



**UNIVERSIDAD
DE GRANADA**

TRABAJO DE FIN DE GRADO
Grado en Ingeniería Informática

Recomendación de revistas científicas

**utilizando técnicas de aprendizaje
automático**

Autor

Juan Manuel Consigliere Picco

Tutores

Juan Francisco Huete Guadix
Luis Miguel de Campos Ibáñez



**Escuela Técnica Superior de Ingenierías Informática
y de Telecomunicación**

Granada, julio de 2022

Recomendación de revistas científicas: Utilizando técnicas de aprendizaje automático

Juan Manuel Consigliere Picco

Palabras clave: Machine Learning, Deep Learning, Python, Métrica, Clasificación textual,
Recomendación, COVID-19, Kaggle

Resumen:

El objetivo principal de este TFG es el de analizar posibles alternativas dado el siguiente problema de recomendación: partimos de un dataset el cual contiene artículos relacionados con diversas temáticas (en nuestro caso todas relacionadas con el ámbito del COVID-19) publicados en una revista por artículo, y nuestra tarea es la de recomendar un ranking de 10 revistas posibles en base al contenido del artículo.

Se explorarán distintos métodos de aprendizaje de la rama del Machine Learning y el Deep Learning, y se analizarán individualmente los resultados obtenidos para poder tomar decisiones en base al rendimiento de estos mediante el uso de diversas métricas. Se tendrá en cuenta tanto el resumen de cada artículo como el texto completo de estos, analizando si el cambio es beneficioso de cara a mejorar la calidad de la predicción.

Las pruebas serán realizadas en notebooks Jupyter, así como el preprocesado de la información con la que contamos. Los notebook Jupyter trabajan con el lenguaje Python, el cual es actualmente el lenguaje de programación más utilizado para el mundo de la Ciencia de Datos. Dentro de este contaremos con librerías como Pandas, la cual nos ayudará a realizar un preprocesado de los datos, además de otras librerías enfocadas puramente al Machine Learning y al Deep Learning, como son SciKit Learn y Tensorflow, respectivamente. Tendremos también el apoyo de otras librerías para realizar gráficas de cara a dar información más profunda acerca de nuestro problema, como es PyPlot.

Scientific journal recommendation: By using Machine Learning Techniques

Juan Manuel Consigliere Picco

Keywords: Machine Learning, Deep Learning, Python, Metric, Text Classification, Recommendation, COVID-19, Kaggle

Abstract:

The main objective of this TFG is to analyze possible alternatives given the following recommendation problem: we start from a dataset which contains articles related to different topics (in our case all related to the field of COVID-19) published in one journal per article, and our task is to recommend a ranking of 10 possible journals based on the content of the article.

Different learning methods from the Machine Learning and Deep Learning branch will be explored, and the results obtained will be analyzed individually in order to make decisions based on their performance using various metrics. Both the summary and the full text of each article will be taken into account, analysing whether the change is beneficial in terms of improving the quality of the prediction.

The tests will be carried out on Jupyter notebooks, as well as the preprocessing of the information we have. The Jupyter notebooks work with the Python language, which is currently the most widely used programming language in the world of Data Science. Within this we will have libraries such as Pandas, which will help us to preprocess the data, as well as other libraries focused purely on Machine Learning and Deep Learning, such as SciKit Learn and Tensorflow, respectively. We will also have the support of other libraries for making graphs to provide more in-depth information about our problem, such as PyPlot.

Juan Manuel Consigliere Picco

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Yo, **Juan Manuel Consigliere Picco**, alumno de la titulación INGENIERÍA INFORMÁTICA de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77148105X, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Juan Manuel Consigliere Picco

Granada a 08 de julio de 2022

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

D. **Juan Francisco Huete Guadix**, Profesor del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

D. **Luis Miguel de Campos Ibáñez**, Profesor del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado **Recomendación de revistas científicas, Utilizando técnicas de aprendizaje automático**, ha sido realizado bajo su supervisión por **Juan Manuel Consigliere Picco**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 08 de julio de 2022.

Los directores:

Juan Francisco Huete Guadix

Luis Miguel de Campos Ibáñez

Agradecimientos

Me gustaría agradecer a mis tutores de este TFG, D. Juan Francisco Huete Guadix y D. Luis Miguel de Campos Ibáñez, por toda la ayuda prestada y la gran profesionalidad, además de la paciencia y el empeño. Agradezco también a los profesores D. Juan Manuel Fernández Luna y a D. José Manuel Benítez Sánchez por otorgarme acceso al servidor bahía.ugr.es, y por estar ahí cuando he necesitado soporte respecto al uso del servidor.

Me gustaría recordar a mi familia y amigos por la ayuda moral que he recibido para seguir adelante, y por el ánimo prestado. Agradezco a mis padres, D. Pablo Adrián Consigliere Rodríguez y D.^a Andrea Stella Maris Picco, por todo el esfuerzo económico que les ha supuesto mi educación y por enseñarme el valor del esfuerzo en la vida, y en especial a mi abuelo, el cual por desgracia falleció, y me gustaría recordarlo en este momento y dedicarle este TFG en su memoria, por todo el cariño de tantos años.

Me gustaría agradecer también a mi gran amigo D. Diego Hernández Moreno por la ayuda recibida para comprender conceptos que se escapaban de mi entendimiento, y a mi pareja D.^a Marina Caparrós Alconcher, que ha estado conmigo en los momentos más duros de este TFG, y me ha ayudado a sobrellevar situaciones complicadas durante la realización de este. Gracias por todo el cariño.

Contenido

Capítulo 1 Introducción.....	9
1.1 Objetivo.....	12
1.2 Capítulos y contenido general	12
Capítulo 2 Trabajos de interés.....	14
Capítulo 3 Presupuesto.....	16
3.1 Planificación.....	16
3.2 Mano de obra.....	16
3.2 Recursos materiales.....	17
Capítulo 4 Clasificación textual	19
4.1 Extracción de características	20
4.1.1 Limpieza del dataset.....	20
4.1.2 Preprocesado de texto básico	21
4.1.3 Tokenización	21
4.1.4 Borrado de palabras vacías.....	21
4.1.5 Lematización	22
4.1.6 Vectorizado	22
4.1.7 Word Embedding (opcional).....	23
4.2 Modelos de clasificación.....	23
4.2.1 Multinomial Naive Bayes.....	24
4.2.2 K-Nearest Neighbors.....	24
4.2.3 Random Forest	26
4.2.4 Multinomial Logistic Regression	27
4.2.5 Complement Naive Bayes.....	28
4.2.6 Long-Short Term Memory	29
Capítulo 5 Dataset escogido.....	31
Capítulo 6 Procesado de CORD-19	35
6.1 Limpieza de datos	36
6.1.1 Número de clases y relevancia	36
6.1.2 Borrado de nulos	36
6.1.3 Borrado de clases irrelevantes.....	38
6.2 Deep Learning - Preprocesado	39
6.2.1 Borrado de stopwords.....	39
6.2.2 División train/test	40

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

6.2.3 Palabras relevantes (extra).....	41
6.3 Machine Learning - Preprocesado.....	42
6.3.1 Paso del abstract por el stemmer	42
6.3.2 Vectorizado Tf-Idf.....	44
Capítulo 7 Datasets y variaciones	45
7.1 Dataset unigramas y bigramas.....	45
7.2 Dataset Undersampling	46
7.3 Dataset Oversampling	47
Capítulo 8 Resultados obtenidos	48
8.1 Métricas empleadas	48
8.1.1 Tiempo de entrenamiento/predicción.....	48
8.1.2 Mean Reciprocal Rank	48
8.1.3 Accuracy Score	49
8.1.4 Top K Accuracy Score	49
8.2 Hardware utilizado	50
8.3 Resultados del dataset inicial	50
8.3.1 Multinomial Naive Bayes.....	50
8.3.2 K-Nearest Neighbours	52
8.3.3 Random Forest	55
8.3.4 Multinomial Logistic Regression	57
8.3.5 Complement Naive Bayes.....	60
8.3.6 Resumen de los métodos utilizados.....	61
8.4 Resultados del dataset unigramas/bigramas	62
8.4.1 Resumen de resultados	62
8.5 Resultados de los dataset de desbalanceo.....	63
8.5.1 Undersampling: selección del método a utilizar	63
8.5.2 Undersampling: resumen de resultados.....	64
8.5.3 Oversampling: selección del método a utilizar	65
8.5.4 Oversampling: resumen de resultados.....	66
8.6 Resultados del dataset Deep Learning.....	67
8.6.1 Construcción de vocabulario y padding	67
8.6.2 Creación de matriz de embeddings	68
8.6.3 Indexado de clases.....	68
8.6.4 Construcción del modelo.....	68
8.6.5 Compilación del modelo	71
8.6.6 Resultados y ajustes de modelo.....	71

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Capítulo 9 Dataset con texto completo	75
9.1 Transformaciones de los datos	75
9.1.1 Extracción de texto de los artículos.....	76
9.1.2 Deep Learning - Preprocesado	76
9.1.3 Machine Learning - Preprocesado.....	77
9.2 Machine Learning - Resultados.....	77
9.3 Deep Learning - Resultados	78
Capítulo 10 Conclusión.....	80
10.1 Vista general del trabajo realizado	80
10.2 Experiencia personal	82
Apéndice	83
Apéndice 1: Archivos de datos.....	83
1. Archivos troncales.....	83
2. Deep Learning - Archivos de entrenamiento	83
3. Machine Learning - Archivos de entrenamiento	83
4. Archivos de test.....	83
Apéndice 2: Código de preprocesado.....	84
1. Pipeline (/src/pipeline)	84
2. Pipeline de texto completo (/src/pipeline-fulltext).....	84
Apéndice 3: Archivos de pruebas.....	85
1. Elección del método de desbalanceo (/src/elección)	85
2. Machine Learning - Archivos de pruebas (/src).....	85
3. Deep Learning - Embedding (/src/lstm/embedding)	85
4. Deep Learning - Archivos de pruebas (/src/lstm)	85
Ilustraciones	86
Bibliografía	87

Capítulo 1

Introducción

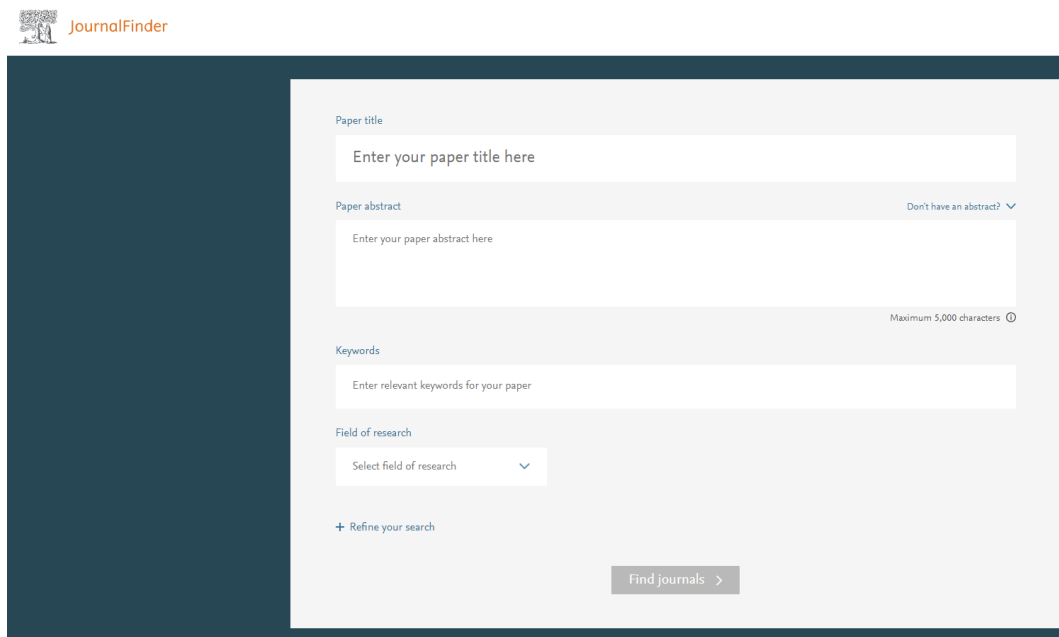
Como lleva siendo desde hace tiempo, la ciencia de datos es cada vez algo más grande. Desde su surgimiento en el año 1962 gracias al estadístico estadounidense John W. Turkey [1] ha estado en constante evolución, y es acompañada de numerosos proyectos que poco a poco nos acercan más a la “perfección”.

