos clásicos como Naive Bayes, máquinas de vectores de soporte (SVM) o regresión logística, pero también técnicas más avanzadas como modelos basados en redes neuronales recurrentes, transformers o embeddings temporales [28]. En todos los casos, la clave está en capturar patrones léxicos, estilísticos o temáticos que tengan correlación con una dimensión temporal.

Sin embargo, uno de los retos principales de esta tarea es el solapamiento de estilos entre años adyacentes, especialmente cuando las clases (años) están muy próximas o desbalanceadas. Esto hace que el problema no solo sea multiclase, sino que tenga una estructura ordinal implícita que no siempre se aprovecha en los modelos tradicionales.

En el caso del presente Trabajo de Fin de Grado, se adopta un enfoque supervisado para predecir el año de publicación de Trabajos Fin de Grado a partir de sus resúmenes y títulos. Aunque este tipo de clasificación temporal en el ámbito de TFGs no está ampliamente explorado en la literatura, los trabajos previos sobre datación de textos científicos y clasificación cronológica de documentos proporcionan una base sólida sobre la que construir y validar esta propuesta.

Así, el proyecto se sitúa en una línea de investigación emergente con aplicaciones potenciales en análisis de tendencias, recuperación de información temporal y enriquecimiento semántico de repositorios académicos.

Capítulo 3

Marco teórico y tecnologías

Este capítulo presenta los fundamentos teóricos y las herramientas técnicas que sustentan el desarrollo del sistema propuesto. En primer lugar, se revisan los conceptos clave del Procesamiento del Lenguaje Natural (PLN) y las técnicas utilizadas para representar texto de forma computacional. A continuación, se describen los principales algoritmos de clasificación supervisada aplicados en el proyecto. Finalmente, se detallan las librerías y tecnologías empleadas en cada fase del flujo de trabajo, desde la recolección de datos hasta la evaluación del modelo.

3.1. Procesamiento del Lenguaje Natural (PLN)

El Procesamiento del Lenguaje Natural (PLN) es una rama de la inteligencia artificial que tiene como objetivo permitir que los sistemas informáticos interpreten, comprendan y generen lenguaje humano. Su aplicación es fundamental en tareas como análisis de sentimientos, traducción automática, extracción de información y, como en este caso, clasificación de documentos textuales [2].

Dentro del PLN, una de las primeras etapas es el *preprocesamiento* del texto, que permite limpiar y normalizar los datos lingüísticos antes de analizarlos. Las técnicas más comunes incluyen la tokenización (división del texto en palabras o unidades menores), la eliminación de palabras vacías (*stopwords*), la conversión a minúsculas, la supresión de signos de puntuación, números, acentos y símbolos especiales, así como la normalización léxica mediante lematización o *stemming* [5].

La **lematización** consiste en reducir una palabra a su forma canónica o lema (por ejemplo, “corriendo” se transforma en “correr”), respetando su categoría gramatical. El **stemming**, en cambio, recorta las palabras hasta obtener una raíz común, sin necesidad de que esta sea una palabra real (por ejemplo, “correr”, “corriendo” y “corre” pueden reducirse a “corr”). Ambas técnicas ayudan a reducir la dimensionalidad del texto y a identificar conceptos recurrentes en distintos documentos [1].

Gracias a estas tareas, el texto —originalmente no estructurado— puede transformarse en una representación más uniforme y adecuada para su análisis computacional mediante modelos de aprendizaje automático.

3.2. Representación vectorial de texto

Una vez preprocesado, el texto debe convertirse en una representación numérica que los algoritmos de clasificación puedan interpretar. Este proceso se conoce como *representación vectorial de texto* y es clave en tareas de minería de texto y aprendizaje automático [2].

Entre las técnicas más utilizadas se encuentran:

- **TF-IDF (Term Frequency – Inverse Document Frequency)**: representa cada documento según la frecuencia de aparición de sus palabras, ponderada por su rareza en el conjunto total de documentos. Esto permite dar más peso a términos relevantes que no aparecen con frecuencia en todos los textos. Es una técnica eficaz y ampliamente adoptada en sistemas de recuperación de información y clasificación [5].
- **Word2Vec**: es un método de incrustación semántica que convierte palabras en vectores densos de baja dimensión. A través de modelos como *Skip-Gram* o *CBOW*, Word2Vec captura similitudes semánticas entre palabras basándose en sus contextos de aparición. Esta representación es especialmente útil para analizar relaciones entre conceptos y mejorar la capacidad de generalización de los modelos [1].

Ambas técnicas tienen enfoques distintos: mientras que TF-IDF genera vectores dispersos centrados en la frecuencia de los términos, Word2Vec produce vectores densos que capturan relaciones semánticas. La elección entre una y otra depende del tipo de tarea, la naturaleza del corpus y las características del modelo de aprendizaje utilizado.

3.3. Algoritmos de clasificación supervisada

En este proyecto se han implementado y evaluado cuatro algoritmos de clasificación supervisada, cada uno con un enfoque distinto respecto al manejo de datos textuales vectorizados. Todos ellos se han entrenado sobre representaciones TF-IDF y Word2Vec, y se han evaluado con las mismas métricas para facilitar la comparación. A continuación se detallan sus características y las herramientas utilizadas. El código fuente utilizado para implementar y entrenar estos modelos puede consultarse en el Apéndice E.

3.3.1. K-Nearest Neighbors (K-NN)

En este trabajo, se aplicó el algoritmo K-NN con $k = 5$ vecinos, utilizando la implementación proporcionada por `scikit-learn`. Este valor se eligió como compromiso entre estabilidad y sensibilidad: valores pequeños (como $k = 1$) pueden

resultar muy sensibles al ruido y a ejemplos atípicos, mientras que valores demasiado grandes pueden diluir las clases minoritarias y provocar predicciones menos precisas. El valor $k = 5$ es una elección habitual en la literatura por ofrecer un buen equilibrio entre sesgo y varianza en la mayoría de problemas de clasificación [2].

El modelo se aplicó sobre representaciones vectoriales obtenidas mediante TF-IDF y Word2Vec, combinadas con dos variantes de preprocesamiento: lematización y *stemming*. En total, se evaluaron las siguientes configuraciones:

- K-NN con TF-IDF y lematización
- K-NN con TF-IDF y *stemming*
- K-NN con Word2Vec y lematización
- K-NN con Word2Vec y *stemming*

Dado que K-NN se basa en distancias, el tipo de vectorización tiene un impacto significativo en su rendimiento. En general, se espera que Word2Vec proporcione una ventaja al capturar mejor la similitud semántica entre documentos, mientras que TF-IDF suele ser más eficaz cuando existen términos distintivos entre clases.

3.3.2. Naive Bayes

En este trabajo se aplicaron dos variantes del algoritmo Naive Bayes, seleccionadas en función del tipo de representación vectorial:

- **Multinomial Naive Bayes**: empleado con vectores TF-IDF, ya que esta variante está diseñada para datos de conteo o frecuencia. Se adapta especialmente bien a la representación basada en ocurrencias de palabras.
- **Gaussian Naive Bayes**: utilizado con Word2Vec, dado que esta representación genera vectores densos y continuos, lo que se ajusta mejor a la suposición de distribución normal de esta variante.

Ambas versiones del algoritmo se evaluaron con las dos variantes de preprocesamiento (lematización y *stemming*), lo que dio lugar a cuatro combinaciones distintas:

- Naive Bayes con TF-IDF y lematización (Multinomial)
- Naive Bayes con TF-IDF y *stemming* (Multinomial)
- Naive Bayes con Word2Vec y lematización (Gaussian)
- Naive Bayes con Word2Vec y *stemming* (Gaussian)

Naive Bayes resulta especialmente útil en contextos donde los datos son escasos o donde se requiere rapidez tanto en entrenamiento como en predicción. Por ello, suele utilizarse como línea base en tareas de clasificación de texto [2].

3.3.3. Random Forest

En este trabajo se utilizó la implementación de Random Forest incluida en la biblioteca `scikit-learn`, configurada con un número de estimadores igual a 100 (`n_estimators=100`). Este valor es comúnmente utilizado en la literatura, ya que proporciona un equilibrio adecuado entre estabilidad en las predicciones y coste computacional. Si bien aumentar este valor puede mejorar la precisión, el beneficio tiende a estabilizarse más allá de cierto umbral [3].

El modelo se aplicó sobre las representaciones vectoriales generadas mediante TF-IDF y Word2Vec, empleando ambas variantes de preprocesamiento (lematización y *stemming*). En total, se entrenaron las siguientes configuraciones:

- Random Forest con TF-IDF y lematización
- Random Forest con TF-IDF y *stemming*
- Random Forest con Word2Vec y lematización
- Random Forest con Word2Vec y *stemming*

Random Forest resulta especialmente adecuado en contextos de alta dimensionalidad como los generados por TF-IDF. Su arquitectura basada en múltiples árboles permite modelar relaciones complejas sin necesidad de una calibración exhaustiva de hiperparámetros, lo que lo convierte en una opción robusta y eficaz para tareas de clasificación de texto [3].

3.3.4. XGBoost

En este trabajo se utilizó la implementación proporcionada por la librería `xgboost`, empleando los parámetros por defecto salvo por la métrica de evaluación, que se fijó como `eval_metric='mlogloss'`. Esta métrica calcula la pérdida logística multiclase y resulta especialmente adecuada para tareas como la predicción del año de publicación, donde las clases son múltiples y no ordinales.

XGBoost se entrenó sobre las cuatro combinaciones de representación vectorial y preprocesamiento, igual que el resto de modelos:

- XGBoost con TF-IDF y lematización
- XGBoost con TF-IDF y *stemming*
- XGBoost con Word2Vec y lematización
- XGBoost con Word2Vec y *stemming*

Gracias a su estructura basada en boosting secuencial y regularización incorporada, XGBoost es especialmente eficaz cuando existen patrones complejos y sutiles en los datos. Por esta razón, se ha convertido en uno de los algoritmos de referencia en tareas exigentes de clasificación de texto [3].

En conjunto, estos cuatro modelos ofrecen enfoques complementarios: K-NN explota la proximidad geométrica entre documentos; Naive Bayes modela proba-

bilidades a partir de frecuencias; Random Forest construye un conjunto robusto de árboles independientes; y XGBoost optimiza árboles encadenados con regularización avanzada. Su rendimiento relativo ha dependido de la representación utilizada y de los parámetros ajustados en cada caso, como se analiza en los capítulos siguientes.