Uno de los problemas abordados por la ciencia de datos es el problema de recomendación, cuyo objetivo es el de recomendar a un usuario final un conjunto de productos que se ajuste a sus intereses. Los sistemas de recomendación están muy presentes en el ecosistema empresarial. Plataformas como YouTube, Netflix, Amazon, entre otras muchas, llevan utilizando estos sistemas desde hace mucho: partiendo de información de usuario como sus preferencias, clicks y tiempos de visualización, se genera una recomendación de K ítems, (libros, películas, vídeos, canciones, etc) en estas webs. La correcta recomendación es un campo de interés para la empresa, ya que se asocia con mayores beneficios tanto económicos como para el cliente.

El marco en el que se sitúa el proyecto es el siguiente: el usuario final será un investigador, y buscamos la recomendación de una serie de productos, que en este caso son revistas científicas. El auge de las revistas científicas en las últimas décadas ha desencadenado una verdadera necesidad a la hora de escoger el medio por el que transmitir nuestros proyectos. Ofrecer recomendaciones en base al contenido de la información que el investigador provee alivia esta necesidad. Algunas editoriales como [Springer](#) o [Elsevier](#), pretenden solucionar este problema y satisfacer esta necesidad tan demandada.

Para un investigador es muy importante encontrar la revista correcta a la que someter el trabajo. Esto no es tarea fácil, ya que hay miles de revistas científicas distintas que tratan temáticas muy dispares. El grado de aceptación del trabajo enviado a una revista depende en gran medida de su contenido y de las temáticas de interés de la propia revista (scope). No seleccionar la revista correcta da lugar a fenómenos como retrasos en la publicación, rechazo del trabajo, e incluso en caso de ser aceptado para su publicación, puede dificultarse la difusión del mismo debido a que el público potencial de la revista es el incorrecto.

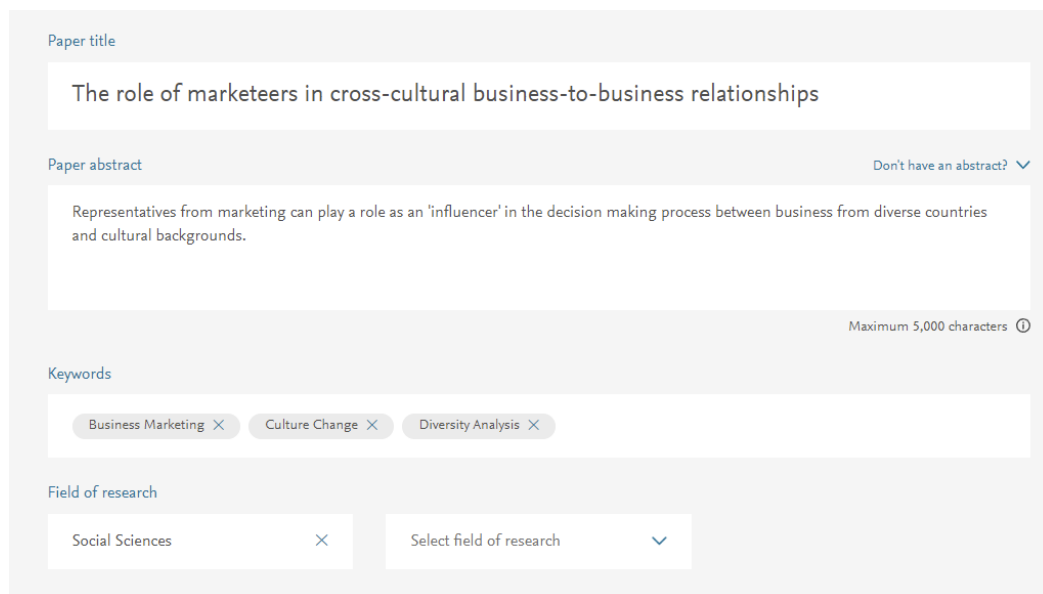
Recomendación de revistas científicas utilizando técnicas de aprendizaje automático



The screenshot shows the Elsevier JournalFinder web interface. It features a dark blue sidebar on the left and a main white content area. The content area contains several input fields: 'Paper title' with a placeholder 'Enter your paper title here', 'Paper abstract' with a placeholder 'Enter your paper abstract here' and a 'Don't have an abstract?' link, 'Keywords' with a placeholder 'Enter relevant keywords for your paper', and 'Field of research' with a dropdown menu labeled 'Select field of research'. There is also a '+ Refine your search' link and a 'Find journals >' button at the bottom right.

Ilustración 1 Sistema de Recomendación Elsevier JournalFinder

Veamos el sistema de recomendación [Elsevier JournalFinder](#) (*ilustración 1*): tiene como datos de entrada el título del artículo, su resumen, su campo de investigación y palabras clave, aunque estos dos últimos son opcionales. Tomamos como ejemplo el siguiente artículo suministrado por la propia herramienta (*ilustración 2*):



The screenshot shows the Elsevier JournalFinder web interface with example data filled in. The 'Paper title' field contains 'The role of marketers in cross-cultural business-to-business relationships'. The 'Paper abstract' field contains 'Representatives from marketing can play a role as an 'influencer' in the decision making process between business from diverse countries and cultural backgrounds.'. The 'Keywords' field contains three tags: 'Business Marketing', 'Culture Change', and 'Diversity Analysis'. The 'Field of research' field contains a tag 'Social Sciences' and a dropdown menu labeled 'Select field of research'.

Ilustración 2 Campos rellenos Elsevier JournalFinder

Al aceptar y buscar nos muestra un total de 34 revistas cuyos temas se ajustan al trabajo especificado (“text match score”), presentadas de mayor a menor grado de emparejamiento.

Recomendación de revistas científicas utilizando técnicas de aprendizaje automático

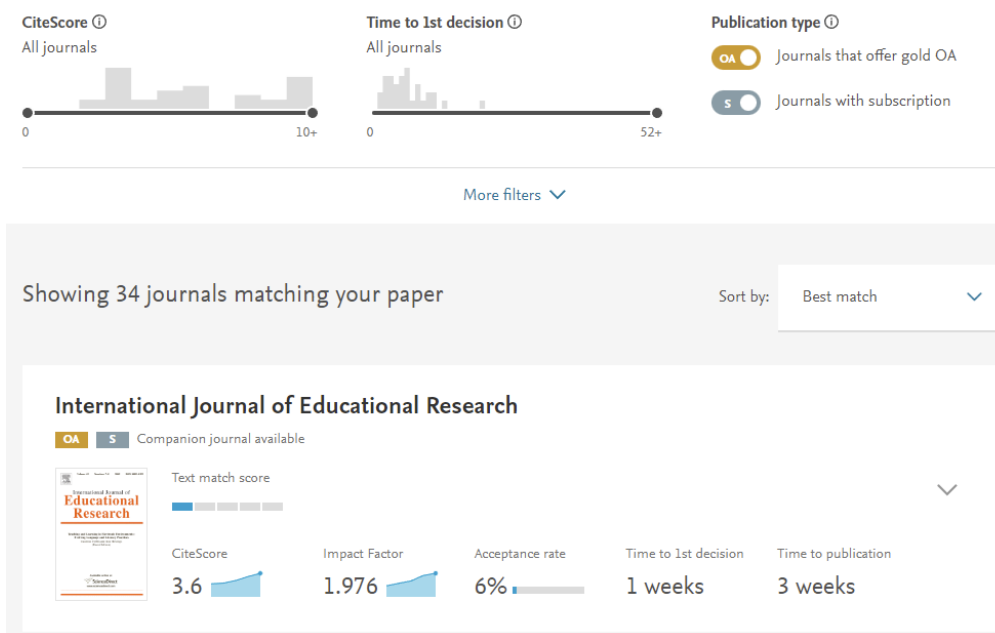


Ilustración 3 Resultados de la búsqueda Elsevier JournalFinder

Vemos también cómo pueden ajustarse algunos filtros para refinar aún más la búsqueda (*ilustración 3*). También se puede observar que para cada revista se muestra información de interés como el porcentaje de aceptación de artículos, tiempo que tarda en publicarlo, entre otros. En nuestro caso trabajaremos con una entrada textual de varios campos, y una salida en forma de ranking de recomendación, al igual que en el Sistema de Recomendación anterior.

Los sistemas de recomendación se pueden dividir en cuatro principales grupos [2]:

- **Sistemas de popularidad:** La variable principal por la que se realiza la recomendación es el número de ventas, características especiales, ofertas, etc. Todas las variables son aquellas que denotan cierta popularidad en un ítem.
- **Sistemas basados en contenido:** Se toman datos del usuario y muestra sugerencias que se ajusten a estos datos. Son los más presentes en la actualidad, y son altamente configurables en base al problema.
- **Sistemas de filtrado colaborativo:** Analiza perfiles del usuario, recolecta información y agrupa usuarios por grupos de interés (perfiles similares). Esto junto con los datos de usuario genera recomendaciones individuales.
- **Sistemas híbridos:** Se le pide al usuario indicar datos previos y, junto con información sobre su perfil y preferencias, se generan las recomendaciones.

La Clasificación Textual es una técnica que se encarga de, mediante un entrenamiento previo, clasificar texto dentro de distintas categorías, y entra dentro del NLP, o Natural Language

Processing. El problema de recomendación puede ser abordado como un problema de Clasificación Textual, y su aprendizaje puede ser realizado mediante técnicas de Machine Learning o mediante Deep Learning. Machine Learning es un subcampo de la Inteligencia Artificial, y busca mediante algoritmos y datos la capacidad de una máquina para imitar el comportamiento inteligente humano. Deep Learning es un subconjunto de técnicas de Machine Learning que utiliza modelos de redes neuronales avanzadas, e intenta imitar la forma en la que los humanos adquieren conocimiento.

Como podemos observar es un problema que se ve día a día en la web y, a pesar de poder ser aplicado como un problema de recuperación de información (disciplina que se encarga de trabajar con información textual), vamos a trabajar por el lado de la ciencia de datos y la inteligencia artificial para lograr nuestro objetivo.

1.1 Objetivo

En nuestro problema contamos con una serie de revistas distintas, las cuales cubren una serie de temáticas más o menos específicas, y serán las clases para nuestro sistema. Estas clases serán alimentadas por un conjunto de artículos pertenecientes a las revistas, los cuales actúan como ejemplo, y cuyo texto conformará el conjunto de atributos que se requiere. Con esta información se construye un clasificador que da como resultado un ranking de revistas al mandarle un artículo de ejemplo.

La verdadera peculiaridad de nuestro problema está en sus dimensiones, ya que no contamos con un número reducido de clases, sino con miles de clases. Esto complica la etapa de aprendizaje no solo en términos de complejidad, sino en términos de tamaño de almacenamiento.

Será un viaje a través de métodos de aprendizaje automático relevantes que nos ayudarán a alcanzar nuestro objetivo: entender cada algoritmo y cuál es el mejor para abordar el problema planteado. Exploraremos distintas configuraciones para el problema, así como distintos métodos para entrenar el clasificador y realizaremos las comparaciones correspondientes, valiéndonos de las métricas adecuadas para ver la calidad de nuestro clasificador en todo momento.

1.2 Capítulos y contenido general

He decidido dividir este trabajo en 10 capítulos distintos, y algunas secciones adicionales. Voy a explicar uno por uno su contenido. En términos generales he dividido mi trabajo en cuatro bloques diferenciados: introducción al problema, procesado de los datos, pruebas sin texto completo y pruebas con texto completo:

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

1. **Introducción:** Introducción al problema que vamos a ver, y dando información acerca de la importancia de este en términos generales.
2. **Trabajos de interés:** En este capítulo se analizarán diversos trabajos interesantes acerca de los Sistemas de Recomendación, y cómo han sido trabajados.
3. **Presupuesto:** Análisis de tiempo y recursos necesarios para llevar a cabo un proyecto de creación de Sistema de Recomendación.
4. **Clasificación textual:** Aquí se define de manera más completa la Clasificación Textual. Incluye tanto los pasos de extracción de características como explicaciones para cada uno de los métodos con los que vamos a trabajar.
5. **Dataset escogido:** Se definen de forma específica los datos con los que vamos a trabajar. Veremos de manera más completa tanto los atributos como las clases del problema al que nos enfrentamos, así como estadísticas y una explicación completa del contenido de los datos.
6. **Procesado de CORD-19:** Explicación extensa de la limpieza y preprocesado de los datos.
7. **Datasets y variaciones:** En este apartado se explicarán cada uno de los datasets distintos con los que vamos a trabajar, junto con las variaciones de cada uno.
8. **Resultados obtenidos:** En este apartado se realizan las pruebas con cada dataset, utilizando técnicas tanto de Machine Learning como de Deep Learning, y se analizarán sus resultados de cara a sacar conclusiones sobre estos.
9. **Dataset con texto completo:** En este apartado veremos los pasos para integrar nuestro dataset limpio con los artículos que contienen texto completo, además de ver las pruebas y de analizar los resultados obtenidos.
10. **Conclusión:** Realizaremos una vista general de las pruebas realizadas y resultados obtenidos, e indicaremos las conclusiones más importantes que hemos obtenido tras todo el estudio realizado. Adicionalmente, hablaré sobre mi experiencia previa al trabajo y sobre mi experiencia personal al haber realizado este trabajo.