3.4. Herramientas y librerías utilizadas

El desarrollo del proyecto se ha realizado íntegramente en **Python 3.10** como lenguaje de programación y entorno principal, por su sintaxis clara y su amplio ecosistema orientado al análisis de datos y aprendizaje automático. A continuación, se enumeran las librerías y herramientas empleadas, agrupadas según la etapa del flujo de trabajo en la que intervienen:

1. Extracción y almacenamiento de datos
    - **requests** (v2.31.0) [29]: permite realizar peticiones HTTP de manera sencilla. Se ha utilizado para acceder automáticamente al repositorio web de TFGs.
    - **BeautifulSoup** (v4.12.2) [30]: facilita la navegación y extracción de contenido HTML. Se ha utilizado junto a `requests` para obtener los resúmenes.
    - **re**: módulo estándar para trabajar con expresiones regulares. Se ha utilizado para limpiar y filtrar contenido textual durante la extracción.
    - **pymongo** (v4.6.1) [31]: cliente oficial para conectarse a MongoDB desde Python. Ha servido para almacenar los TFGs extraídos como documentos estructurados.
2. Preprocesamiento lingüístico
    - **nltk** (v3.8.1) [32]: biblioteca fundamental para el procesamiento del lenguaje natural. Se ha utilizado para tokenización, eliminación de *stopwords* y stemming.
    - **spaCy** (v3.7.2) [33]: herramienta moderna para PLN que ofrece modelos entrenados para distintos idiomas. Se ha empleado especialmente para la lematización en español.
    - **stopwords** y **punkt** (de `nltk`): recursos necesarios para eliminar palabras vacías comunes y realizar segmentación léxica.
    - **SnowballStemmer** (de `nltk.stem.snowball`): permite aplicar *stemming* en varios idiomas, incluyendo español.
3. Representación vectorial del texto
    - **TfidfVectorizer** (de `scikit-learn` v1.4.2) [34]: transforma documentos en vectores dispersos ponderados según la frecuencia e importancia relativa de cada término.

- **gensim.models.Word2Vec** (v4.3.2) [35]: genera vectores densos a partir de contextos semánticos utilizando modelos de *Skip-Gram* o *CBOW*.
- **VarianceThreshold** (de `sklearn.feature_selection`): técnica de filtrado que elimina características con varianza baja.

4. Modelado y evaluación
   - **scikit-learn** (v1.4.2) [34]: biblioteca integral para aprendizaje automático. Se han utilizado los clasificadores `KNeighborsClassifier`, `MultinomialNB`, `GaussianNB` y `RandomForestClassifier`, así como herramientas de validación y métricas de evaluación.
   - **xgboost** (v2.0.3) [36]: implementación optimizada del algoritmo de *gradient boosting*. Se ha utilizado por su alto rendimiento en tareas de clasificación.
   - **LabelEncoder** (de `sklearn.preprocessing`): codifica etiquetas categóricas (años) como valores numéricos para los modelos.
   - `train_test_split`, `f1_score`, `accuracy_score`, `precision_score`, `recall_score`: funciones clave de `sklearn` utilizadas para dividir el dataset y evaluar los resultados obtenidos.
5. Manipulación y análisis de datos
   - **pandas** (v2.2.2) [37]: permite gestionar estructuras de datos tabulares (DataFrames) de forma eficiente. Se ha utilizado para la limpieza, inspección y análisis de resultados.
   - **numpy** (v1.26.4) [38]: proporciona estructuras y operaciones matemáticas básicas para cálculos vectoriales y matriciales. Se ha utilizado en múltiples etapas del proyecto.

Capítulo 4

Preparación de los datos

Antes de entrenar cualquier modelo de clasificación, es imprescindible preparar adecuadamente los datos de entrada. En este capítulo se describen las distintas fases que componen esta etapa crítica del pipeline, desde la recolección inicial de los resúmenes hasta su transformación en vectores numéricos listos para ser procesados por algoritmos de aprendizaje automático. Se abordan aspectos clave como la extracción automatizada desde el repositorio institucional, el preprocesamiento lingüístico de los textos, las técnicas de representación semántica utilizadas (TF-IDF y Word2Vec) y la reducción de dimensionalidad aplicada para mejorar la eficiencia y estabilidad del sistema.

4.1. Extracción y almacenamiento de los datos

Los datos utilizados en este trabajo fueron obtenidos del **Archivo Digital de la Universidad Politécnica de Madrid (UPM)**[1], un repositorio institucional de acceso abierto que recoge documentos académicos de diversa índole, incluyendo los Trabajos Fin de Grado (TFG) de múltiples titulaciones.

Para llevar a cabo la extracción, se seleccionaron cuatro titulaciones con presencia regular en el repositorio y una cobertura temporal suficiente entre los años 2016 y 2024. Las titulaciones escogidas fueron:

- **Grado en Ciencias de la Actividad Física y del Deporte** `https://oa.upm.es/view/degree/Grado_en_Ciencias_de_la_Actividad_F=EDsica_y_del_Deporte.html`
- **Grado en Fundamentos de la Arquitectura** `https://oa.upm.es/view/degree/Grado_en_Fundamentos_de_la_Arquitectura.html`
- **Grado en Ingeniería en Tecnologías Industriales** `https://oa.upm.es/view/degree/Grado_en_Ingenier=EDa_en_Tecnolog=EDas_Industriales.html`

[1] `https://oa.upm.es/`

- **Grado en Ingeniería Informática** https://oa.upm.es/view/degree/Grado_en_Ingenier=EDa_Inform=E1tica.html

Para garantizar un conjunto de datos balanceado, se extrajeron **11 TFGs por año**, cubriendo el período entre 2016 y 2024. Esta selección fue cuidadosamente realizada, ya que algunas titulaciones no contaban con suficientes trabajos publicados en todos los años, lo que obligó a filtrar y elegir aquellas con disponibilidad homogénea en la década considerada.

De cada TFG se extrajo la siguiente información:

- Título
- Grado
- Año de publicación
- Escuela
- Departamento
- Resumen
- URL del trabajo en el repositorio

El Listado 4.1 muestra un ejemplo representativo del formato en el que se almacena cada entrada de TFG en la base de datos *MongoDB*, con los campos ya estructurados y normalizados tras el proceso de extracción y preprocesamiento.

```
{
  "_id": ObjectId('682b6c7c75f2284367053467'),
  "Título": "Actividad física después de la retirada de
     deportistas profesionales",
  "Grado": "Ciencias de la Actividad Física y del Deporte",
  "Ano": 2024,
  "Escuela": "Facultad de Ciencias de la Actividad Física y del
     Deporte (INEF) (UPM)",
  "Departamento": "Deportes",
  "Resumen": "La actividad física es uno de los pilares más
     importantes del concepto..."
  "URL": "https://oa.upm.es/82164/",
  "resumen_lematizado": "actividad físico pilar importante
     concepto vida saludable longeir ocur...",
  "resumen_stemming": "activ fisic pilar import concept vid
     salud longev ocurr deport profes..."
}
```

Listing 4.1: Ejemplo de entrada estructurada de un TFG almacenada en MongoDB

La extracción se realizó mediante un proceso automatizado de **web scraping**, utilizando las librerías `requests` y `BeautifulSoup` para acceder a las páginas web y extraer el contenido. Se aplicaron expresiones regulares con `re` para filtrar el texto y aislar los campos relevantes. Toda la información recolectada se almacenó estructuradamente en una base de datos `MongoDB`, utilizando la librería `pymongo`, lo que permitió su posterior consulta y procesamiento de manera eficiente (véase Apéndice A para más detalles sobre la implementación).

4.2. Preprocesamiento del texto

Antes de aplicar técnicas de representación vectorial o entrenar modelos de clasificación, es necesario transformar los textos originales en versiones más limpias y normalizadas. Este proceso de preprocesamiento permite reducir el ruido lingüístico, homogeneizar las expresiones y facilitar la extracción de patrones relevantes por parte de los algoritmos.

El preprocesamiento realizado sobre los resúmenes extraídos incluyó las siguientes etapas:

- **Conversión a minúsculas**: todos los caracteres se pasaron a minúsculas para evitar distinciones artificiales entre palabras iguales con diferente capitalización.
  - Antes: "`Redes Neuronales`"
  - Después: "`redes neuronales`"
- **Eliminación de caracteres no alfabéticos**: se eliminaron signos de puntuación, números y otros caracteres especiales, conservando únicamente letras del alfabeto.
  - Antes: "`educación 4.0: ¿necesidad o moda?`"
  - Después: "`educación necesidad o moda`"
- **Tokenización**: se dividió el texto en unidades léxicas (palabras), utilizando los módulos `nltk` y `spaCy`, adaptados al español.
  - Antes: "`Las redes neuronales están revolucionando la educación.`"
  - Después: [“Las”, “redes”, “neuronales”, “están”, “revolucionando”, “la”, “educación”]
- **Eliminación de *stopwords* en español e inglés**: se eliminaron palabras vacías o de escaso valor semántico, como “el”, “de”, “que” o “the”, “and”, “with”. Esta decisión responde a que, aunque los TFGs están mayoritariamente redactados en español, muchos resúmenes contienen términos o fragmentos en inglés (términos técnicos, nombres de herramientas, etc.), por lo que se aplicó un filtro multilingüe utilizando el corpus de `nltk` para ambos idiomas.

  - Antes: `["la", "educación", "es", "importante", "en", "el", "mundo"]`
  - Después: `["educación", "importante", "mundo"]`

- **Lematización**: se transformaron las palabras a su forma base o lema, manteniendo su categoría gramatical. Para ello, se utilizó el modelo `es_core_news_sm` de `spaCy`, entrenado específicamente para el español.
  - Antes: `["trabajo", "presenta", "estudio", "redes", "neuronales", "aplicacion", ...]`
  - Después: `["trabajo", "presentar", "estudio", "red", "neuronal", "aplicación", ...]`
- **Stemming**: como alternativa a la lematización, también se aplicó *stemming* mediante el `SnowballStemmer` de `nltk`. Esta técnica recorta las palabras hasta obtener una raíz común, sin necesidad de que sea una palabra real.
  - Antes: `["educación", "educativo"]`
  - Después: `["educ"]`

Se generaron dos versiones del corpus preprocesado: una utilizando lematización y otra utilizando *stemming*. Esta dualidad permitió posteriormente evaluar el impacto de cada técnica sobre los resultados de clasificación.

Todas las transformaciones fueron implementadas en Python mediante las librerías `nltk`, `spaCy` y `re`. Este preprocesamiento es una fase esencial en el pipeline de análisis textual, ya que mejora notablemente la calidad de la representación semántica de los documentos [39].

El código completo que implementa estas transformaciones puede consultarse en el Apéndice B.

Una vez finalizado el preprocesamiento, ambas versiones (lematizada y stemeada) de cada resumen fueron almacenadas en la base de datos. Esto permitió reutilizarlas posteriormente durante las etapas de vectorización y clasificación, sin necesidad de repetir el procesamiento.

4.3. Representación vectorial del texto

Una vez preprocesado el corpus textual, es necesario convertir cada resumen en una representación numérica que pueda ser interpretada por los algoritmos de clasificación. Esta transformación se conoce como representación vectorial y constituye una de las fases más importantes del pipeline de procesamiento de lenguaje natural.

En este trabajo se han utilizado dos técnicas complementarias: **TF-IDF** y **Word2Vec**. A continuación se describe cada una, junto con ejemplos simplificados de su funcionamiento.

TF-IDF (Term Frequency - Inverse Document Frequency)

TF-IDF es una técnica que asigna un peso a cada término en función de su frecuencia en un documento y su rareza en el conjunto completo de documentos. Cuanto más frecuente es una palabra en un documento pero más rara en los demás, mayor será su valor TF-IDF.

Este método produce vectores dispersos, donde cada dimensión representa un término del vocabulario. Es especialmente útil para identificar palabras clave que caracterizan un documento frente al resto [5].

Ejemplo:

- Texto original: `["educación", "neuronales", "educación"]`
- Vector TF-IDF simplificado: `[0.44, 0.72, 0.44]`

En este caso, la palabra "neuronales", menos común en el corpus, obtiene un mayor peso que "educación".

El código completo que implementa la vectorización mediante TF-IDF puede consultarse en el Apéndice C.

Word2Vec

Word2Vec genera vectores densos para cada palabra en función del contexto léxico en el que aparece, basándose en el principio de que palabras que aparecen en contextos similares tienden a tener significados similares. A diferencia de TF-IDF, no se basa en la frecuencia de aparición, sino en la coocurrencia de palabras. Captura relaciones semánticas y sintácticas, permitiendo representar palabras con significados similares mediante vectores cercanos en el espacio [40].

En este trabajo se utilizó una versión entrenada con el propio corpus de resúmenes, entrenando un modelo con el algoritmo *Skip-Gram*, que predice palabras de contexto a partir de una palabra objetivo. El código utilizado para entrenar y aplicar esta representación se incluye en el Apéndice D.

Ejemplo:

- Palabra: `"educación"`
- Vector Word2Vec: `[0.13, -0.07, 0.21, ..., 0.05]` (vector de 100 dimensiones)

Para representar un resumen completo, se promedió el vector de todas sus palabras.

Ambas representaciones se aplicaron por separado sobre las dos versiones del corpus (lema y stem), dando lugar a cuatro variantes vectorizadas que fueron posteriormente utilizadas para el entrenamiento de los modelos clasificadores.

Estas técnicas de representación semántica son ampliamente utilizadas en minería de texto y aprendizaje automático [2].

4.4. Reducción de dimensionalidad

Tras la vectorización de los resúmenes utilizando TF-IDF y Word2Vec, el espacio de características resultante era considerablemente elevado. En el caso de TF-IDF, se genera una dimensión para cada término único del corpus, lo que da lugar a vectores muy largos y dispersos (sparse), con miles de características en algunos casos. Este tipo de representación puede generar ruido, dificultar el aprendizaje y ralentizar el entrenamiento de los modelos de clasificación.

Con el objetivo de reducir la complejidad del espacio vectorial y mejorar el rendimiento de los clasificadores, se aplicaron dos técnicas de filtrado de características sobre las representaciones TF-IDF:

- **Filtrado por varianza cero**: se eliminaron todas aquellas características que presentaban exactamente el mismo valor en todos los documentos. Este tipo de términos no aporta capacidad discriminativa y puede eliminarse sin pérdida de información útil. Para ello se utilizó la clase `VarianceThreshold` de `scikit-learn` con umbral igual a cero. Esto se implementó mediante la librería `sklearn.feature_selection`, que permite filtrar características en función de su varianza.

  Ejemplo:

  - TF-IDF inicial: `["redes", "neuronal", "fútbol", "la"]` → `[0.8, 0.5, 0.0, 0.0]`
  - Tras filtrado: `["redes", "neuronal"]` → `[0.8, 0.5]`

- **Eliminación de características altamente correlacionadas**: posteriormente, se detectaron y eliminaron pares de términos cuya correlación era extremadamente alta, ya que aportaban información redundante. Esta eliminación de multicolinealidad ayuda a evitar sobreajuste y mejora la interpretabilidad de los modelos basados en árboles o distancias. Un ejemplo se muestra en la Figura 4.1, donde se observan términos con patrones de aparición casi idénticos entre documentos.

	sistema	aplicación	redes	aprendizaje
Doc1	0.65	0.64	0.40	0.30
Doc2	0.58	0.60	0.35	0.25
Doc3	0.00	0.00	0.20	0.10

Figura 4.1: Ejemplo de términos altamente correlacionados en vectores TF-IDF

Por el contrario, en el caso de la representación mediante **Word2Vec**, no fue necesario aplicar técnicas adicionales de reducción de dimensionalidad. Este modelo genera vectores densos de longitud fija (por ejemplo, 100 dimensiones), en los que cada dimensión ya representa una combinación latente aprendida

del contexto semántico de las palabras. A diferencia de TF-IDF, Word2Vec no depende del tamaño del vocabulario, y sus vectores están diseñados para capturar relaciones significativas de forma comprimida y continua.

Además, los vectores de cada resumen se obtienen como promedio de los vectores de sus palabras, lo que implica que ya están suavizados y compactados. Aplicar una reducción posterior como filtrado de varianza o correlación no tendría sentido, ya que las dimensiones no corresponden directamente a términos y su estructura interna responde a un espacio semántico aprendido.

El uso de técnicas de reducción de dimensionalidad como el filtrado por varianza y la eliminación de características redundantes está ampliamente documentado como estrategia efectiva para mejorar el rendimiento y la estabilidad de los modelos de aprendizaje automático, especialmente en contextos de alta dimensionalidad como la minería de texto [2, 3].

Capítulo 5

Evaluación de resultados

La fase de evaluación constituye un componente esencial del proyecto, ya que permite determinar qué configuraciones de modelo, representación y preprocesamiento resultan más eficaces a la hora de predecir el año de publicación de un TFG a partir de su resumen. En este capítulo se detallan las metodologías empleadas para medir el rendimiento de los clasificadores, los resultados obtenidos en distintos escenarios y las comparativas entre técnicas. Asimismo, se analiza la influencia de cada componente del sistema —modelo, vectorización y preprocesamiento—, y se identifican las configuraciones óptimas alcanzadas. Finalmente, se reflexiona sobre las limitaciones del experimento.

5.1. Metodología de evaluación

Para evaluar el rendimiento de los distintos modelos de clasificación utilizados en este trabajo, se adoptó un enfoque basado en la repetición de particiones aleatorias del conjunto de datos. En cada ejecución, los datos se dividieron en un **75 % para entrenamiento** y un **25 % para prueba**. Esta proporción se seleccionó por ser una práctica común que proporciona un buen equilibrio entre disponer de suficientes datos para entrenar modelos robustos y reservar una fracción representativa para evaluar su capacidad de generalización [3].

Con el fin de obtener resultados más estables y reducir la dependencia de una única partición, este proceso se repitió **30 veces**, cada una con una semilla distinta. Esta técnica, conocida como *repeated random sub-sampling validation*, permite estimar el rendimiento promedio del modelo y obtener una visión más fiable de su comportamiento general frente a la variabilidad de los datos. Es especialmente útil cuando el conjunto de datos no es muy grande o cuando no se aplica validación cruzada completa.

Inicialmente, el problema se abordó como una clasificación multiclase con **9 etiquetas**, una por cada año comprendido entre 2016 y 2024. No obstante, los primeros experimentos mostraron que los modelos presentaban dificultades para distinguir con precisión entre años consecutivos, dado que los resúmenes compartían muchas características léxicas comunes. Como los errores solían

implicar confusiones entre años cercanos, se optó por reagrupar los años en **tres clases** (2016–2018, 2019–2021 y 2022–2024), lo que permitió mantener la dimensión temporal del problema y, al mismo tiempo, mejorar la robustez del sistema y la calidad de la evaluación.

Las métricas utilizadas para valorar el rendimiento fueron las siguientes [41, 42].

- **Accuracy**: proporción de predicciones correctas sobre el total de ejemplos. Es útil como referencia general, pero puede verse afectada por el desequilibrio entre clases.
- **F1-score macro**: media del F1 por clase, calculada sin ponderar por la frecuencia de cada clase. Esta métrica ofrece una visión equilibrada del rendimiento en todas las clases (rangos de años), lo que es especialmente relevante cuando algunas contienen menos ejemplos que otras.
- **Precisión y recall macro**: métricas complementarias que reflejan la calidad de las predicciones positivas por clase, también promediadas sin ponderación.

El uso del **F1-score macro** como métrica principal se justifica por la naturaleza multiclase del problema y la distribución no perfectamente balanceada entre clases, a pesar de los esfuerzos por uniformizar las muestras durante la recopilación. Esta métrica permite valorar si el modelo mantiene un rendimiento homogéneo en todas las clases, sin favorecer aquellas más frecuentes [3].

5.2. Resultados por modelo

Esta sección presenta los resultados obtenidos por cada uno de los modelos de clasificación supervisada evaluados en el estudio: K-NN, Naive Bayes, Random Forest y XGBoost. Para cada modelo, se analiza el rendimiento en función de la técnica de vectorización empleada (TF-IDF o Word2Vec), el tipo de preprocesamiento aplicado (lematización o stemming) y el número de clases objetivo (9 etiquetas o 3 rangos temporales). Los resultados se expresan mediante métricas como F1 macro, precisión, exactitud y error medio, acompañadas de tablas y gráficas comparativas que permiten identificar patrones de comportamiento y configuraciones óptimas.

5.2.1. K-Nearest Neighbors (K-NN)

El modelo K-NN mostró un rendimiento modesto en la tarea de clasificación, especialmente en la versión inicial del problema con 9 etiquetas. Las métricas obtenidas reflejan una elevada sensibilidad a la representación y al preprocesamiento, con valores de F1 macro generalmente bajos.

Con 9 etiquetas, el uso de TF-IDF combinado con lematización dio lugar a un F1 medio de 0.0543, mientras que el uso de stemming mejoró ligeramente este valor hasta 0.0646. En ambos casos, el *accuracy* se mantuvo en torno al 10–12 %, apenas por encima del azar, y con errores medios de predicción superiores a 2.8 años. Con Word2Vec, el rendimiento fue algo más elevado (F1 macro de hasta

0.0908 con stemming), aunque el error medio siguió siendo elevado, superando los 3 años en la mayoría de configuraciones. Estos resultados se ilustran en la Figura 5.1, donde se comparan los valores medios de F1 obtenidos por cada configuración con 9 etiquetas.