Tras el capítulo 10 tenemos a disposición el apéndice en donde encontraremos los archivos utilizados para la realización del trabajo, el apartado ilustraciones y la bibliografía.

Capítulo 2

Trabajos de interés

Para ponernos en contexto cabe mencionar otros trabajos los cuales ya han explorado este problema que veremos, y que considero interesantes de conocer.

Uno de ellos es el publicado por Hossein Dehdarirad y Javad Ghazimirsaeid en febrero de 2019: Scholarly publication venue recommender systems - A systematic literature review [3].

Este trabajo realiza una investigación sobre distintas maneras de afrontar el problema de recomendación, realizando búsquedas a través de cuatro bases de datos (Scopus, Web Of Science, IEEE Xplore y ScienceDirect) de estudios que tratan este tema realizados hasta 2018. En este trabajo se puede apreciar cómo el ámbito más explorado es el de sistemas basados en contenido y colaborativos.

De cada trabajo se extrae el nombre del estudio, las fuentes de datos que utiliza, el tipo de sistema de recomendación, los métodos de entrenamiento que utiliza, el dataset que utiliza como punto de partida y la metodología de evaluación, las cuales los autores definen como tres tipos distintos:

- Offline: No considera intervención de un usuario final en la evaluación. Usa datos del dataset y se espera que las predicciones coincidan con esos datos escogidos.
- User: Una serie de usuarios realizan pruebas y tras haber probado el sistema correctamente se responde a preguntas de interés.
- Online: Evalúa el rendimiento a gran escala, llevando el sistema al mundo real y los usuarios no tienen directrices a la hora de utilizar el sistema.

Centrándonos en los sistemas basados en contenido, apreciamos que las fuentes de datos más usadas son el título y el resumen del artículo, y como tercer candidato las keywords. Entre todos los trabajos que se estudian podemos destacar el artículo publicado por Martijn J. Schuemie y Jan A. Kors en enero de 2008: Jane - suggesting journals, finding experts [4], en el cual se utiliza el dataset [PubMed](#), que contiene una recopilación de artículos relacionados con medicina y biotecnología.

Toma como datos el título y el resumen de cada artículo, junto con varios filtros como idioma y tipo de publicación, y estos se entrenan para clasificación textual devolviendo como salida un ranking de revistas y de autores que se ajusten al artículo dado.

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Otro trabajo que me gustaría destacar es el publicado por Zaihan Yang y Brian D. Davison en junio de 2012: Distinguishing Venues by Writing Styles [5], en el cual, como su propio nombre indica, busca realizar una clasificación textual de revistas/conferencias en base al estilo de escritura, y utilizando tres tipos de características: léxicas, sintácticas y estructurales.

Lo que personalmente me gusta de este trabajo es que realiza pruebas con varios algoritmos de Machine Learning distintos, y es ese el objetivo de este TFG: un estudio comparativo de varias maneras de afrontar un distinto problema, con sus pros y sus contras característicos y aplicando distintas alternativas.

Un trabajo muy interesante, el cual no entra en [3], y es más reciente, es el Harinni Kodur y Tanya Tyagi en diciembre de 2019: ACM Venue Recommendation System CS6604 – Digital Libraries Final Project Report [6].

Este trabajo de investigación busca solucionar el problema de recomendación, en este caso de revistas, para el dataset de la [Biblioteca Digital ACM](#) (ACM-DL). Este dataset se compone de archivos XML con la información necesaria para realizar el correcto entrenamiento del Sistema de Recomendación.

Realiza una limpieza previa de los datos, por ejemplo, con revistas que han cambiado de nombre con el paso de los años, las cuales generan inconsistencias a la hora del entrenamiento. En esta etapa también tenemos un borrado de stopwords y de palabras muy comunes, e incluso ordinales como “first”, “second”, etc, que a primera instancia no aportan nada al dataset. Situándonos en las columnas disponibles, son relevantes el texto completo, el resumen, los autores, las palabras clave y la revista, siendo los cuatro primeros los datos a entrenar y la revista la clase a clasificar tras el entrenamiento.

Mediante el uso de Redes Neuronales Artificiales y Convolucionales (Deep Learning) busca solucionar su problema de recomendación. La peculiaridad de su trabajo es la creación para cada revista un clasificador binario, y más tarde estos se combinan para crear un clasificador multiclase. De todos los trabajos que hemos visto este es el único que trabaja con las redes neuronales, una tecnología más avanzada y reciente que los métodos de Machine Learning.

Para concluir, hemos visto cómo todos los artículos analizados anteriormente descartan el texto completo para realizar el entrenamiento del sistema, por lo que es un tema interesante el cual explorar: las posibles diferencias entre contar con un conjunto reducido de palabras, como el resumen, y un gran conjunto de palabras, como el de un artículo al completo.

Capítulo 3

Presupuesto

En este apartado analizaremos el caso de llevar mi proyecto a escala mayor, y los costes totales que conllevaría. Nos situaremos en el caso de que una empresa requiriese de mis conocimientos de Ciencia de Datos para realizar un estudio sobre posibles métodos para acelerar y mejorar un Sistema de Recomendación. Dispongo de 2 meses para presentar una mejora significativa al sistema con el que cuentan.

3.1 Planificación

En este apartado mediante una tabla realizaré una planificación de horas con descripción de las tareas realizadas (*tabla 1*):

Tabla 1: Planificación del problema

Recurso	Días empleados
Análisis del problema	3
Realizar instalación de los recursos necesarios	1
Explorar opciones disponibles	15
Realización de diversas pruebas	20
Recopilación y análisis de resultados	15
Redacción de conclusiones y alternativas	6
	60

3.2 Mano de obra

En el grupo de trabajo solamente me encuentro yo: un ingeniero software. En la *tabla 2* se recoge la información respecto al salario. El salario medio anual de este puesto de trabajo está sacado de la web GlassDoor (<https://www.glassdoor.es/index.htm>). :

Tabla 2: Salario mano de obra

Puesto	Horas	Salario anual medio	Salario por mes	Salario total
Ingeniero software	480	31.500€	2.610€	5.220€

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Al salario individual de cada trabajador se le tiene que restar cada mes un porcentaje de IRPF (Impuesto sobre la Renta de las Personas Físicas), por lo que el salario total del trabajador sería:

- Ingeniero software: $(2.610\text{€} - 30\% \text{ IRPF}) * 2 \text{ meses} = 3.654\text{€}$

3.3 Recursos materiales

En este apartado nos enfocaremos en las necesidades materiales del trabajador y del propio proyecto. Se dividen en recursos software y recursos hardware. La empresa proporcionará todo tipo de material necesario. La manera de trabajar escogida es remota, por lo que se requerirá equipo acorde a las necesidades.

Tabla 3: Recursos hardware

Recurso	Cantidad	Precio
Portátil Dell Inspiron 15 3000 (2022)	1	559€
Logitech MK235 Combo Teclado y Ratón Inalámbrico	1	19'99€
Monitor - Xiaomi OB02608 Mi Desktop Monitor 27" EU, Full HD, 75hz	1	129€
		707'99€

Tabla 4: Recursos software

Recurso	Cantidad	Precio
SO Ubuntu 22.04	1	0€
LibreOffice 7.3.3	1	0€
Google Colab Pro+ Mensual	2	84'50€
		84'50€

Como podemos observar, en la *tabla 4* el coste es nulo, debido a que los programas que necesitamos no requieren una licencia. Lo que si requeriremos es contrataciones de servidores y hostings. He escogido Google Colab, ya que para realizar las pruebas se realiza todo absolutamente en remoto, además de que cuenta con el acceso a las TPU, que son tarjetas gráficas creadas específicamente para entrenamientos de redes neuronales, lo que aumenta la velocidad muchísimo.

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Los recursos con los que contamos en Google Colab Pro+ son los siguientes [7]:

- 52GB RAM.
- GPUs K80, T4 y P100 con prioridad máxima.
- Ejecución en segundo plano de hasta 24 horas.

En cuanto a la *tabla 3*, he considerado que no es necesario un ordenador extremadamente potente, por lo que utilizar un ordenador de gama media es suficiente para poder realizar las pruebas con soltura. El monitor adicional agiliza enormemente el trabajo, y es un coste asumible.

3.3 Coste total

Para finalizar este capítulo, realizaremos la suma total de coste del proyecto. En la tabla siguiente (*tabla 5*) se encuentran diferenciados los costes. Considero un proyecto asequible para la empresa, y bien pagado por parte del Ingeniero Software.

Tabla 5: Tabla de costes totales

Tipo de recurso	Precio
Recursos humanos	5.220€
Recursos Hardware	707'99€
Recursos Software	84'50€
	6012'49€

Capítulo 4

Clasificación textual

La clasificación textual [8] es el ámbito que categoriza texto dentro de grupos organizados, en base a unas características dadas. Es una de las tareas que cubre el Machine Learning y el Deep Learning, y todo es posible mediante el Natural Language Processing. Natural Language Processing, también conocido como NLP, es la habilidad de una máquina de entender el lenguaje humano, tanto hablado como escrito. Existe desde hace 50 años y se encuentra muy presente en el marco empresarial actual y en la ciencia.

Al principio se parte de un dataset que contiene datos textuales en uno o varios campos distintos. Estos textos en distintos campos se dividen en:

- Clases: Agrupación a la que pertenece el texto dado. Normalmente se compone de un solo campo.
- Características: Información que permite agrupar correctamente las clases existentes en nuestro dataset. Se compone de uno o más campos textuales.

El texto proveniente de las diversas características es tratado como una bolsa de palabras, y será lo que se utilice para entrenar un algoritmo escogido, el cual tras el entrenamiento tendrá la capacidad para realizar predicciones sobre la clase a la que pertenece el texto que le es suministrado.

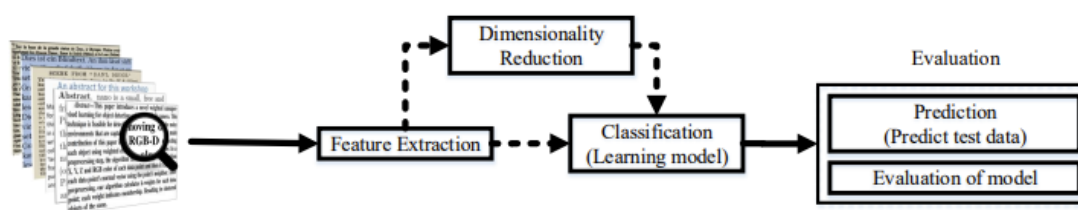


Ilustración 4 Pipeline de la clasificación de texto [8]

En la *ilustración 4* observamos las partes fundamentales dentro de la clasificación textual:

- Extracción de características: Las secuencias de texto deben ser convertidas previamente, antes de que el algoritmo de clasificación entrene el modelo. Previo a la extracción de características hay un proceso crucial por el que nuestro dataset debe pasar, y será decisivo para garantizar una eficacia superior del modelo a entrenar: la limpieza y el preprocesamiento de texto. Tras este proceso se puede realizar la conocida reducción de

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

dimensionalidad: Reduce el conjunto de características de problema de cara a mejorar el coste de tiempo y la complejidad de memoria. Este paso es opcional.

- Entrenamiento del modelo: Mediante diversas técnicas de aprendizaje automático se busca emplear el conjunto de características extraído anteriormente, y entrenar el modelo correctamente de cara a la etapa de predicción. Se aplican tanto técnicas de Machine Learning como de Deep Learning. Distintos métodos se verán a continuación.
- Predicción y evaluación: En esta etapa se pone a prueba el modelo entrenado previamente, y mediante diversas métricas se evalúa su eficacia. La métrica usada dependerá en gran medida del resultado que se espera.

A continuación, hablaremos de lo mencionado en la extracción de características, pero de manera amplia. Es un paso muy importante de cara a un correcto entrenamiento y rendimiento.

4.1 Extracción de características

Aquí se definirán los distintos pasos durante la extracción de características de manera amplia, entrando en mayor detalle en cada uno de los pasos que existen dentro de esta parte.