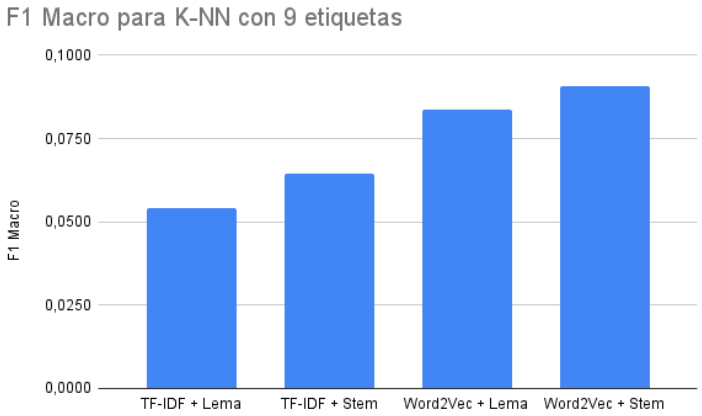

Figura 5.1: Comparación de F1 Macro para K-NN con 9 etiquetas

Al agrupar en 3 etiquetas, se observó una mejora considerable en el rendimiento del modelo. Con TF-IDF, el F1 macro aumentó hasta 0.2129 con lematización y alcanzó 0.2817 con stemming, lo que sugiere que una simplificación del espacio de clases permite a K-NN extraer patrones más generales. En el caso de Word2Vec, el rendimiento también mejoró sustancialmente, con valores de F1 de 0.3245 (lema) y 0.3395 (stem), lo que posiciona a Word2Vec como una representación más adecuada para este algoritmo. En este caso, no se calculó el error medio en años, ya que la predicción se realiza sobre rangos temporales amplios en lugar de años individuales. La Figura 5.2 muestra visualmente la comparación entre configuraciones con 3 etiquetas.

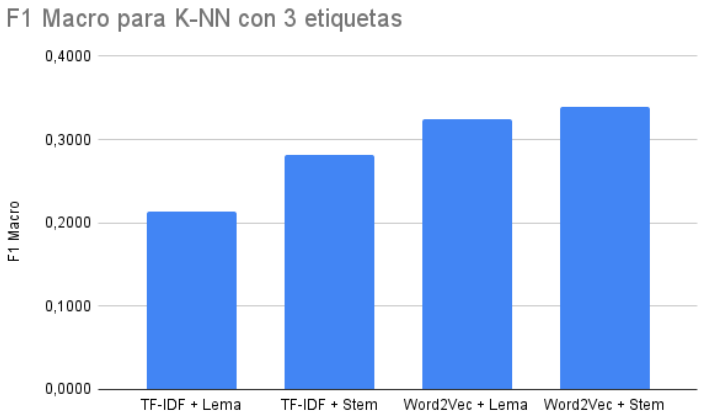

Figura 5.2: Comparación de F1 Macro para K-NN con 3 etiquetas

En la Tabla 5.1 se recogen los resultados medios del modelo K-NN para todas las combinaciones de vectorización, preprocesamiento y número de etiquetas. Esta tabla permite observar de forma comparativa cómo varía el rendimiento del modelo en función de las distintas configuraciones.

Etiquetas	Vectorización	Preprocesado	F1 Macro	Accuracy	Error medio
3	TF-IDF	Lema	0.2129	0.3411	—
3	TF-IDF	Stem	0.2817	0.3384	—
3	Word2Vec	Lema	0.3245	0.3296	—
3	Word2Vec	Stem	**0.3395**	**0.3512**	—
9	TF-IDF	Lema	0.0543	0.1170	**2.89**
9	TF-IDF	Stem	0.0646	0.1032	3.52
9	Word2Vec	Lema	0.0838	0.0899	3.16
9	Word2Vec	Stem	**0.0908**	**0.0965**	3.20

Cuadro 5.1: Resultados medios del modelo K-NN con 3 y 9 etiquetas

En general, K-NN mostró un mejor comportamiento cuando se utilizaron representaciones Word2Vec y preprocesado mediante stemming, probablemente debido a su dependencia directa de la geometría del espacio vectorial. A pesar de las mejoras observadas con 3 etiquetas, este modelo fue, en términos generales, el menos competitivo frente al resto de enfoques considerados en este trabajo.

5.2.2. Naive Bayes

Naive Bayes fue uno de los modelos con mejor comportamiento general en la tarea de clasificación, especialmente en las configuraciones que utilizaban la representación TF-IDF. Su simplicidad computacional y eficacia en entornos de texto con alta dimensionalidad le permitieron ofrecer resultados notables incluso con configuraciones mínimas.

Con 9 etiquetas, el modelo Naive Bayes mostró un rendimiento superior al de K-NN, especialmente al emplear representaciones TF-IDF. En concreto, la combinación de TF-IDF con lematización ofreció el mejor F1 macro (0.1111) y un *accuracy* del 11.75 %, superando claramente al resto de configuraciones dentro de este modelo. Aunque las diferencias con el stemming fueron pequeñas, se mantuvo una ligera ventaja para la lematización en términos de precisión global. El error medio en años osciló entre 2.58 (mejor valor con TF-IDF + stemming) y 3.15 (Word2Vec + lematización), lo que indica que, aunque el modelo acertaba menos a nivel exacto, sus errores tendían a producirse en años cercanos. Por el contrario, con Word2Vec los resultados fueron más bajos y más erráticos, con F1 macros en torno al 0.07. Esto confirma que Naive Bayes, al basarse en conteos y frecuencias, se adapta mejor a representaciones esparsas como TF-IDF que a incrustaciones densas y continuas. Estos resultados pueden visualizarse en la Figura 5.3, donde se comparan los F1 macro obtenidos en las distintas configuraciones.

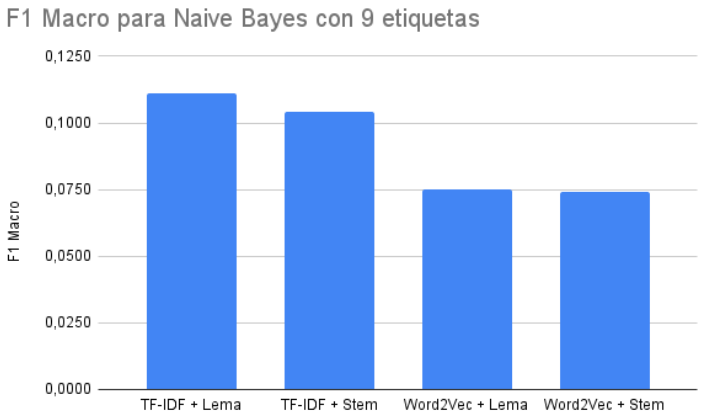

Figura 5.3: Comparación de F1 Macro para Naive Bayes con 9 etiquetas

En el escenario con 3 etiquetas, Naive Bayes experimentó una mejora clara en todas las métricas. Las configuraciones con TF-IDF volvieron a destacar, alcanzando F1 macros de 0.3659 con lematización y 0.3611 con stemming. Además, el *accuracy* se situó cerca del 37 %, lo que evidencia una mayor capacidad para capturar patrones generales entre rangos temporales. Incluso con Word2Vec, donde el modelo había mostrado un rendimiento más débil con 9 etiquetas, se logró una mejora significativa: el F1 macro alcanzó 0.3160 con stemming, lo que indica que la reducción del número de clases beneficia al modelo incluso en representaciones más complejas. Esta tendencia sugiere que Naive Bayes es especialmente competitivo en contextos simplificados, donde puede explotar con eficacia las diferencias más gruesas entre grupos de años, incluso cuando el preprocesamiento y la vectorización no son óptimos. La Figura 5.4 refleja esta mejora al mostrar comparativamente los resultados en F1 macro para cada variante.

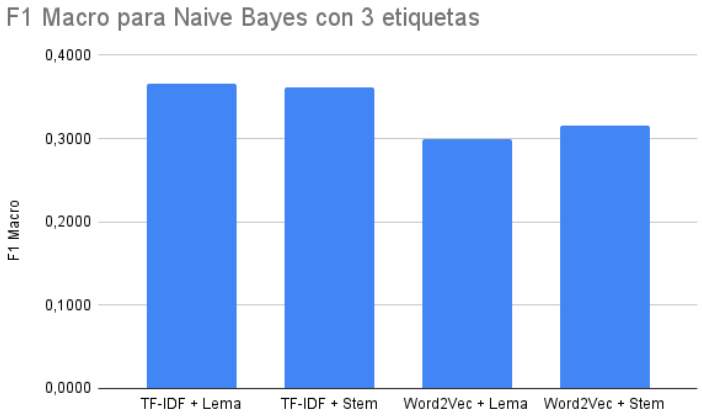

Figura 5.4: Comparación de F1 Macro para Naive Bayes con 3 etiquetas

En la Tabla 5.2 se recogen los resultados medios del modelo Naive Bayes para todas las combinaciones de representación vectorial, preprocesamiento y número de etiquetas, lo que permite observar el impacto de cada factor en el rendimiento final del modelo.

Etiquetas	Vectorización	Preprocesado	F1 Macro	Accuracy	Error medio
3	TF-IDF	Lema	**0.3659**	**0.3697**	—
3	TF-IDF	Stem	0.3611	0.3660	—
3	Word2Vec	Lema	0.2998	0.3290	—
3	Word2Vec	Stem	0.3160	0.3424	—
9	TF-IDF	Lema	**0.1111**	**0.1175**	2.69
9	TF-IDF	Stem	0.1041	0.1116	**2.58**
9	Word2Vec	Lema	0.0752	0.1054	3.15
9	Word2Vec	Stem	0.0742	0.1081	3.10

Cuadro 5.2: Resultados medios del modelo Naive Bayes con 3 y 9 etiquetas

En resumen, Naive Bayes funcionó especialmente bien con TF-IDF, destacando la versión Multinomial aplicada a datos de frecuencia. Aunque con Word2Vec no fue tan competitivo, la simplificación a 3 etiquetas permitió estabilizar los resultados. Esto refuerza la idea de que Naive Bayes sigue siendo una opción robusta para tareas de clasificación de texto supervisada, cuando se emplea con representaciones basadas en ocurrencias.

5.2.3. Random Forest

El modelo Random Forest presentó un comportamiento notablemente sólido y consistente en la mayoría de configuraciones evaluadas. Su capacidad para manejar espacios de alta dimensionalidad, así como su robustez frente al sobreajuste y al ruido, lo convirtieron en una de las opciones más competitivas del estudio.

Con 9 etiquetas, Random Forest fue el modelo más eficaz, superando de forma clara a K-NN y Naive Bayes en todas las combinaciones. En particular, la combinación de TF-IDF con lematización alcanzó un F1 macro de **0.1408** (el mayor en este escenario) y un *accuracy* del 15.05 %, mientras que con stemming se logró un F1 de 0.1303. El error medio en años se situó por debajo de 2.6 en ambos casos, lo que indica que, aunque no siempre acertaba exactamente, el modelo se equivocaba con una desviación temporal moderada. Incluso con Word2Vec, que había resultado menos competitivo en otros modelos, Random Forest obtuvo valores razonables de F1 (hasta 0.1037), mejorando el rendimiento frente a K-NN y Naive Bayes en esta representación. Esta tendencia puede observarse en la Figura 5.5, donde se comparan visualmente los valores de F1 macro para las distintas combinaciones evaluadas. En la tabla 5.3, estos valores aparecen claramente destacados para facilitar la comparación.

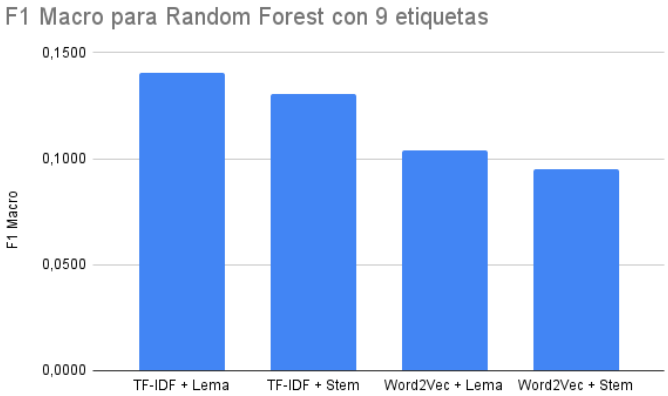

Figura 5.5: Comparación de F1 Macro para Random Forest con 9 etiquetas

Con 3 etiquetas, el rendimiento de Random Forest mejoró notablemente en todas las configuraciones. La combinación de TF-IDF con stemming fue la que alcanzó el F1 macro más alto del estudio: **0.4321**, junto con un *accuracy* del 43.40 %. Estos valores reflejan la gran capacidad del modelo para distinguir entre rangos temporales amplios. También con Word2Vec se alcanzaron resultados sólidos, con F1 de 0.3417 en el mejor caso. Esta mejora general se explica tanto por la mayor robustez de Random Forest como por la simplificación del espacio de clases, que reduce la ambigüedad entre etiquetas similares. El modelo supo aprovechar bien la riqueza de las representaciones vectoriales —especialmente TF-IDF— combinadas con un preprocesamiento agresivo como el stemming, que reduce ruido léxico sin comprometer la información esencial. La mejora puede apreciarse visualmente en la Figura 5.6, que compara el rendimiento de las distintas variantes. La tabla 5.3 muestra en negrita los mejores resultados por bloque de etiquetas, y se omite el cálculo del error medio en este escenario al no trabajar con fechas individuales, sino con rangos.

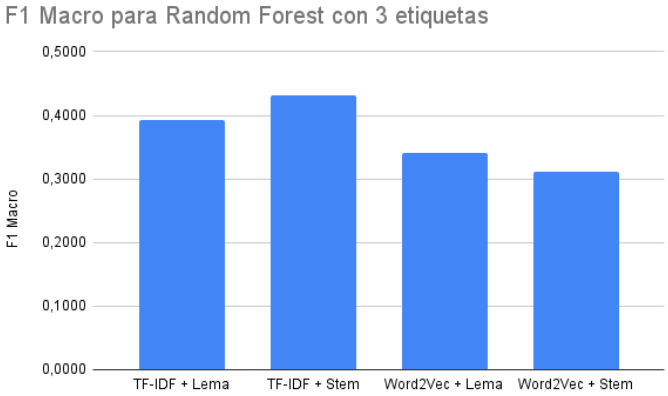

Figura 5.6: Comparación de F1 Macro para Random Forest con 3 etiquetas

En la Tabla 5.3 se resumen los resultados medios obtenidos por el modelo Random Forest para todas las combinaciones posibles de representación, preprocesamiento y número de etiquetas. Esta tabla permite comprobar de forma detallada el rendimiento diferencial del modelo en función del tipo de entrada utilizada.

Etiquetas	Vectorización	Preprocesado	F1 Macro	Accuracy	Error medio
3	TF-IDF	Lema	0.3934	0.3970	—
3	TF-IDF	Stem	**0.4321**	**0.4340**	—
3	Word2Vec	Lema	0.3417	0.3448	—
3	Word2Vec	Stem	0.3116	0.3138	—
9	TF-IDF	Lema	**0.1408**	**0.1505**	2.58
9	TF-IDF	Stem	0.1303	0.1374	**2.57**
9	Word2Vec	Lema	0.1037	0.1076	3.02
9	Word2Vec	Stem	0.0952	0.0975	2.98

Cuadro 5.3: Resultados medios del modelo Random Forest con 3 y 9 etiquetas

En conjunto, Random Forest demostró una excelente capacidad de generalización, con especial efectividad al trabajar con TF-IDF y stemming. La combinación de decisiones múltiples y la diversidad de árboles permitió al modelo captar patrones relevantes incluso en configuraciones de mayor complejidad.

5.2.4. XGBoost

XGBoost fue el modelo con mejor rendimiento global del estudio, destacando especialmente en configuraciones con representación TF-IDF. Su arquitectura basada en árboles de decisión potenciados mediante gradiente permitió capturar interacciones complejas entre variables y adaptarse mejor a las particularidades del problema.

Con 9 etiquetas, XGBoost fue el modelo con mejor rendimiento global en este escenario. En particular, con representación TF-IDF y lematización, se alcanzó un **F1 macro de 0.1527** y un **accuracy del 15.72 %**, siendo los valores más altos registrados entre todos los modelos para esta configuración. El error medio de predicción en años también fue el más bajo (2.67), lo que refuerza su capacidad para discriminar entre clases temporales estrechas. Aunque el rendimiento con Word2Vec fue más modesto (F1 entre 0.0863 y 0.1129), XGBoost mantuvo su superioridad relativa frente al resto de algoritmos. Esta tendencia puede observarse en la Figura 5.7, que resume comparativamente los resultados de F1 macro. La Tabla 5.4 recoge estos valores y destaca en negrita la mejor configuración obtenida con 9 etiquetas..

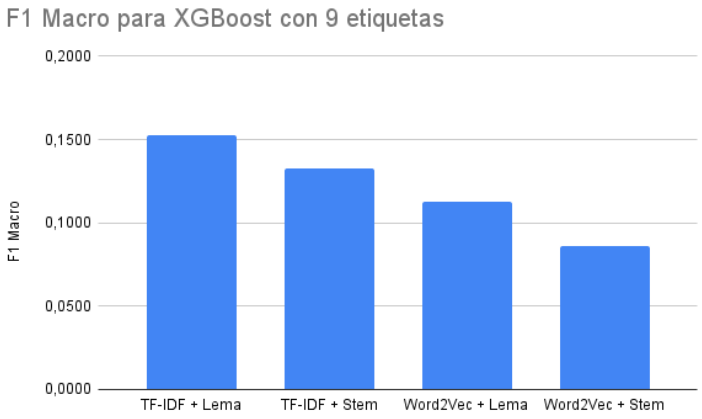

Figura 5.7: Comparación de F1 Macro para XGBoost con 9 etiquetas

Con 3 etiquetas, XGBoost volvió a destacar como uno de los modelos más eficaces, alcanzando valores de F1 macro muy cercanos a los obtenidos por Random Forest. Con TF-IDF y stemming, se logró un **F1 macro de 0.4050** y un **accuracy de 40.81 %**, marcando el segundo mejor rendimiento absoluto de todo el estudio. La simplificación del problema en clases temporales amplias permitió al modelo capitalizar su capacidad para detectar relaciones complejas entre características. Incluso con Word2Vec, XGBoost superó a otros modelos con la misma representación, confirmando su versatilidad. Los resultados se presentan en la Figura 5.8, mientras que la Tabla 5.4 resume todas las combinaciones, resaltando los mejores resultados en negrita.

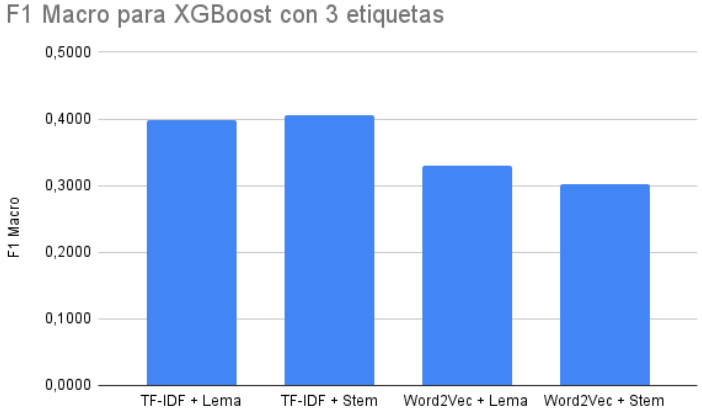

Figura 5.8: Comparación de F1 Macro para XGBoost con 3 etiquetas

En la Tabla 5.4 se resumen los resultados medios obtenidos por el modelo XGBoost en todas las combinaciones de vectorización, preprocesamiento y número de etiquetas. Esta recopilación permite comparar su rendimiento con el de otros modelos presentados previamente.

Etiquetas	Vectorización	Preprocesado	F1 Macro	Accuracy	Error medio
3	TF-IDF	Lema	0.3981	0.3997	—
3	TF-IDF	Stem	**0.4050**	**0.4081**	—
3	Word2Vec	Lema	0.3294	0.3316	—
3	Word2Vec	Stem	0.3016	0.3037	—
9	TF-IDF	Lema	**0.1527**	**0.1572**	**2.67**
9	TF-IDF	Stem	0.1325	0.1357	2.64
9	Word2Vec	Lema	0.1129	0.1157	2.95
9	Word2Vec	Stem	0.0863	0.0889	3.06

Cuadro 5.4: Resultados medios del modelo XGBoost con 3 y 9 etiquetas

En conjunto, XGBoost combinó robustez, precisión y estabilidad, mostrando una excelente capacidad para adaptarse tanto a TF-IDF como a Word2Vec. Sus resultados confirman que los modelos de boosting son especialmente adecuados para tareas de clasificación multiclase con texto, siempre que se cuente con una buena representación de características.

5.3. Influencia del preprocesamiento

El tipo de preprocesamiento léxico aplicado sobre los resúmenes tuvo un impacto apreciable en el rendimiento de los modelos. En este trabajo se compararon dos enfoques clásicos: la **lematización**, que conserva la forma base gramatical de las palabras, y el **stemming**, que recorta las palabras a una raíz común sin tener en cuenta su corrección lingüística.

En general, los resultados mostraron que **ninguna técnica fue universalmente superior**, sino que su efectividad dependía del modelo y de la representación vectorial utilizada:

- **Con TF-IDF**, el stemming produjo mejores resultados en la mayoría de los casos. Tanto Random Forest como XGBoost alcanzaron sus mayores puntuaciones de F1 macro con esta variante, lo que sugiere que una mayor reducción léxica favorece a los modelos basados en árboles cuando se utilizan representaciones esparcidas.
- **Con Word2Vec**, las diferencias fueron menos consistentes. En algunos casos, como K-NN y Naive Bayes, el stemming ofreció mejores métricas, pero en otros, como XGBoost, la lematización obtuvo valores más altos. Esto indica que la reducción agresiva propia del stemming puede perjudicar la integridad semántica de las palabras en modelos que dependen de relaciones contextuales, como Word2Vec.

Además, se observó que el stemming tiende a generar un vocabulario más reducido, lo que puede simplificar el espacio vectorial y beneficiar a clasificadores sensibles a la dimensionalidad. No obstante, esta simplificación también puede eliminar información morfológica valiosa, especialmente en modelos que extraen significado a partir del contexto semántico.

En conclusión, aunque el stemming fue generalmente más eficaz con TF-IDF —particularmente en modelos como Random Forest y XGBoost—, la lematiza-

ción mostró una mayor estabilidad y preservación semántica en configuraciones con Word2Vec. Esta comparación subraya la importancia de evaluar empíricamente cada técnica de preprocesamiento en función del modelo y del tipo de vectorización empleados.