4.1.1 Limpieza del dataset

Se le llama limpieza de dataset a la conversión y borrado de datos que pueden generar problemas a la hora de clasificar nuestro modelo de aprendizaje automático [9]. Este proceso de limpieza comprende varias tareas que deben de tenerse en cuenta:

- Paso a minúsculas de las clases existentes: De cara a las siguientes tareas, es importante pasar a minúsculas todas las clases con las que trabajemos.
- Borrado de tuplas duplicadas: Muchas veces podemos encontrarnos con tuplas duplicadas en el dataset, y eso puede provocar un mal entrenamiento de clases concretas, o un excesivo entrenamiento de estas. Es por esto que se debe de realizar una limpieza de este tipo de tuplas.
- Unión de clases iguales con nombre diferente: En algunos casos, por ejemplo, el de revistas que han cambiado con el paso del tiempo de nombre, pero cuyo contenido se ha mantenido intacto, se puede dar que se mencione la misma revista con nombres distintos. Esto también puede darse con clases que utilicen nombre completo y clases que utilicen siglas. Es importante encontrar estas clases y unir las de cara a mejorar el entrenamiento y la predicción de la propia clase.

- Borrado de tuplas con contenido nulo: Las tuplas con contenido nulo, tanto en las características como en las clases, no aportan información ninguna al modelo y deben de borrarse.

Una vez limpio el dataset lo siguiente es aplicar un correcto preprocesamiento al texto con el que conformaremos el conjunto de características.

4.1.2 Preprocesado de texto básico

En esta parte se realiza una limpieza del texto de cara a facilitar el trabajo a los pasos siguientes. Su propósito es limpiar el texto de manera que no se encuentren fenómenos como el hecho de tener palabras distintas por su uso de mayúsculas, por ejemplo. Algunas de las posibles tareas son las siguientes:

- Conversión a minúsculas: evita la distinción de palabras por el uso de mayúsculas.
- Conversión de todos los caracteres a ASCII: De esta manera, nos aseguramos que nuestro texto puede ser leído correctamente por cualquier método que utilicemos.
- Borrado de números: Los números no tienen un aporte excesivo a la hora de entrenar, por lo que es mejor no contar con ellos.
- Borrado de caracteres especiales: Caracteres especiales son aquellos caracteres que no pueden ser leídos por algunos métodos, o simplemente no aportan significado alguno de cara al entrenamiento.

4.1.3 Tokenización

El tokenizado consiste en la separación de cada palabra del dataset en tokens de cara a realizar cambios a nivel de término. Este proceso permitirá realizar más pasos del preprocesado, ya que es uno de los pasos centrales.

4.1.4 Borrado de palabras vacías

Los stopwords [10] o palabras vacías son palabras muy frecuentes cuyo aporte al entrenamiento general del modelo es muy bajo. Son palabras vacías los pronombres, artículos, determinantes, etc. No cabe en la mayoría de los casos la decisión de mantener estas palabras, ya que, además de carecer de significado para el entrenamiento, son extremadamente frecuentes y al figurar en un porcentaje muy alto del texto no tienen aporte ninguno.

4.1.5 Lematización

El stemming o lematización es el proceso por el cual se busca convertir la forma de las palabras del texto a su forma básica, por ejemplo: accept -> acceptable, acceptance, acceptances, acceptation, accepted, accepting. Este método nos permite obtener un texto básico, reduciendo cada palabra a su raíz y ayuda a reducir el espacio de almacenamiento y a mejorar la eficiencia de la búsqueda [11]. Este proceso es muy importante en el ámbito del Machine Learning, pero no es necesario en el Deep Learning.

4.1.6 Vectorizado

El vectorizado es el último paso dentro de la extracción de características. Consiste en la separación del texto disponible en el dataset de forma que se conviertan en números reales, ya que este es el formato que soportan los métodos de entrenamiento de Machine Learning y Deep Learning: la llamada bolsa de palabras. Un método más refinado para llevar a cabo esta tarea es el vectorizado Tf-Idf [12].

El esquema Tf-Idf evita que los documentos largos tengan ventaja por frecuencia sobre los documentos cortos, y eso se consigue juntando los dos conceptos siguientes:

- tf: Frecuencia del término en el documento. Se basa en la hipótesis de que un término que aparece varias veces en un documento es más importante que uno que aparece una sola vez.
- idf: Frecuencia documental inversa. Se basa en la hipótesis de que un término que aparece pocas veces en la colección tiene más poder discriminador que uno que aparece en todos los documentos.

$$idf(t) = \log \left[\frac{n}{df(t)} \right] + 1$$

La fórmula general del Tf-Idf es la siguiente:

$$tf - idf(t, d) = tf(t, d) * idf(t)$$

Siendo t el término, d el documento, df(t) es la frecuencia documental, y n el número total de documentos. La razón por la que al idf se le hace +1 es porque si no los términos con df(t)=n serían ignorados, y de esta manera se tienen en cuenta muy suavemente.

Para el vectorizado puede servir trabajar, además de con palabras sueltas, con secuencias de palabras, ya que añade contexto y secuencias de palabras juntas pueden ser discriminatorias para una clase. Es aquí en donde entran los n-gramas.

Los n-gramas son secuencias de n palabras contiguas provenientes de un texto. Se asignan probabilidades a las secuencias de palabras existentes, es decir, se tienen en cuenta secuencias de n palabras dentro de la bolsa de palabras [13]. Lo más común dentro de los n-gramas es el uso de bigramas, es decir, secuencias de dos palabras como máximo.

4.1.7 Word Embedding (opcional)

El proceso de embedding, es el proceso de conversión de datos de altas dimensiones a una dimensión menor en forma de vector, manteniendo su semántica similar [14]. Aplica técnicas de Machine Learning para aprender las relaciones entre las palabras. Una de las utilidades más importantes es que resuelven el problema de dispersión de los datos en un dataset.

Un método interesante es Word2Vec. Este método realiza un embedding mediante dos métodos posibles de Deep Learning [15], e intenta aprender las asociaciones de las palabras a partir de un conjunto de texto dado. Los posibles métodos de Deep Learning con los que se puede realizar son los siguientes:

- Common Bag Of Words: También conocido por las siglas CBOW, es un modelo que toma el contexto de cada palabra como entrada e intenta predecir la palabra con el propio contexto.
- Skip-Gram: Intenta obtener el contrario a CBOW, que es predecir las palabras del contexto dada una palabra [16].

El método FastText es similar a Word2Vec, pero con la peculiaridad de integrar n-gramas de palabras durante el entrenamiento. Esto aumenta el tamaño y el tiempo de procesamiento del modelo, pero se adquiere la habilidad de predecir variaciones de palabras [17].

Este paso es muy popular en el caso del Deep Learning y las redes neuronales, ya que se deben de procesar un conjunto muy grande de atributos distintos y ayuda mucho a disminuir el tamaño en términos de memoria, y a aumentar la velocidad de entrenamiento. Lo más importante es que en la gran mayoría de los casos un embedding bien hecho no supone una pérdida significativa de calidad en el modelo de predicción.

4.2 Modelos de clasificación

Los métodos que van a verse a continuación pertenecen a la etapa de entrenamiento. Son maneras distintas de afrontar el mismo problema, que es el de la clasificación textual.

Los algoritmos que vamos a ver a continuación tienen en común que utilizan la bolsa de palabras derivada de la extracción de características. En esta parte veremos los siguientes métodos de aprendizaje:

- [Machine Learning] Multinomial Naive Bayes
- [Machine Learning] K-Nearest Neighbors
- [Machine Learning] Random Forest
- [Machine Learning] Logistic Regression
- [Machine Learning] Complement Naive Bayes
- [Deep Learning] Long-Short Term Memory

4.2.1 Multinomial Naive Bayes

El algoritmo Multinomial Naive Bayes [18] consiste en una comparación de probabilidades entre clases dado un conjunto de atributos, y es un algoritmo muy potente para problemas relacionados con Natural Language Processing (NLP). Se calcula para cada clase la probabilidad de que, dado un conjunto de atributos, la predicción sea la propia clase, y se escoge la clase con mayor probabilidad. La fórmula es la siguiente:

$$P(Class|Att) = \frac{(\prod_{i=1}^n P(Att_i|Class)) * P(Class)}{P(Att)}$$

Siendo Class la clase y Att el conjunto de atributos. Como observamos se realiza un producto de probabilidades de que cada atributo individual pertenezca a la clase dada $P(Att_i|Class)$. Esto último se justifica con la llamada presunción de independencia, la cual dice que cada atributo de una clase contribuye de manera independiente a la probabilidad de la clase. Este principio es común para todos los algoritmos derivados del Teorema de Bayes.

4.2.2 K-Nearest Neighbors

El método KNN consiste en la realización de un cálculo de distancias entre la instancia a clasificar que le damos y las demás del modelo de entrenamiento. Se toman los K puntos más cercanos y la instancia que queremos clasificar pertenecerá a la clase con más coincidencias dentro de esos K vecinos más cercanos [19].

Veremos a través del siguiente ejemplo sencillo el funcionamiento de este algoritmo:

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

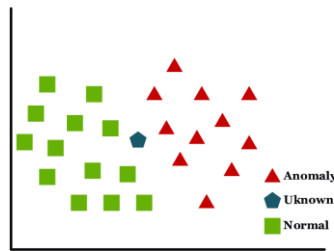


Ilustración 5 Ejemplo espacio KNN [20]

Veamos la *ilustración 5*: la información se muestra como puntos de datos en un espacio de atributos de 2 dimensiones. Se busca clasificar el ítem “desconocido”. En este caso tomándose los 3 vecinos más cercanos (3NN) el ítem sería clasificado en la clase “normal”.

Hay dos aspectos a mencionar sobre este método: tipos de organización del espacio, y maneras de calcular las distancias entre puntos de este espacio. Se pueden diferenciar tres maneras de organizar los puntos del espacio [21]:

- KD Tree: Los puntos del espacio se distribuyen en un espacio euclideo de K dimensiones. Tiene una estructura de árbol binario y se divide en nodos que contienen puntos. Se busca mejorar el rendimiento con esta representación, y solamente está disponible para datasets no dispersos. La dispersión (sparse), significa que en la matriz de pesos en muchos casos existen pesos cero [22].
- Ball Tree: Parecido a KD Tree pero los datos se almacenan en hiper-esferas anidadas. Mucho más eficiente para dimensionalidades altas pero más lento. Solamente se encuentra disponible en datasets no dispersos.
- Brute Force: Método de cálculo de distancias estándar. Método más lento.

El correcto cálculo de distancias es otro factor crucial a la hora de buscar una buena eficiencia y calidad de predicción. Se pueden diferenciar tres maneras distintas de calcular las distancias:

- Distancia euclídea: Distancia ordinaria. Se calcula de la siguiente manera para un caso bidimensional [23]:

$$d_E(x,y) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- Distancia Manhattan: La distancia entre dos puntos es la suma de las diferencias absolutas de sus coordenadas, es decir, la suma de las longitudes de los dos catetos del triángulo rectángulo (geometría del taxi) [24]. La fórmula para el caso bidimensional es la siguiente:

$$d_M(x, y) = |x_1 - y_1| + |x_2 - y_2|$$

- Distancia Minkowski: Generalización entre la distancia euclídea y la distancia Manhattan. Si $p=1$ es la distancia Manhattan y $p=2$ es la distancia euclídea [25]:

$$d_C(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

4.2.3 Random Forest

Construye múltiples árboles de decisión durante el entrenamiento. Los datos pasan por cada árbol y se toma la decisión más común entre todos ellos. Un árbol de decisión es un árbol que representa una toma de decisiones [26].

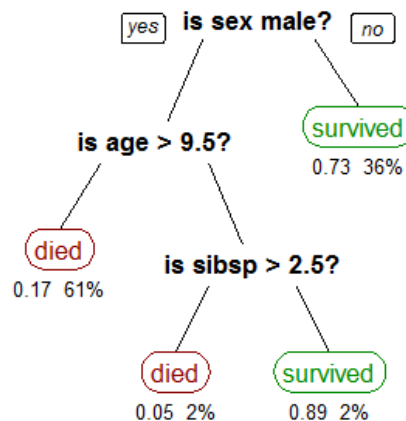


Ilustración 6 Ejemplo árbol de decisión [n]

Este ejemplo (ilustración 6) es sencillo: tenemos que predecir si una persona va a sobrevivir a un accidente de avión en base a tres atributos distintos: sexo, edad y número de niños alrededor. Cada nivel es una respuesta y se van realizando podas del árbol, descartando ramas y avanzando en nivel en otras, hasta llegar a una respuesta, que sería la predicción.

El método Random Forest es muy utilizado en los problemas de clasificación por su eficiencia con grandes cantidades de datos y sus buenas predicciones, además de que combate el sobreentrenamiento [27].

Para maximizar la eficacia se necesitan un conjunto de atributos con cierto poder predictivo y discriminatorio. Otra manera de mejorar la predicción será escogiendo correctamente el criterio que va a utilizarse a la hora de hacer las particiones (split) del árbol:

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

- Criterio Gini: Frecuencia entre 0 y 0.5 con la cual cualquier elemento del dataset será clasificado incorrectamente cuando es clasificado de manera aleatoria [28].
- Criterio Entropy: Medida de desorden entre 0 y 1 de los atributos sobre la clase objetivo [28].
- Criterio Log-loss: Probabilidad entre 0 y 1 de que la predicción sea verdadera con la partición realizada [29].

Tras crearse una serie de árboles para nuestro problema, con la profundidad deseada (a mayor profundidad mayor grado de exactitud), se pasará el mismo conjunto de atributos a todos los árboles. Se obtienen los resultados de las predicciones de cada árbol y la predicción más común entre todos ellos es la que se toma como decisión final.

4.2.4 Multinomial Logistic Regression

El método [30] está pensado principalmente para problemas de clasificación binaria, pero veremos cómo se puede aplicar a un problema de clasificación multiclase. Primero hablaremos del problema binario: se basa en el cálculo de probabilidades para las clases existentes en el modelo dado un conjunto de atributos.

En este problema los resultados se comprenden entre 0 y 1, por lo que si el score de una probabilidad está por encima de 0.5, la clase es A, y si está por debajo, la clase es B (*ilustración 7*).

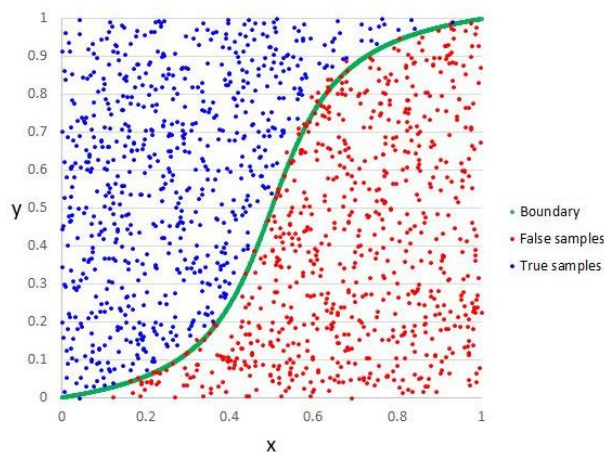


Ilustración 7 Scatter plot con sigmoide [31]

Se trabaja con un conjunto de atributos, los cuales tienen una serie de pesos (coeficientes de regresión), y se ajustarán durante el entrenamiento del modelo. La fórmula que contiene cada peso es la siguiente:

$$w(Att) = b_0 + Att_1b_1 + \dots + Att_nb_n$$

Esta fórmula es conocida como función lineal. La función lineal está compuesta de un conjunto de atributos Att , y b_0, b_1, \dots, b_n , que son los coeficientes de regresión de los cuales hemos hablado anteriormente. Para que el resultado de la función anterior se comprenda entre 0 y 1 se usa la función sigmoide. La función es la siguiente:

$$S(X) = \frac{1}{1 + e^{-x}}$$

La función principal de la Regresión Logística consiste en la unión entre la función sigmoide y la función de ajuste de pesos:

$$P(Class|Att) = S(w(Att)) = \frac{1}{1 + e^{-w(Att)}}$$

Vemos cómo finalmente la función es un cálculo de probabilidades de que, dado un conjunto de atributos Att la clase sea $Class$. Como se ha dicho anteriormente, se comparan los scores y si están por encima de 0.5 se pertenece a una clase, y si están por debajo pertenece a otra.

Ahora que hemos hablado del problema binario, hablaremos del problema multiclase. Se puede aplicar mediante dos algoritmos distintos:

- Multinomial: Aplica la función SoftMax, la cual es:

$$P(Class|Att) = \frac{e^{w(Att)}}{\sum_{i=0}^k e^{w(Att_i)}}$$

Se aplica para cada clase la probabilidad de que la predicción sea $Class$ dado el conjunto de atributos Att . Se realiza la sumatoria para todas las clases posibles [32]. Se realiza ese cálculo de probabilidades y el resultado es la clase más probable es el resultado final.

- One VS Rest: Se crea un modelo binario de pertenencia a cada clase, se genera un score de probabilidad de que, dados unos atributos, se pertenezca (o no) a la clase, y con la ayuda de la función argmax (devuelve el índice de la clase con mayor score) se escoge la clase más probable [33].

4.2.5 Complement Naive Bayes

Este algoritmo está pensado para corregir las suposiciones severas que realiza Multinomial Naive Bayes. Es muy adecuado para conjuntos de datos desbalanceados, así que podríamos ver una gran diferencia en este método respecto a los demás.

La diferencia entre Complement Naive Bayes y Multinomial Naive Bayes es que, en lugar de calcular la probabilidad de que, dado un elemento con sus propios atributos, ese elemento

pertenezca a alguna clase, se calcula la probabilidad de que ese elemento sea parte de todas las demás clases, y eso es denominado complemento. Se escoge la clase con menor probabilidad [34].

4.2.6 Long-Short Term Memory

Este método es el único del que hablaremos que pertenece al conjunto de las redes neuronales y el Deep Learning. Long-Short Term Memory (LSTM) es un tipo de red neuronal que pretende resolver numerosas tareas que no pueden ser resueltas con otros tipos de redes neuronales recurrentes de manera eficiente. Se basa en el principio de recordar información anterior para que sea influyente en el procesamiento de la información actual [35]. Sus tareas fundamentales son:

1. Olvidar información irrelevante
2. Agregar/actualizar nueva información
3. Enviar la información actualizada al siguiente paso

Las dos unidades mínimas de información de este método son [36]:

- Cell State: Memoria a largo plazo del modelo LSTM. Unidad de información que es transferida de unidad de tiempo en unidad de tiempo.
- Hidden State: Memoria a corto plazo del modelo LSTM. Almacena información de la unidad de tiempo inmediatamente anterior.

Primeramente, se decide si la información de la célula anterior se mantiene o se olvida, después se intenta actualizar la célula con nueva información proveniente del input, y finalmente se envía la célula actual al siguiente estado.

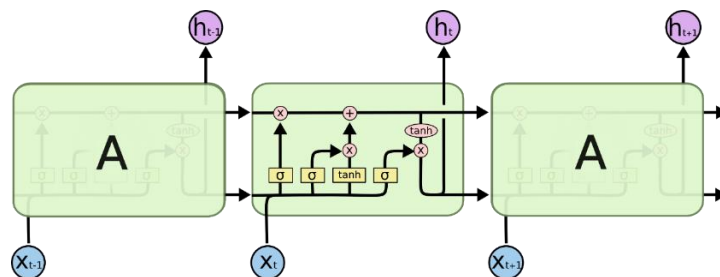


Ilustración 8 Estructura interna de red LSTM [35]

La información va acompañada de una serie de pesos, provenientes de una matriz de embedding, y con ello se realizarán operaciones para decidir la importancia de la nueva información, de la información previa, y tomar decisiones al respecto.

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Tras haber hablado ampliamente de la clasificación textual, y habernos puesto en contexto sobre el problema que va a ser abordado, vamos a entrar en materia y ver los datos con los que contamos y el preprocesamiento realizado.

Capítulo 5

Dataset escogido

El dataset “COVID-19 Open Research Dataset (CORD-19)” de la plataforma Kaggle, es una recopilación de artículos sobre COVID-19 y SARS-CoV-2 [37]. El 16 de marzo de 2020, el Allen Institute for AI (AI2), con el apoyo de organizaciones de renombre como Microsoft Research o The White House Office of Science and Technology Policy (OSTP), y con la ayuda de Kaggle, publicaron la primera versión de CORD-19. El dataset inicial contenía aproximadamente 28.000 artículos, y el 80% de estos artículos contenían texto completo [38].

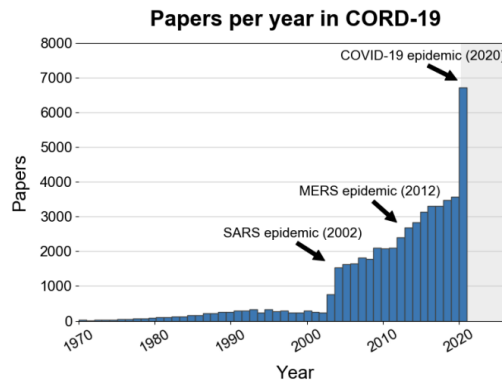


Ilustración 9 Gráfica de artículos de CORD-19 hasta 2020 [38]

En la *ilustración 9* observamos cómo, a raíz de la pandemia mundial a principios de 2020, ha aumentado el número de artículos, y así seguirá siendo hasta el fin de la pandemia. Actualmente puedes encontrar más de 500.000 artículos académicos (200.000 con texto completo).

El objetivo actual en Kaggle para este dataset es la investigación mediante herramientas de Inteligencia Artificial maneras de responder a preguntas científicas de alta prioridad, con premios de hasta 1.000 dólares, sin embargo, mi enfoque será distinto: tomaremos el enfoque de la recomendación de revistas en forma de ranking, como he comentado previamente.

```
Kaggle/  
  cord_19_embeddings/  
  document_parsers/  
    pdf_json/  
      0000028b5cc154f68b8a269f6578f21e31f62977.json  
      ...  
    pmc_json/  
      COVID.DATA.LIC.AGMT.pdf  
      json_schema.txt  
      metadata.csv  
      metadata.readme
```


Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Inmediatamente arriba podemos ver el árbol de directorios del dataset. De todo el árbol de directorios es de nuestro interés la siguiente información:

- `document_parsers/pdf_json`: En esta carpeta encontraremos más de 400.000 artículos acompañados de su texto completo, entre otra información relevante. Dentro de `document_parsers/pmc_json` nos encontramos con información similar, con la diferencia que esos archivos son los incluidos en la otra carpeta, pero que contienen el campo PMCID relleno.
- `metadata.csv`: Contiene información relevante del artículo, como el id SHA1 (Cryptographic Hash Algorithm), título del artículo, resumen (abstract) del artículo, revista a la que pertenece, id DOI, autores, URL, etc.

La estructura de los artículos tipo JSON de `document_parsers/pdf_json` es la siguiente:

- `paper_id` (str): id SHA1 del artículo.
- `metadata`: Estructura que contiene los siguientes campos:
 - `title`: Título del artículo.
 - `authors`: Lista de autores que contiene los campos nombre, apellido, sufijo, email, medio para el que trabaja y de dónde proviene.
 - `abstract`: Lista de párrafos del resumen que contiene los párrafos del resumen, las citas bibliográficas y secciones del resumen.
 - `body_text`: Lista de párrafos del texto completo que contiene los párrafos del texto, las citas bibliográficas y secciones del artículo.
 - `bib_entries`: Lista de entradas bibliográficas que se han ido recopilando anteriormente. Contiene una estructura BIBREFN que almacena el id de la bibliografía, título, autores, año, ISSN, etc.
 - `ref_entries`: Lista de entradas como ilustraciones, tablas, etc, que se han ido recopilando anteriormente. Contiene una estructura FIGREFN que almacena el texto del objeto y el tipo.
 - `back_matter`: Lista que contiene información final del artículo.

En la siguiente página podemos observar un ejemplo de archivo JSON y otro del CSV, para poder diferenciarlo mejor (*ilustración 10*). Aunque incompleto, nos sirve para visualizar los datos que vamos a ver.