5.4. Comparativa de representaciones vectoriales

Uno de los objetivos principales del trabajo era analizar cómo distintas representaciones semánticas del texto afectan al rendimiento de los modelos de clasificación. Para ello se compararon dos técnicas ampliamente utilizadas en procesamiento de lenguaje natural: **TF-IDF** y **Word2Vec**.

TF-IDF (Term Frequency–Inverse Document Frequency) convierte cada documento en un vector de pesos numéricos que reflejan la importancia relativa de cada palabra dentro del conjunto. Esta técnica es especialmente útil en tareas de clasificación supervisada, ya que crea vectores dispersos pero altamente diferenciables entre clases.

Word2Vec, por otro lado, genera vectores densos de dimensión fija que capturan relaciones semánticas y sintácticas entre palabras. En este trabajo se aplicó un enfoque de tipo *embedding average*, promediando los vectores de todas las palabras del resumen para obtener una representación completa del documento.

Los resultados experimentales mostraron una tendencia clara:

- **TF-IDF obtuvo sistemáticamente mejores resultados** que Word2Vec en todos los modelos, especialmente con Random Forest y XGBoost, donde las diferencias de F1 macro llegaron a superar los 10 puntos porcentuales.
- Word2Vec, aunque conceptualmente más potente para capturar contexto y significado, mostró una mayor sensibilidad a las decisiones de preprocesamiento y a la naturaleza sintética del resumen como unidad textual. Al tratarse de textos breves, la media de vectores puede diluir parte de la semántica individual.
- TF-IDF fue también más estable ante la variabilidad de muestras, y resultó más eficaz en la discriminación entre clases, especialmente al usar solo los resúmenes como fuente de información.

En resumen, **TF-IDF se posiciona como la opción más eficaz para la clasificación supervisada del año de publicación a partir del resumen de un TFG**, al menos en contextos con textos breves y bien segmentados como los analizados en este proyecto.

5.5. Configuración óptima obtenida

Tras el análisis comparativo entre modelos, técnicas de preprocesamiento y representaciones vectoriales, se identificaron dos configuraciones destacadas: una para el caso original con 9 etiquetas (años individuales) y otra para el escenario simplificado con 3 etiquetas (agrupaciones temporales).

En el escenario con 3 etiquetas, la configuración que obtuvo el mejor rendimiento fue:

- **Modelo**: Random Forest
- **Representación**: TF-IDF
- **Preprocesamiento**: Stemming
- **F1 Macro**: 0.4321
- **Accuracy**: 0.4340

Estos resultados indican que, al reducir la granularidad temporal, un enfoque clásico basado en árboles de decisión, combinado con una representación basada en frecuencia relativa y un preprocesamiento agresivo (stemming), permite una segmentación eficaz entre clases. La robustez de Random Forest frente al ruido y la alta dimensionalidad resultó especialmente ventajosa en este contexto.

En el caso con 9 etiquetas, la configuración más efectiva fue:

- **Modelo**: XGBoost
- **Representación**: TF-IDF
- **Preprocesamiento**: Lematización
- **F1 Macro**: 0.1527
- **Accuracy**: 0.1572
- **Error medio en años**: 2.67

Aunque los valores absolutos fueron más bajos que en el caso anterior, esta configuración mostró una mayor capacidad para discriminar entre años consecutivos. La arquitectura secuencial de XGBoost permitió capturar patrones más sutiles, mientras que la lematización ayudó a preservar el significado léxico, facilitando la detección de diferencias temporales.

En conjunto, los resultados reflejan que, si bien la clasificación por año individual implica una mayor complejidad y margen de error, modelos potentes como XGBoost pueden adaptarse parcialmente al reto. No obstante, la reducción del número de clases aporta mejoras claras en estabilidad y rendimiento global, como demuestra la superioridad de Random Forest en el escenario de etiquetas agrupadas.

5.6. Limitaciones y observaciones

Aunque los resultados obtenidos en este trabajo permiten extraer conclusiones sólidas sobre la eficacia de distintas configuraciones para la clasificación temporal de TFGs, es importante señalar ciertas limitaciones que condicionan el alcance y la generalización de los experimentos realizados.

En primer lugar, el **tamaño del corpus** se mantuvo acotado a una cantidad equilibrada de trabajos por año y titulación. Esta elección favoreció un diseño

experimental controlado y comparativo, pero redujo la diversidad global del conjunto, limitando así la capacidad de extrapolación a titulaciones no incluidas o a volúmenes mayores de datos.

Por otro lado, la **calidad lingüística de los resúmenes** fue variable. Algunos textos presentaban errores gramaticales, eran excesivamente breves o contenían fragmentos en otros idiomas, lo cual pudo afectar negativamente a las representaciones vectoriales y, en consecuencia, al rendimiento de los modelos.

Además, el análisis se centró exclusivamente en el *resumen* como campo de entrada. Si bien esto permitió evaluar su poder predictivo de forma aislada, la omisión de información contextual adicional —como el *título*, el *grado* o el *departamento*— puede haber limitado el potencial informativo del sistema.

En lo relativo a las configuraciones con Word2Vec, se utilizó un enfoque simple basado en el promedio de vectores, sin aplicar estrategias más sofisticadas como la ponderación por frecuencia, el uso de embeddings preentrenados de mayor calidad o arquitecturas basadas en atención. Esto pudo reducir la capacidad del modelo para captar matices semánticos importantes entre resúmenes cercanos en el tiempo.

Estas observaciones no invalidan los resultados, pero permiten contextualizarlos dentro de un marco experimental simplificado y controlado. Asimismo, constituyen una base sólida para futuras líneas de mejora y ampliación del sistema.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Resumen de resultados

Este Trabajo de Fin de Grado ha tenido como objetivo principal desarrollar un sistema capaz de inferir el año de publicación de un Trabajo de Fin de Grado (TFG) a partir de su resumen textual, utilizando técnicas de procesamiento del lenguaje natural y clasificación supervisada. Se construyó un corpus equilibrado de TFGs pertenecientes a diferentes titulaciones y años, y se evaluaron múltiples combinaciones de modelos, representaciones vectoriales y técnicas de preprocesamiento.

Los resultados experimentales demostraron que la agrupación temporal en tres etiquetas mejora significativamente el rendimiento predictivo respecto a la clasificación por año individual. En particular, la combinación de **Random Forest** con **TF-IDF** y *stemming* obtuvo un **F1 macro** de 0.4321, mientras que, para la clasificación con nueve etiquetas, el mejor rendimiento se alcanzó con **XGBoost**, **TF-IDF** y *lematización* (**F1 macro** de 0.1527). Estas diferencias reflejan la dificultad inherente a discriminar entre años concretos a partir de resúmenes breves, así como la efectividad de las representaciones basadas en frecuencia para tareas de este tipo.

6.2. Reflexión personal

Desde un punto de vista personal, este proyecto ha supuesto una experiencia altamente enriquecedora. No solo ha permitido consolidar conocimientos adquiridos a lo largo del grado, sino que también ha fomentado la autonomía, la capacidad de análisis crítico y la toma de decisiones en contextos reales de desarrollo. Asimismo, ha ofrecido la oportunidad de trabajar con herramientas de uso profesional como `MongoDB`, `spaCy`, `NLTK`, `scikit-learn` o `XGBoost`, lo que aporta una base sólida para afrontar futuros retos en el ámbito del análisis de datos y la inteligencia artificial.

6.3. Líneas de trabajo futuro

Este proyecto abre la puerta a varias líneas de trabajo futuras:

- Incluir más campos textuales (como el título o el departamento) para enriquecer la representación semántica de los TFGs y mejorar la precisión del modelo.
- Ampliar el conjunto de datos a más titulaciones y universidades, aumentando así la capacidad de generalización y explorando patrones comunes entre distintas disciplinas.
- Explorar enfoques no supervisados, como algoritmos de *clustering*, que permitirían descubrir agrupaciones naturales en los resúmenes sin necesidad de etiquetas previas. Esto podría ser útil para detectar tendencias temáticas emergentes o para análisis exploratorios de corpus académicos.
- Incorporar mecanismos de explicabilidad en los modelos utilizados, como LIME o explicaciones contrafactuales, que ayuden a interpretar las predicciones y aumentar la confianza en los resultados en contextos sensibles.

En definitiva, el proyecto demuestra la viabilidad de aplicar técnicas de procesamiento del lenguaje natural a tareas de clasificación temporal, y sienta una base sólida para futuras mejoras tanto en precisión como en aplicabilidad real.

6.4. Análisis de impacto

A lo largo de esta sección se analiza el impacto potencial del trabajo desarrollado en distintos ámbitos, tanto desde una perspectiva individual como colectiva. El objetivo es reflexionar no solo sobre los beneficios esperados del sistema propuesto, sino también sobre sus posibles limitaciones o efectos secundarios. Este ejercicio permite contextualizar el alcance del proyecto dentro del entorno académico, tecnológico y social actual.

Asimismo, se valorará en qué medida este trabajo contribuye o puede contribuir al cumplimiento de algunos de los Objetivos de Desarrollo Sostenible (ODS) establecidos en la Agenda 2030, y se señalarán las decisiones tomadas durante el desarrollo del TFG que han estado motivadas por consideraciones de impacto.

Impacto personal

Este Trabajo de Fin de Grado ha supuesto un impacto significativo a nivel personal y académico. Ha consolidado conocimientos teóricos adquiridos a lo largo del grado en áreas como minería de texto, aprendizaje automático y procesamiento del lenguaje natural, permitiendo su aplicación práctica en un caso real. La necesidad de tomar decisiones de diseño, comparar enfoques experimentales y justificar los resultados ha fomentado el pensamiento crítico y la autonomía investigadora. Además, el proyecto ha fortalecido habilidades técnicas como el uso de librerías especializadas en Python, así como competencias en redacción científica y comunicación de resultados.

Impacto académico y social

El sistema propuesto impacta positivamente tanto en el ámbito académico como en el social, al facilitar la organización cronológica y la accesibilidad de los Trabajos de Fin de Grado alojados en repositorios institucionales. Desde el punto de vista académico, permite estructurar mejor la producción estudiantil, apoyando tanto a estudiantes que buscan referencias como a docentes e investigadores que analizan la evolución de temáticas. A nivel social, promueve el acceso abierto y equitativo al conocimiento generado en la universidad, beneficiando especialmente a instituciones con recursos limitados al ofrecer una solución tecnológica eficiente, automatizada y de bajo coste para la gestión documental.

Impacto medioambiental

El proyecto no aborda directamente cuestiones medioambientales, pero sí adopta prácticas responsables. Al optar por algoritmos eficientes y evitar el uso de modelos de aprendizaje profundo, se reduce significativamente el consumo energético. Esta decisión contribuye a una inteligencia artificial más sostenible, coherente con la creciente preocupación por la huella de carbono de los sistemas computacionales.