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

```
{
  "paper_id": "7abc008948e36df6f7f8e8c0747ab069f29429b2",
  "metadata": {
    "title": "",
    "authors": [
      {
        "first": "M\u00e9rio",
        "middle": [],
        "last": "Morais-Almeida",
        "suffix": "",
        "affiliation": {
          "laboratory": "",
          "institution": "CUF Descobertas Hospital",
          "location": {
            "settlement": "Lisbon",
            "country": "Portugal"
          }
        }
      },
      {
        "email": "mmoraisalmeida@netcabo.pt.telephone:351917232267."
      }
    ]
  },
  "abstract": [
    {
      "text": "Historically, mental health ..."
      "cite_spans": [
        {
          "start": 67,
          "end": 70,
          "text": "(5)",
          "ref_id": "BIBREF4"
        },
        {
          "start": 343,
          "end": 346,
          "text": "(6)",
          "ref_id": "BIBREF5"
        }
      ],
      "ref_spans": [],
      "section": ""
    }
  ],
  "body_text": [
    {
      "text": "*Both authors contributed equally to this paper",
      "cite_spans": [],
      "ref_spans": [],
      "section": ""
    },
    {
      "text": "Based on current recommendations, ..."
      "cite_spans": [],
      "ref_spans": [],
      "section": "Accepted Article"
    }
  ],
  "bib_entries": {
    "BIBREF0": {
      ...
    }
  }
}
```

Recomendación de revistas científicas utilizando técnicas de aprendizaje automático

cord_uid	sha	source_x	title	doi	pmcid	pubmed_id	license	abstract	publish_time	authors	journal
0	ug7v89j	d1aafb70c066a2068b02786f829f93c900837b	PMIClinical features of culture-proven Mycoplasma...	10.1186/1471-2334-1-6	PMC35282	11472656	no-cc	OBJECTIVE: This retrospective chart review des...	2001-07-04	Madani, Tariq A; Al-Ghamdi, Asma A	BMC Infect Dis
1	02bnwd4m	6b0567729c2143a66d737eb0a2b63f2dce2e5a7d	PMICNitric oxide: a pro-inflammatory mediator in L...	10.1186/rn14	PMC59543	11667967	no-cc	Inflammatory diseases of the respiratory tract...	2000-08-15	Vliet, Albert van der; Eisenich, Jason P; Crook...	Respir Res
2	ejv2xln0	06ced00a5fc04215949aa72528f2eaae1d58927	PMICSurfactant protein-D and pulmonary host defense	10.1186/rv19	PMC59549	11667972	no-cc	Surfactant protein-D (SP-D) participates in th...	2000-08-25	Crouch, Erika C	Respir Res
3	2b73a25n	348055649b6b8cf2b9a376495d9b041f7123605	PMICRole of endothelin-1 in lung disease	10.1186/rn44	PMC59574	11686871	no-cc	Endothelin-1 (ET-1) is a 21 amino acid peptide...	2001-02-22	Fagan, Karen A; McMurtry, Ivan P; Rodman, David M	Respir Res
mag_id	who_covidenc_id	arxiv_id	pdf_json_files		pmc_json_files		url		s2_id		
NaN	NaN	NaN	document_parses/pdf_json/d1aafb70c066a2068b027...	document_parses/pmc_json/PMC35282.xml.json	https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3...		NaN				
NaN	NaN	NaN	document_parses/pdf_json/6b0567729c2143a66d737...	document_parses/pmc_json/PMC59543.xml.json	https://www.ncbi.nlm.nih.gov/pmc/articles/PMCS...		NaN				
NaN	NaN	NaN	document_parses/pdf_json/06ced00a5fc04215949aa...	document_parses/pmc_json/PMC59549.xml.json	https://www.ncbi.nlm.nih.gov/pmc/articles/PMCS...		NaN				
NaN	NaN	NaN	document_parses/pdf_json/348055649b6b8cf2b9a37...	document_parses/pmc_json/PMC59574.xml.json	https://www.ncbi.nlm.nih.gov/pmc/articles/PMCS...		NaN				

Ilustración 10 Ejemplo archivo CSV dataset

Este dataset se actualiza constantemente, y los datos con los que trabajo son los descargados de la plataforma el día **22 de marzo de 2022**, por lo que descargar nuevos datos implicará cambios en los resultados.

Es importante mencionar que solamente en los archivos JSON se encuentra el texto completo, y en el CSV anterior solamente encontramos el abstract. Esto dará pie a realizar diversas pruebas utilizando exclusivamente el abstract o el texto completo. Se enfatizará en la división entre dataset con texto completo y dataset sin el, ya que es una característica muy significativa con la que explorar.

El dataset metadata.csv cuenta con un total de 902.589 artículos y 50.369 revistas distintas, y la carpeta pdf_json contiene un total de 366.192 artículos de texto completo distintos. Además, contamos con 2.554.898 palabras únicas y 156.096.129 palabras totales en los campos de interés.

El dataset, como ya se ha hablado anteriormente, requiere de una correcta limpieza y preprocesamiento para ser útil de cara al entrenamiento y test. No podemos contar con el número máximo de clases debido a que muchas de ellas no tendrán manera de ser correctamente entrenadas, por lo que la limpieza es necesaria. El siguiente capítulo abordará este tema de la forma más completa posible. Todos los archivos de los que hemos hablado se pueden encontrar en el *apéndice 1.1*.

Capítulo 6

Procesado de CORD-19

En este apartado veremos algunas estadísticas sobre la información que tenemos, que de cara a la toma de decisiones para el preprocesado nos puede servir mucho, y realizaremos borrados y transformaciones hasta obtener el dataset listo para aplicarle las variaciones que veamos oportunas. Vamos a intentar lidiar con el problema de tener un gran número de revistas únicas, los cuales sin procesado asciende a 50.369.

Se crearán puntos de partida tanto para los dataset referentes al Machine Learning como los dataset referentes al Deep Learning. Todos los archivos de entrada y de salida se pueden encontrar en el *apéndice 1*, y los notebooks de Jupyter con los que he realizado el pipeline se encuentran en el *apéndice 2.1*.

La transformación de los datos incluye las siguientes partes diferenciadas:

1. Limpieza de datos
 - Borrado de columnas sin utilizar
 - Borrado de artículos duplicados
 - Borrado de nulos en el dataset
 - Unión de abstract y title
 - Borrado de revistas irrelevantes
2. [Deep Learning] Preprocesado de datos
 - Conversión de texto a minúsculas y borrado de caracteres no ASCII
 - Borrado de stopwords
 - División en conjunto de entrenamiento y conjunto de test
3. [Machine Learning] Preprocesado de datos
 - Conversión de texto a minúsculas y borrado de caracteres no ASCII
 - Borrado de stopwords
 - División en conjunto de entrenamiento y conjunto de test
 - Stemming del texto disponible
 - Tokenizado del texto aplicando Tf-Idf

Todo este proceso va acompañado de la extracción de características propias del dataset para una visualización más completa durante el proceso.

6.1 Limpieza de datos

Como punto de partida vamos a borrar todas las columnas que no vamos a utilizar. Nos quedamos exclusivamente con title, abstract, journal y sha. Posteriormente concatenamos title y abstract para así contar con exclusivamente tres columnas.

La columna sha es utilizada para tener un punto de conexión entre el dataset con los artículos y el dataset con los textos completos, ya que esta columna coincide y gracias a esta podremos recopilar la información.

6.1.1 Número de clases y relevancia

Según la información que nos brinda Kaggle se puede observar cómo hay un desbalanceo entre las clases disponibles. En la columna journal nos encontramos con un casi 1% de artículos que han sido publicados en la revista PLoS One, una famosa revista científica.

Para ver la información con más claridad, convertimos todo el contenido de la columna journal a minúsculas para evitar que haya varias revistas del mismo nombre. También borraremos los artículos duplicados, lo que nos deja con un total de 813.817 artículos distintos.

6.1.2 Borrado de nulos

Ahora vamos a pasar al borrado de artículos con contenido de las columnas abstract, title o journal nulo, ya que no nos interesan porque estamos perdiendo información que queremos.

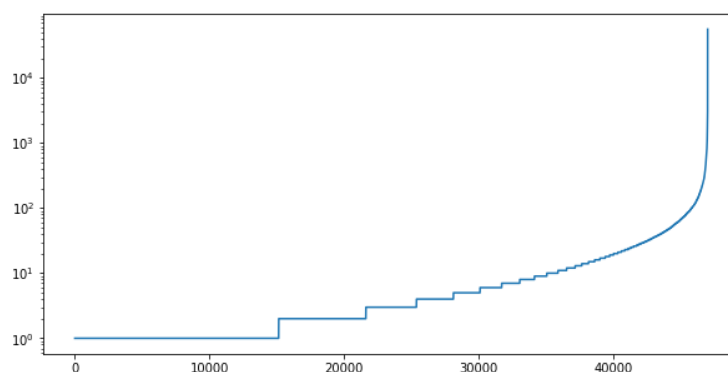


Ilustración 11 X: Revistas Y: Artículos (log2) antes de quitar nulos

En la *ilustración 11* podemos observar cómo hay una clase que predomina absolutamente sobre las demás. Esta es la clase “nan” que tiene 55.853 artículos, y no se tiene en cuenta como nulo para Pandas. Todos los artículos que coincidan con la clase anterior clase se borran: primero convierto todas las cadenas “nan” de title en “”, ya que esto no nos interesa perderlo (el título es

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

muy corto y borrar los “nan” en este campo haría que perdiéramos artículos que nos podrían ser de ayuda), borrar cuando encontremos un “nan” en las columnas journal o abstract. Una vez borrado vemos el conjunto ha disminuido a 503.613 artículos. Vemos la gráfica actualizada (*ilustración 12*):

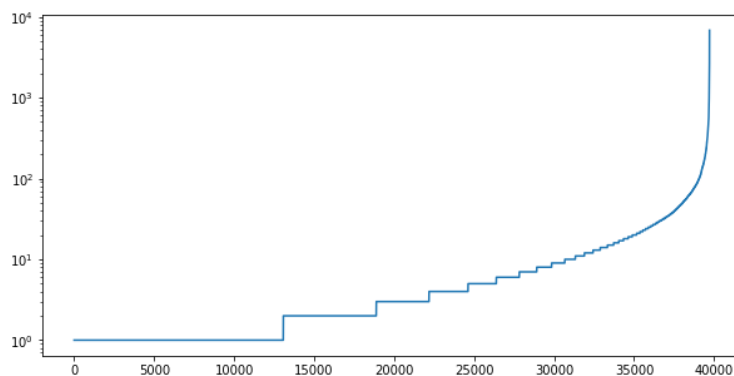


Ilustración 12 X: Revistas Y: Artículos (log2) tras quitar nulos

Puede observarse cómo hay una alta cantidad de revistas con pocos artículos, y un desbalanceo excesivo puede generar problemas a la hora de entrenar el modelo. En total hay 39.066 revistas con 100 muestras o menos, y solamente 26 con más de 1000. Las técnicas de undersampling y oversampling pueden ayudarnos a solucionar nuestro problema de desbalanceo de las clases.

Cuando sacamos el conteo de muestras máximo (*tabla 6*) podemos observar que la revista que más artículos tiene es PLoS One, con 6819. Esto va a significar que si intentamos recuperar varias posibles revistas para un usuario final al menos uno de ellos, en la mayoría de los casos, será PLoS One. Esto no significa que sea una mala predicción, pero se dará el fenómeno de overfitting, el cual va a privar a las otras revistas de ser entrenadas con una probabilidad mayor.

Tabla 6: TOP revistas por número de artículos

Revista	Num. artículos
plos one	6819
biorxiv	6485
int j environ res public health	5245
sci rep	3602
cureus	2683
bmj open	2485
front psychol	2163
viruses	2073
sustainability	1962
front immunol	1829

Finalmente concatenamos el abstract con el title para tener toda la información en un solo campo, ya que vamos a tratar con una bolsa de palabras y no con un texto.

6.1.3 Borrado de clases irrelevantes

Cuando hablamos de clases irrelevantes nos referimos a los que no tienen muestras suficientes como para que el modelo sea entrenado en base a sus características correctamente.

A menos muestras de la clase menos probabilidades de que este salga por encima de otro, así que nos interesa perder de vista muchos de las clases del problemas, y esto se da debido a que, como se ha explicado anteriormente, hay una cantidad enorme de clases que no llegan a 100 muestras, y dado que la diferencia es tan grande de cara a otras clases, es apropiado limpiar el dataset de clases que no cumplan el requisito.

Para el borrado de revistas primero se recupera el conteo de artículos para cada revista con la librería `collections.Counter`, la cual me devuelve un diccionario clave-valor con el nombre de la revista y el número de artículos, y creamos una función con los siguientes argumentos:

- `data`: El dataset correspondiente.
- `column`: La columna correspondiente.
- `value`: El valor por el que quieres comparar el número de revistas.

La función itera por todos los objetos clave-valor del diccionario, y para cada uno comprueba si el número de muestras es menor que el valor. Si esto es así todas las muestras que pertenezcan a esa revista se borran del dataset. He considerado que el mínimo es de 100 muestras para que las revistas cobren relevancia.