Impacto cultural

El sistema propuesto también tiene valor cultural dentro del entorno universitario, al poner en valor la producción académica de los estudiantes y facilitar su acceso. Esto fomenta una cultura de respeto por el conocimiento generado durante la etapa formativa y refuerza el archivo académico como recurso vivo y útil para la comunidad universitaria.

Contribución a los ODS y decisiones motivadas por el impacto

Este TFG contribuye a los siguientes Objetivos de Desarrollo Sostenible (ODS):

- **ODS 4: Educación de calidad**. Mejora el acceso y la reutilización de trabajos académicos.
- **ODS 9: Industria, innovación e infraestructura**. Aplica IA al servicio de la infraestructura educativa digital.
- **ODS 12: Producción y consumo responsables**. Utiliza modelos eficientes y datos existentes, sin entrenamiento adicional costoso.

Decisiones motivadas por impacto:

- Descartar aprendizaje profundo por su alto coste computacional.
- Seleccionar datos de forma controlada para asegurar representatividad y equilibrio.

Estas decisiones reflejan un compromiso con la sostenibilidad, la eficiencia tecnológica y el respeto por los principios de equidad y responsabilidad social.

Bibliografía

[1] D. Sarkar, *Text Analytics with Python: A Practical Real-World Approach to Gaining Actionable Insights from Your Data*. Apress, 2016.

[2] C. C. Aggarwal and C. Zhai, Eds., *Mining Text Data*. Springer, 2012.

[3] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[4] Q. Chen, Y. Ma, Y. Li, and W. Li, "Fine-grained document dating with attention-based deep neural networks," *arXiv preprint arXiv:2101.01235*, 2021. [Online]. Available: https://arxiv.org/abs/2101.01235

[5] J. Silge and D. Robinson, *Text Mining with R: A Tidy Approach*. O'Reilly Media, 2017.

[6] J. R. García-González, P. A. Sánchez-Sánchez, M. Orozco, and S. Obredor, "Extracción de conocimiento para la predicción y análisis de los resultados de la prueba de calidad de la educación superior en colombia," *Formación Universitaria*, vol. 12, no. 4, 2019. [Online]. Available: https://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0718-50062019000400055

[7] E. Cambria and B. White, "Big social data: Challenges and opportunities," *IEEE Computer*, 2014. [Online]. Available: https://arxiv.org/abs/1403.4008

[8] N. Vázquez. (2021) Datos estructurados. [Online]. Available: https://www.nestorvazquez.com/datos-estructurados/

[9] R. Feldman and J. Sanger, *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, 2007. [Online]. Available: https://www.cambridge.org/core/books/abs/text-mining-handbook/CBAB2D238E342B63E7DB8F7A73B86CE2

[10] B. Z. Y. A. B. M. S. Leilani H. Gilpin, David Bau and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," *Harvard Journal of Law & Technology*, 2019. [Online]. Available: http://arxiv.org/abs/1806.00069v3

[11] A. Saraiva Leão *et al.*, "Evaluating local interpretable model-agnostic explanations on clinical machine learning classification models," in *33rd IEEE In-*

ternational Symposium on Computer-Based Medical Systems (CBMS), 2020. [Online]. Available: https://www.researchgate.net/publication/344063540

[12] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, 2009. [Online]. Available: https://doi.org/10.1109/TKDE.2008.239

[13] Z. C. Lipton, "The mythos of model interpretability," *arXiv preprint arXiv:1606.03490*, 2016. [Online]. Available: https://arxiv.org/abs/1606.03490

[14] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009. [Online]. Available: https://web.stanford.edu/~hastie/ElemStatLearn/

[15] P.-N. Tan, M. Steinbach, A. Karpatne, and V. Kumar, *Introduction to Data Mining*. Pearson, 2018.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[17] A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, 1998. [Online]. Available: https://www.cs.cmu.edu/~knigam/papers/multinomial-aaaiws98.pdf

[18] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *European Conference on Machine Learning*. Springer, 1998.

[19] J. C. N. C. (2023) Support vector machine em python. [Online]. Available: https://joaoclaudionc.medium.com/support-vector-machine-em-python-119a7da1b473

[20] L. Breiman, "Random forests," *Machine learning*, 2001. [Online]. Available: https://link.springer.com/article/10.1023/A:1010933404324

[21] PredikData. (2020) Modelos predictivos: Tipos, beneficios y ejemplos (guía completa). [Online]. Available: https://predikdata.com/es/que-son-y-para-que-se-usan-los-modelos-predictivos/

[22] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794. [Online]. Available: https://arxiv.org/abs/1603.02754

[23] Shiksha. (2023) Xgboost algorithm in machine learning. [Online]. Available: https://www.shiksha.com/online-courses/articles/xgboost-algorithm-in-machine-learning/

[24] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, 1967. [Online]. Available: https://ieeexplore.ieee.org/document/1053964

[25] A. E. E. Wellington Kanyongo, "Feature selection and importance of predictors of non-communicable diseases medication adherence from machine learning," *Data Science and Applications*, 2023. [Online]. Available: https://doi.org/10.1016/j.dsa.2023.100109

[26] F. de Jong, H. Rode, and D. Hiemstra, "Temporal language models for the disclosure of historical text," in *Proceedings of the 1st International Conference on Language Resources and Evaluation (LREC)*, 2005. [Online]. Available: https://www.cl.utwente.nl/~hiemstra/papers/lrec05.pdf

[27] N. Kanhabua and W. Nejdl, "Learning to identify high-impact time intervals in time-aware search," in *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, 2011, pp. 535–544. [Online]. Available: https://doi.org/10.1145/2009916.2009988

[28] R. Bamler and S. Mandt, "Dynamic word embeddings," in *International Conference on Machine Learning (ICML)*, 2017. [Online]. Available: https://arxiv.org/abs/1702.08329

[29] "Requests: Http for humans," versión 2.31.0. [Online]. Available: https://docs.python-requests.org

[30] "Beautifulsoup documentation," versión 4.12.2. [Online]. Available: https://www.crummy.com/software/BeautifulSoup

[31] "Pymongo documentation," versión 4.6.1. [Online]. Available: https://pymongo.readthedocs.io

[32] "Natural language toolkit (nltk)," versión 3.8.1. [Online]. Available: https://www.nltk.org

[33] "spacy: Industrial-strength nlp," versión 3.7.2. [Online]. Available: https://spacy.io

[34] "scikit-learn: Machine learning in python," versión 1.4.2. [Online]. Available: https://scikit-learn.org

[35] "Gensim: Topic modelling for humans," versión 4.3.2. [Online]. Available: https://radimrehurek.com/gensim

[36] "Xgboost documentation," versión 2.0.3. [Online]. Available: https://xgboost.readthedocs.io

[37] "Pandas: Python data analysis library," versión 2.2.2. [Online]. Available: https://pandas.pydata.org

[38] "Numpy," versión 1.26.4. [Online]. Available: https://numpy.org

[39] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. Pearson, 2023, draft version available at https://web.stanford.edu/~jurafsky/slp3/.

[40] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[41] Scikit-learn Developers, "Model evaluation: quantifying the quality of predictions," https://scikit-learn.org/stable/modules/model_evaluation.html, 2024.

[42] T. Sing, O. Sander, N. Beerenwinkel, and T. Lengauer, "Rocr: Visualizing the performance of scoring classifiers," https://cran.rstudio.com/web/packages/ROCR/vignettes/ROCR.html, 2005.

Anexos

Apéndice A

Código fuente del scraper

```
import requests
from bs4 import BeautifulSoup
from pymongo import MongoClient

def limpiar_resumen(texto):
    texto = texto.lstrip()
    if texto.lower().startswith('resumen'):
        texto = texto[len('resumen'):].lstrip()
    return texto

# Conexión a MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['upm_tfgs']  # Nombre de la base de datos
coleccion = db['tfgs']   # Nombre de la colección

# Lista de grados
grados = {
    "Ciencias de la Actividad Física y del Deporte": 'https://oa
       .upm.es/view/degree/Grado_en_Ciencias_de_la_Actividad_F=
       EDsica_y_del_Deporte.html',
    "Fundamentos de la Arquitectura": 'https://oa.upm.es/view/
       degree/Grado_en_Fundamentos_de_la_Arquitectura.html',
    "Ingeniería en Tecnologías Industriales": 'https://oa.upm.es
       /view/degree/Grado_en_Ingenier=EDa_en_Tecnolog=
       EDas_Industriales.html',
    "Ingeniería Informática": 'https://oa.upm.es/view/degree/
       Grado_en_Ingenier=EDa_Inform=E1tica.html'
}

for grado_nombre, url in grados.items():
    print(f"\n========== Grado: {grado_nombre} ==========")
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
```

```
    for year in range(2024, 2014, -1):
        print(f"\n--- Ano {year} ---")
        anchor = soup.find('a', attrs={'name': f'group_{year}'})
        h2_tag = anchor.find_next('h2') if anchor else None

        if h2_tag:
            tfg_links = []
            next_tag = h2_tag.find_next('a', href=True)
            while next_tag and len(tfg_links) < 10:
                href = next_tag['href']
                if href.startswith('https://oa.upm.es/') and
                   href.strip('https://oa.upm.es/').rstrip('/').
                   isdigit():
                    tfg_links.append(href)
                next_tag = next_tag.find_next('a', href=True)

            for tfg_url in tfg_links:
                tfg_response = requests.get(tfg_url)
                tfg_soup = BeautifulSoup(tfg_response.content, '
                   html.parser')

                # Título
                title_tag = tfg_soup.find('h1', class_='
                   ep_tm_pagetitle')
                titulo = title_tag.text.strip() if title_tag
                   else 'No encontrado'

                # Metadatos
                departamento = escuela = resumen = 'No
                   encontrado'
                metadata_rows = tfg_soup.find_all('tr')
                for row in metadata_rows:
                    header = row.find('th')
                    data = row.find('td')
                    if header and data:
                        label = header.text.strip()
                        value = data.text.strip()
                        if 'Departamento' in label:
                            departamento = value
                        elif 'Escuela' in label or 'Centro' in
                           label:
                            escuela = value
                        elif 'Resumen' in label:
                            resumen = limpiar_resumen(value)
```

```
                # Si no encuentra el resumen en la tabla, busca
                    en el div.abstract
                if resumen == 'No encontrado':
                    abstract_div = tfg_soup.find('div', class_='
                        abstract')
                    if abstract_div:
                        resumen = limpiar_resumen(abstract_div.
                            get_text(strip=True))

                # Insertar en MongoDB
                coleccion.insert_one({
                    'Título': titulo,
                    'Grado': grado_nombre,
                    'Ano': year,
                    'Escuela': escuela,
                    'Departamento': departamento,
                    'Resumen': resumen,
                    'URL': tfg_url
                })
```