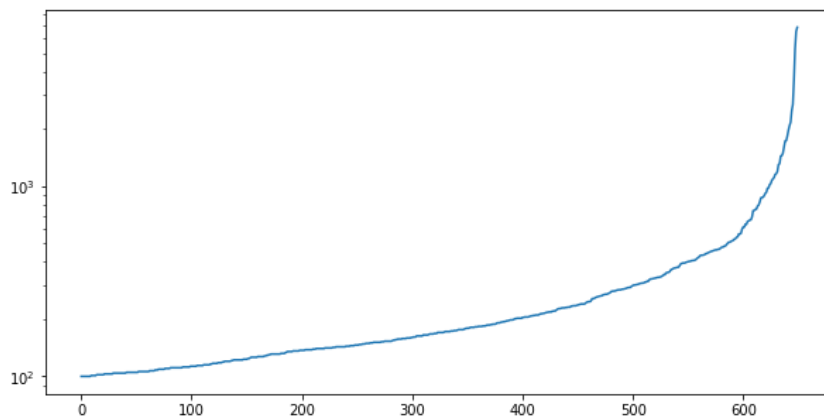


Ilustración 13 X: Revistas Y: Artículos (log2) tras ajustar revistas

En la *ilustración 13* observamos cómo el número de filas del dataset ha disminuido drásticamente. Nos quedamos con un total de 195.181 muestras distintas, y el número de clases disminuye a 650. Recordemos que habíamos partido con 902.589 muestras y 50.369, por lo que el sesgo ha sido grande.

Contamos con una bolsa de palabras muy amplia. Actualmente, sin haber comenzado el preprocesado, contamos con un total de 1.059.781 palabras únicas, y el total es 45.358.647.

6.2 Deep Learning - Preprocesado

Esta fase de la transformación comprende hasta el borrado de los stopwords, ya que para los dataset que serán entrenados por el método de Deep Learning que escojamos nos basta con un borrado de stopwords y un filtrado de texto simplemente.

6.2.1 Borrado de stopwords

Primero procedemos a pasar todo el texto a minúsculas y a quitar los caracteres no ASCII. Con esto el filtrado de palabras es más preciso y menos costoso, ya que palabras que difieren en el uso de las mayúsculas dejan de contar como palabras distintas.

En nuestro dataset nos encontramos con 141 palabras vacías distintas, las cuales aparecen un total de 15.553.667 veces. Las palabras más frecuentes son:

Tabla 7: TOP palabras más frecuentes

Palabra	Num. apariciones
the	2279870
of	1790829
and	1692114
in	1112481
to	979662
a	741123
with	534191
for	470616
covid19	408746
were	337874

En la *tabla 7* se puede observar que las primeras 8 posiciones son las palabras vacías, y la novena es covid19, debido a que el tema principal del dataset es el Covid-19 y sus variantes.

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

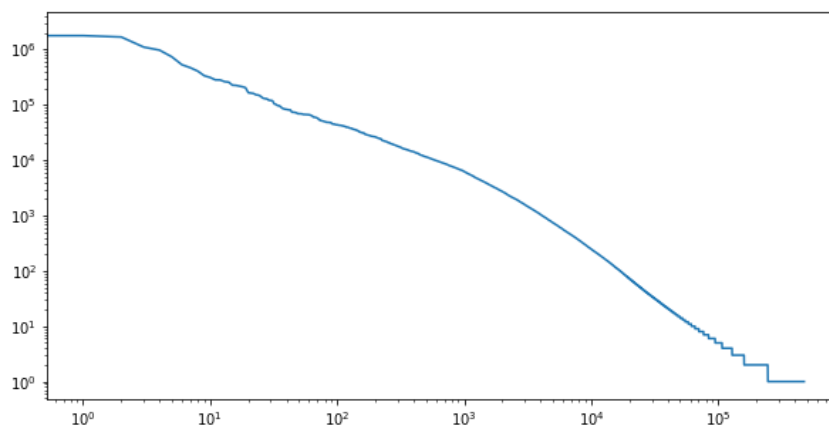


Ilustración 14 X palabras distintas, Y frecuencia de las palabras | Demostración Ley de Zipf

Observando la *ilustración 14* ya podemos hablar de un fenómeno muy interesante: la Ley de Zipf. Esta Ley nos dice que las frecuencias de las palabras son inversamente proporcionales a sus rangos. Aplicado a nuestro problema vemos que hay una cantidad muy grande de palabras con poca frecuencia, un conjunto de palabras medio con frecuencia media, y un conjunto muy pequeño de palabras con frecuencia muy alta. Para visualizar la característica línea diagonal que dibuja la Ley de Zipf los ejes X e Y tienen que estar en escala logarítmica, tal y como se ve en la ilustración. Hay un total de 469.223 palabras con menos de 1000 muestras y 34 palabras con más de 100.000, lo que confirma la ley.

Para borrar los stopwords nos valdremos de la librería NLTK, la cual contiene un lector de archivos corpus que usaremos para comparar nuestras cadenas de texto con las palabras del corpus “english” que nos proporciona la propia librería. Nos quedamos exclusivamente con las palabras relevantes.

La razón por la que utilizamos un set para recorrer los stopwords es porque la estructura de estos es una tabla hash, por lo que las comprobaciones de que un ítem se encuentre en el set son de complejidad $O(1)$, siendo el peor de los casos $O(n)$. En el caso de las listas la complejidad es $O(n)$, y en el peor de los casos tendremos que recorrer la lista entera n veces (en el caso de que ningún ítem se encuentre en la lista).

6.2.2 División train/test

Ahora, mediante la función de `train_test_split` brindada por SKLearn dividiremos el conjunto total en entrenamiento y test:

```
X_train, X_test, y_train, y_test = train_test_split(X_final, Y,  
test_size=0.25, stratify=Y)
```

Se divide el dataset en X_{train} (bolsa de palabras/abstract de entrenamiento), y_{train} (clases/journal de entrenamiento), X_{test} (bolsa de palabras/abstract de test), e y_{test} (clases/journal de test). El conjunto de test sirve para validar que el modelo ha entrenado correctamente, y nos servirá para calcular la precisión ($acc@k$).

El tamaño del conjunto de test es de un 25% del conjunto total, por lo que nos quedamos con un 75% de información para el conjunto de entrenamiento. Es un test estratificado, lo que significa que habrá un número proporcional de artículos de cada clase en el set de entrenamiento y de test, lo que pretende solucionar el desbalanceo de distribución de las clases. Finalmente contamos con un dataset de entrenamiento de 146.385 artículos, y uno de test de 48.796.

6.2.3 Palabras relevantes (extra)

Tenemos un total de 473.187 palabras relevantes únicas, y el total de palabras relevantes es 29.606.556. Aun así, muchas de estas no serán relevantes: las palabras que aparecen muy poco son palabras discriminatorias, ya que la aparición de estas puede ser decisivo a la hora de escoger la revista apropiada, y las que aparecen en todos o casi todos los artículos o revistas realmente aportan poco.

Tabla 8: TOP palabras poco frecuentes

Palabra	Num. apariciones
phagocytederived	1
oxygenase2	1
ho1co	1
67gallium	1
67gatransferrin	1
immunodepresssion	1
hogg1	1
hmth	1
psf911	1
feiidependent	1

Dentro del saco de palabras poco frecuentes hay palabras que por aparecer demasiado poco no llegan a ser discriminatorias para el conjunto de datos. En la *tabla 8* podemos ver algunas de las palabras menos frecuentes, y como estas (que aparecen una sola vez) hay un total de 228.234, y son potencialmente descartables.

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Volviendo a las palabras altamente frecuentes: en la *tabla 9* observamos cómo palabras como covid19, pandemic, infection, etc, se encuentran en el TOP. Estas palabras al ser tan frecuentes pueden no ser necesarias, ya que las hace poco discriminatorias.

Tabla 9: TOP palabras más frecuentes tras borrado de stopwords

Palabra	Num. apariciones
covid19	408746
patients	284208
study	166700
sarscov2	143585
health	133150
pandemic	130249
results	121827
disease	119222
infection	102262
data	97306

Los pasos que hemos realizado anteriormente nos han permitido eliminar los datos de texto que contienen información que no necesitamos como símbolos, fechas, horas y números, que pueden llegar a suponer hasta un 80% del documento. También contamos con un conjunto de entrenamiento (146.385 artículos) y un conjunto de test (48.796 artículos).

6.3 Machine Learning - Preprocesado

Este apartado puede verse como una ampliación del anterior, ya que los pasos realizados hasta ahora se vuelven a repetir de manera idéntica. Adicionalmente se realizará un stemming del conjunto de entrenamiento y test, así como un vectorizado tf-idf para el de entrenamiento exclusivamente, y se guardarán por separado.

6.3.1 Paso del abstract por el stemmer

Utilizaremos el Stemmer que nos proporciona NLTK. La función proporcionada tiene tres modos distintos, los cuales son tres versiones del algoritmo que se emplea para sacar las raíces [39]:

- ORIGINAL_ALGORITHM: Algoritmo original de Stemmer.
- MARTIN_EXTENSIONS: Mejoras en la implementación hechas por Martin Porter, sacadas de su propia web (<https://www.tartarus.org/~martin/PorterStemmer/>). No habrán futuras mejoras.

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

- NLTK_EXTENSIONS: Opción por defecto. MARTIN_EXTENSIONS mejorada por contribuidores de NLTK.

Nosotros utilizamos la opción por defecto, ya que nos da el mejor stemming, y es lo que buscamos. Creamos una función que nos lo aplique para cada uno de los artículos con los que contamos. Esta función lo que hace es llamar a PorterStemmer, tokenizar el texto del abstract y recorrer cada palabra individualmente, comprobando que la palabra carácter por carácter sea completamente alfabética, y añadiendo a una lista auxiliar todas las palabras habiendo sido pasadas por el PorterStemmer(). Finalmente aplicamos esta función a todas las filas del abstract, tanto en entrenamiento como en test.

Con este proceso reducimos el tamaño del conjunto de palabras de entrenamiento de 22.202.780 a 13.800.377, y un total de 210.394 palabras únicas, por lo que hemos reducido tanto la calidad de los datos como el tamaño de almacenamiento que ocupan.

Finalmente, hasta esta parte, contamos con 650 clases distintas, y los siguientes datasets de entrenamiento y test:

- Dataset de entrenamiento (Deep Learning):
 - 146.385 artículos sin texto completo.
 - 399.928 palabras únicas.
 - 22.202.780 palabras totales.
- Dataset de entrenamiento (Machine Learning):
 - 146.385 artículos sin texto completo.
 - Se realiza stemming de la bolsa de palabras.
 - 210.394 palabras únicas.
 - 13.800.377 palabras totales.
- Dataset de test (Deep Learning):
 - 48.796 artículos sin texto completo.
 - Test estratificado.
 - 203.495 palabras únicas.
 - 7.403.776 palabras totales.
- Dataset de test (Machine Learning):
 - 48.796 artículos sin texto completo.
 - Test estratificado.
 - Se realiza stemming de la bolsa de palabras.
 - 110.326 palabras únicas.

- 4.602.709 palabras totales.

6.3.2 Vectorizado Tf-Idf

Vamos a trabajar específicamente con `TfidfVectorizer`, función que nos proporciona la librería `SKLearn`. Llamamos a la función con los argumentos `min_df` y `max_df`. El `min_df` es el mínimo de veces que el término tiene que aparecer en la colección para que entre en el conjunto de tokens, y el `max_df` es el máximo de ocurrencias del término en la colección para que se incluya en el conjunto de tokens.

```
tfidf = TfidfVectorizer(min_df=20, max_df=0.9)
X_final = tfidf.fit_transform(X)
```

Empezamos a contar a partir de las 20 apariciones en el total de documentos, y que aparezca en como máximo el 90% de los documentos. Considero que para la cantidad de palabras que tenemos en nuestro dataset son parámetros adecuados. Tras todo el preprocesado el vectorizado Tf-Idf nos deja con un total de 16.686 atributos distintos.

Este paso se encuentra en los notebooks del *apéndice 3*, ya que se realiza inmediatamente antes de ejecutar las pruebas.

Capítulo 7

Datasets y variaciones

En este capítulo veremos los datasets con los que vamos a trabajar en esta primera parte, la cual no incluye el texto completo, y tendrá un capítulo dedicado a ello. Se explicarán uno por uno cada variación distinta, y partimos de los procesamientos anteriores. Las variaciones escogidas son las siguientes:

- Dataset 1: Dataset preprocesado (Machine Learning)
- Dataset 2: Uso de unigramas y bigramas
- Dataset 3: Undersampling
- Dataset 4: Oversampling
- Dataset 5: Dataset preprocesado (Deep Learning)

El dataset 1 y el 5 han sido explicados anteriormente, en el capítulo 2, por lo que no voy a parar en ellos debido a que no hay más que decir. El primer dataset contiene un total de 16.686 atributos tras el vectorizado, y el último cuenta con un total de 390.053 atributos.