Apéndice B

Código preprocesamiento

```
import re
import spacy
import nltk
from pymongo import MongoClient
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from sklearn.feature_extraction import text

# Descargar recursos NLTK
nltk.download('punkt')
nltk.download('stopwords')

# Cargar modelo spaCy
nlp = spacy.load("es_core_news_sm")

# Cargar stopwords combinadas (spaCy + NLTK + inglés sklearn)
stopwords_spacy = nlp.Defaults.stop_words
stopwords_nltk = set(stopwords.words("spanish"))
stopwords_english = text.ENGLISH_STOP_WORDS
stopwords_combinadas = stopwords_spacy.union(stopwords_nltk).
   union(stopwords_english)

# Preprocesamiento con lematización (spaCy)
def lematizar(texto):
    texto = texto.lower()
    texto = re.sub(r'[^a-záéíóúnü\s]', '', texto)
    doc = nlp(texto)
    palabras = [
        token.lemma_ for token in doc
        if token.lemma_ not in stopwords_combinadas and token.
           is_alpha and len(token) > 2
    ]
    return ' '.join(palabras)
```

```
# Preprocesamiento con stemming (SnowballStemmer)
stemmer = SnowballStemmer("spanish")

def stemmear(texto):
    texto = texto.lower()
    texto = re.sub(r'[^a-záéíóúnü\s]', '', texto)
    palabras = nltk.word_tokenize(texto, language='spanish')
    palabras = [
        stemmer.stem(p) for p in palabras
        if p not in stopwords_combinadas and len(p) > 2 and p.
           isalpha()
    ]
    return ' '.join(palabras)

# Conexión a MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['upm_tfgs']  # Nombre de la base de datos
coleccion = db['tfgs']   # Nombre de la colección

# Procesar todos los documentos con resumen
total = coleccion.count_documents({"Resumen": {"$exists": True
   }})
print(f"Procesando {total} documentos...")

for doc in coleccion.find({"Resumen": {"$exists": True}}):
    resumen = doc["Resumen"]

    resumen_lematizado = lematizar(resumen)
    resumen_stemmed = stemmear(resumen)

    coleccion.update_one(
        {"_id": doc["_id"]},
        {"$set": {
            "resumen_lematizado": resumen_lematizado,
            "resumen_stemming": resumen_stemmed
        }}
    )
```

Apéndice C

Código TF-IDF

```
from pymongo import MongoClient
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import VarianceThreshold
import pandas as pd
import numpy as np

# Conexión a MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['upm_tfgs']
coleccion = db['tfgs']

# Cargar documentos
documentos = list(coleccion.find({
    "resumen_lematizado": {"$exists": True},
    "resumen_stemming": {"$exists": True},
    "Ano": {"$exists": True}
}))
print(f"Se han cargado {len(documentos)} documentos.")

# Extraer textos y etiquetas
textos_lema = [doc["resumen_lematizado"] for doc in documentos]
textos_stem = [doc["resumen_stemming"] for doc in documentos]
y = [doc["Ano"] for doc in documentos]

# Vectorización TF-IDF
vectorizer_lema = TfidfVectorizer(max_features=1000)
vectorizer_stem = TfidfVectorizer(max_features=1000)

x_lema = vectorizer_lema.fit_transform(textos_lema)
x_stem = vectorizer_stem.fit_transform(textos_stem)

vocab_lema = vectorizer_lema.get_feature_names_out()
vocab_stem = vectorizer_stem.get_feature_names_out()
```

```
# Eliminar baja varianza
var_selector = VarianceThreshold(threshold=0.001)

x_lema_var = var_selector.fit_transform(x_lema)
vocab_lema_var = vocab_lema[var_selector.get_support()]

x_stem_var = var_selector.fit_transform(x_stem)
vocab_stem_var = vocab_stem[var_selector.get_support()]

# Eliminar alta correlación
def quitar_columnas_correladas(x, nombres, umbral=0.7):
    df = pd.DataFrame(x.toarray(), columns=nombres)
    corr_matrix = df.corr().abs()
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape)
       , k=1).astype(bool))
    columnas_a_quitar = [col for col in upper.columns if any(
       upper[col] > umbral)]
    df_filtrado = df.drop(columns=columnas_a_quitar)
    return df_filtrado, columnas_a_quitar

df_lema_final, cols_quitadas_lema = quitar_columnas_correladas(
   x_lema_var, vocab_lema_var)
df_stem_final, cols_quitadas_stem = quitar_columnas_correladas(
   x_stem_var, vocab_stem_var)

# Anadir la columna de etiquetas
df_lema_final["Ano"] = y
df_stem_final["Ano"] = y
```

Apéndice D

Código Word2Vec

```
from pymongo import MongoClient
from gensim.models import Word2Vec
import nltk
import numpy as np

# Descargar recursos
nltk.download('punkt')

# Conexión a MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['upm_tfgs']
coleccion = db['tfgs']

# Cargar documentos con ambos resúmenes y ano
documentos = list(coleccion.find({
    "resumen_lematizado": {"$exists": True},
    "resumen_stemming": {"$exists": True},
    "Ano": {"$exists": True}
}))

# Tokenización
corpus_lema = [nltk.word_tokenize(doc["resumen_lematizado"]) for
    doc in documentos]
corpus_stem = [nltk.word_tokenize(doc["resumen_stemming"]) for
   doc in documentos]
y = [doc["Ano"] for doc in documentos]

# Entrenar modelos Word2Vec
model_lema = Word2Vec(sentences=corpus_lema, vector_size=100,
   window=5, min_count=2, workers=4, seed=42)
model_stem = Word2Vec(sentences=corpus_stem, vector_size=100,
   window=5, min_count=2, workers=4, seed=42)
```

```
# Vector promedio
def vector_promedio(palabras, modelo):
    vectores = [modelo.wv[p] for p in palabras if p in modelo.wv
        ]
    return np.mean(vectores, axis=0) if vectores else np.zeros(
        modelo.vector_size)

x_w2v_lema = np.array([vector_promedio(palabras, model_lema) for
    palabras in corpus_lema])
x_w2v_stem = np.array([vector_promedio(palabras, model_stem) for
    palabras in corpus_stem])
```

Apéndice E

Código Modelos

```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, accuracy_score,
   precision_score, recall_score
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.preprocessing import LabelEncoder

# Cargar vectores desde archivos .npz
tfidf = np.load("vectores_tfidf.npz")
word2vec = np.load("vectores_word2vec.npz")

x_lema_tfidf = tfidf["lema"]
x_stem_tfidf = tfidf["stem"]
y_raw = tfidf["y"]  # <- versión original

x_lema_word2vec = word2vec["lema"]
x_stem_word2vec = word2vec["stem"]

# Transformar etiquetas (anos reales) a clases [0, 1, ..., 8]
# from sklearn.preprocessing import LabelEncoder
# le = LabelEncoder()
# y = le.fit_transform(y_raw)

# Agrupar anos en 3 clases: 0 = antiguos, 1 = medios, 2 =
   recientes
def agrupar_anios(y):
    return np.select(
        [y <= 2018, (y >= 2019) & (y <= 2021), y >= 2022],
        [0, 1, 2]
```

```
    )

y = agrupar_anios(y_raw.astype(int))

# Evaluación repetida aleatoria mejorada
def evaluar_repetidamente(x, y, modelo, repeticiones=30,
   test_size=0.25):
    f1_scores = []
    accuracies = []
    precisions = []
    recalls = []
    errores_en_anos = []

    for _ in range(repeticiones):
        x_train, x_test, y_train, y_test = train_test_split(x, y
           , test_size=test_size, stratify=y)
        modelo.fit(x_train, y_train)
        y_pred = modelo.predict(x_test)

        f1_scores.append(f1_score(y_test, y_pred, average='macro
           '))
        accuracies.append(accuracy_score(y_test, y_pred))
        precisions.append(precision_score(y_test, y_pred,
           average='macro', zero_division=0))
        recalls.append(recall_score(y_test, y_pred, average='
           macro', zero_division=0))

        # Convertir de clases (0-8) a anos reales
        # y_test_real = le.inverse_transform(y_test)
        # y_pred_real = le.inverse_transform(y_pred)

        # Calcular error en anos y guardar la media
        # errores = np.abs(y_test_real - y_pred_real)
        # errores_en_anos.append(np.mean(errores))

    return (
        round(np.mean(f1_scores), 4),
        round(np.mean(accuracies), 4),
        round(np.mean(precisions), 4),
        round(np.mean(recalls), 4),
        # round(np.mean(errores_en_anos), 2)
    )

# Combinaciones a evaluar
combinaciones = [
    ("K-NN", "TF-IDF", "Lema", KNeighborsClassifier(n_neighbors
       =5), x_lema_tfidf),
```

```
    ("K-NN", "TF-IDF", "Stem", KNeighborsClassifier(n_neighbors
       =5), x_stem_tfidf),
    ("Naive Bayes", "TF-IDF", "Lema", MultinomialNB(),
       x_lema_tfidf),
    ("Naive Bayes", "TF-IDF", "Stem", MultinomialNB(),
       x_stem_tfidf),
    ("Random Forest", "TF-IDF", "Lema", RandomForestClassifier(
       n_estimators=100), x_lema_tfidf),
    ("Random Forest", "TF-IDF", "Stem", RandomForestClassifier(
       n_estimators=100), x_stem_tfidf),
    ("XGBoost", "TF-IDF", "Lema", XGBClassifier(eval_metric='
       mlogloss'), x_lema_tfidf),
    ("XGBoost", "TF-IDF", "Stem", XGBClassifier(eval_metric='
       mlogloss'), x_stem_tfidf),

    ("K-NN", "Word2Vec", "Lema", KNeighborsClassifier(
       n_neighbors=5), x_lema_word2vec),
    ("K-NN", "Word2Vec", "Stem", KNeighborsClassifier(
       n_neighbors=5), x_stem_word2vec),
    ("Naive Bayes", "Word2Vec", "Lema", GaussianNB(),
       x_lema_word2vec),
    ("Naive Bayes", "Word2Vec", "Stem", GaussianNB(),
       x_stem_word2vec),
    ("Random Forest", "Word2Vec", "Lema", RandomForestClassifier
       (n_estimators=100), x_lema_word2vec),
    ("Random Forest", "Word2Vec", "Stem", RandomForestClassifier
       (n_estimators=100), x_stem_word2vec),
    ("XGBoost", "Word2Vec", "Lema", XGBClassifier(eval_metric='
       mlogloss'), x_lema_word2vec),
    ("XGBoost", "Word2Vec", "Stem", XGBClassifier(eval_metric='
       mlogloss'), x_stem_word2vec),
]

# Ejecutar evaluaciones
resultados = []

for nombre, tipo_vec, preproc, modelo, x in combinaciones:
    f1_avg, acc_avg, prec_avg, recall_avg =
       evaluar_repetidamente(x, y, modelo)
    resultados.append({
        "Modelo": nombre,
        "Vectorización": tipo_vec,
        "Preprocesado": preproc,
        "F1 Macro (avg)": f1_avg,
        "Accuracy (avg)": acc_avg,
        "Precision (avg)": prec_avg,
        "Recall (avg)": recall_avg,
```

```
        # "Error medio en anos": error_anios
    })
```

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jun 04 21:49:40 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)