Recordemos también que todas las variaciones se realizan sobre el conjunto de datos de entrenamiento. Anteriormente realizamos un split en el dataset completo que dividía conjunto de entrenamiento y de test. El conjunto de test es común a todos los dataset y no sufre ningún cambio para que la comparación no sea dispar.

Los diversos archivos con los que trabajaremos se encuentran en el *apéndice 1*.

7.1 Dataset unigramas y bigramas

Para el segundo dataset analizaremos cómo influye el uso de n-gramas en la tokenización, y en concreto el uso de unigramas y bigramas.

Utilizaremos TfidfVectorizer para aplicar los n-gramas, y estos son controlados por el parámetro ngram_range, en el cual especificaremos el máximo n-gram y el mínimo n-gram. Por ejemplo, si ponemos ngram_range=(1,3) significa que se aceptarán unigramas, bigramas y trigramas. Los parámetros de TfidfVectorizer serán de la siguiente manera:

```
tfidf = TfidfVectorizer(ngram_range=(1,2), min_df=20, max_df=0.9)
```

Esto forzará a que solamente se tengan en cuenta los unigramas y bigramas en la bolsa de palabras. Contamos con 110.431 atributos frente a los 16.686 del dataset inicial, lo que es un salto bastante significativo.

7.2 Dataset Undersampling

A este dataset se le aplicará la técnica de undersampling para balancear las clases existentes en los datos. Esta técnica tiene como contra la pérdida de datos, pero aplicada correctamente nos puede servir para mejorar la predicción.

Vamos a utilizar la librería ImbLearn, que nos provee de múltiples técnicas para atajar nuestro problema de desbalanceo de clases. La mayoría de estas técnicas están basadas en el método K-Nearest Neighbors. Los métodos más relevantes son:

- CondensedNearestNeighbour: Borra del dataset los artículos que no se clasifiquen correctamente usando el algoritmo KNN con $K=1$. Sensible al ruido. Creado por Peter Hart en 1968 [40].
- EditedNearestNeighbour: Aplica KNN y borra los artículos que no se parezcan lo suficiente a su vecino [41].
- RepeatedNearestNeighbour: Como el anterior pero repetido n iteraciones (la cantidad de sesgo será mayor) Su primera aparición fue en 1976 por Ivan Tomek [42].
- AllKNN: Como el anterior, pero por cada iteración aumenta en 1 el número de vecinos de KNN [42].
- TomekLinks: Algoritmo derivado de CondensedNearestNeighbour que borra pares de diferentes clases, A y B, y que cumplen que A es el NN de B y viceversa. Borra la instancia que pertenezca a la clase grande. Por Ivan Tomek en 1976 [43].
- OneSidedSelection: Mezcla TomekLinks y CondensedNearestNeighbour. Primero aplica TomekLinks para borrar artículos que generen ruido, y luego se aplica 1NN.
- RandomUnderSampler: Borrado aleatorio.

Durante la etapa de pruebas realizaremos una comparación entre métodos para escoger el que mejor funcione y se adapte a nuestros datos.

7.3 Dataset Oversampling

A este dataset se le aplica la técnica de oversampling para balancear las clases existentes en los datos. Esta técnica se basa en añadir más artículos a las revistas con menos artículos, para de esta manera balancear todas las clases del problema.

Vamos a utilizar la librería ImbLearn, que nos provee de múltiples técnicas para atajar nuestro problema de desbalanceo de clases. Definimos los métodos más importantes:

- RandomOverSampler: Clona muestras del dataset para las clases pequeñas hasta alcanzar a la clase mayor.
- SMOTE: Selecciona artículos cercanos en espacio y dibuja una línea entre los ejemplos. A través de esa línea se escribirá un nuevo artículo en el espacio. Es un KNN a la hora de escoger el vecino más cercano. Por Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, y W Philip Kegelmeyer en 2002 [44].
- ADASYN: Modificación de SMOTE el cual añade muchos artículos al espacio en zonas donde la densidad de artículos es baja, y muy pocos en zonas de densidad alta. Por Haibo He, Yang Bai, Edwardo A Garcia, y Shutao Li en 2008 [45].

Durante la etapa de pruebas realizaremos una comparación entre métodos para escoger el que mejor funcione y se adapte a nuestros datos.

Capítulo 8

Resultados obtenidos

En este capítulo comenzamos a realizar las pruebas entre datasets distintos con varios métodos, y exploraremos tanto Machine Learning como Deep Learning. Dividiremos las pruebas en varias partes: una por cada dataset distinto, y la última parte será dedicada exclusivamente al Deep Learning. Para ponernos en contexto, los dataset que vamos a explorar serán los siguientes:

- Dataset 1: Dataset preprocesado (Machine Learning)
- Dataset 2: Uso de unigramas y bigramas
- Dataset 3: Undersampling
- Dataset 4: Oversampling
- Dataset 5: Dataset preprocesado (Deep Learning)

Los resultados se pueden encontrar en el *apéndice 3.2*.

8.1 Métricas empleadas

Para este problema el abanico de métricas es muy reducido, debido a que la cantidad de datos y clases será demasiado alta como para realizar visualizaciones en gráficas. Veremos un total de cuatro: tiempo de entrenamiento/predicción, Mean Reciprocal Rank, Accuracy Score y Top K Accuracy Score.

8.1.1 Tiempo de entrenamiento/predicción

Se compara el tiempo de entrenamiento para cada método abordado. En algunos casos se tiene en cuenta el tiempo de predicción y no el de entrenamiento debido a que hay métodos que consumen tiempo de ejecución pero son de entrenamiento inmediato (KNN).

8.1.2 Mean Reciprocal Rank

También llamado MRR. Métrica que nos da un score en base a las posiciones en las que se encuentre la clase correcta dentro de nuestra predicción de n ítems posibles (sistema de recomendación). A más alta posición en el ranking en términos generales, mejor score. Dependiente del número de ítems que queramos recuperar.

A pesar de que no necesitemos recuperar una sola revista es correcto analizar en qué medida nuestras predicciones son correctas a la primera, o casi correctas. Se calcula con la siguiente fórmula [46]:

$$MRR = \frac{1}{|Q|} * \sum_{i=1}^{|Q|} \frac{1}{Rank_i}$$

Siendo Q el número de artículos totales, y se realiza la sumatoria de 1 partido la posición de la revista i en la predicción. Si el ítem relevante se encuentra en la primera posición del ranking, el MRR será 1, si es siempre el segundo lugar será 1/2, si es el tercer lugar será 1/3, y así sucesivamente. Si ninguna respuesta es correcta, el resultado será 0. Fue inventado por Nick Craswell en 2009 [47].

8.1.3 Accuracy Score

Es la fracción de predicciones que nuestro método ha acertado. La fórmula general es la siguiente:

$$acc(x, y) = \frac{1}{Q} * \sum_{i=1}^Q I(y_i = x_i)$$

Siendo Q el número de artículos, y la revista correspondiente, x la predicción, y la función $I(y=x)$ es la función indicador, la cual es 1 si la predicción es correcta, y 0 si la predicción es incorrecta [48].

8.1.4 Top K Accuracy Score

Variación de la métrica anterior, llevada al problema de recuperación de K ítems en un ranking. Su fórmula es:

$$acc(X, y) = \frac{1}{Q} * \sum_{i=1}^Q \sum_{j=1}^k I(y_i = X_{i,j})$$

Siendo Q el número de artículos, y la revista correspondiente, X el conjunto de predicciones, y_i la revista correcta, $X_{i,j}$ una matriz la cual contiene en cada fila las n predicciones para cada caso, y la función $I(y=x)$ es la función indicador, la cual ha sido comentada anteriormente.

En este caso se toma la predicción correcta cuando el ítem que corresponde se encuentra entre las K primeras predicciones del método utilizado [48].

8.2 Hardware utilizado

Para realizar las distintas pruebas he utilizado dos sistemas, los cuales voy a citar a continuación junto con las especificaciones más relevantes:

- Mi propia máquina:
 - CPU: Intel Core i5 12600K | 6P+4E cores | 16 hilos | 4.9GHz
 - GPU: RTX 3060 | 152 Tensor cores | 12GB VRAM GDDR6 | PCI Express x16 4.0
 - RAM: 16GB DDR4 | 2x8GB CL16 | 3200MHz
 - Sistema Operativo: Windows 11 Pro
- SSH bahia.ugr.es:
 - CPU: 2x Intel Xeon E5-2630 v4 | 10 cores | 20 hilos | 3.10GHz
 - GPU: ASPEED rev. 30 (no he encontrado más información)
 - RAM: 192GB (no he encontrado más información)
 - Sistema Operativo: Fedora 34 Server Edition

También exploré la posibilidad de utilizar Google Colab, pero su límite de 12GB de RAM me impedía realizar la mayoría de las pruebas, así que decidí usar mi máquina y el servidor que se me proporcionó.

Usaré mi máquina con los métodos Multinomial Naive Bayes, Complement Naive Bayes y K-Nearest Neighbours (aunque este último en el dataset 4 lo ejecutaré con el servidor), y en bahia.ugr.es ejecutaré Random Forest y Logistic Regression.

8.3 Resultados del dataset inicial

El primer dataset con el que vamos a realizar pruebas es el dataset limpio. Recordemos que a este dataset se le pasaba un vectorizado tf-idf sin n-gramas, con mínimo de 20 apariciones por palabra y el máximo son palabras que estuvieran en más del 90% de los documentos. Tras el tokenizado contamos con 16.686 atributos distintos.

8.3.1 Multinomial Naive Bayes

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Dado que MultinomialNB es un algoritmo que entrena y predice rápidamente podemos realizar cambios en los parámetros con más facilidad. La **primera prueba** será con los parámetros por defecto: suavizado de Laplace en 1.0 y sin probabilidad a priori.

Tabla 10: Scores MultinomialNB prueba 1

acc@k	Accuracy score
acc@1	0.09812279695056972
acc@5	0.2530125420116403
acc@10	0.3342896958767112

Viendo los resultados en la *tabla 10* es fácil darse cuenta de que la predicción es de muy mala calidad. A partir de la acc@10 podemos hablar de un score de 0.33, pero eso es bajo contando con que recuperamos 10 revistas posibles. Esta predicción se debe muy posiblemente al suavizado de Laplace que se le aplica, por lo que en pruebas posteriores veremos cómo cambia la predicción en base a eso.

No vamos a trabajar con prioridades previas, ya que no hay información ninguna sobre ello en el dataset original, y no hay ningún criterio por el cual pueda añadir prioridades a cada clase, además de que el número de clases es bastante alto.

La **segunda prueba** consiste en suprimir el suavizado del método ($\alpha=0$). La función no nos permite tomar $\alpha=0$, por lo que nos pone el valor $\alpha=0.0000000001$. Los resultados han mejorado bastante, aunque están lejos de ser perfectos:

Tabla 11: Scores MultinomialNB prueba 2

acc@k	Accuracy score
acc@1	0.2526026723501926
acc@5	0.4685220100008197
acc@10	0.5754570046725141

En la *tabla 11* vemos cómo la predicción realmente ha pasado de menos de 0.1 a un 0.25 para acc@1, y de hecho se puede mejorar más ajustando el parámetro de α correctamente. Tras realizar numerosas pruebas, he visto que el mejor valor de α ronda 0.004, es decir, un suavizado muy leve, pero veremos cómo influye en la **tercera prueba**:

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Tabla 12: Scores MultinomialNB prueba 3

acc@k	Accuracy score
acc@1	0.2952905975899664
acc@5	0.5593901139437659
acc@10	0.6749118780227887

En la *tabla 12* la acc@1 aumenta en casi 0.05 sobre el valor anterior, y recuperando 10 revistas aumenta hasta 0.67, lo que para el problema que abordamos está muy bien. He calculado el Mean Reciprocal Rank, y para esta última prueba lo he calculado. Los valores son los siguientes:

Tabla 13: MRR MultinomialNB prueba 3

MRR@K	Score
MRR@5	0.39299293657951656
MRR@10	0.40837721541059285

En la *tabla 13* como vemos no aumenta mucho la calidad de la predicción aumentando el número de revistas recuperadas, pero es normal, ya que solamente estamos aumentando la precisión individual de cada predicción. De todas maneras, un MRR de 0.4 no está del todo mal.

Considero que, para nuestro problema, el cual es solventar un problema de recuperación de varias revistas, el resultado encontrado con Multinomial Naive Bayes está muy bien. Otro plus es que el tiempo de entrenamiento es extremadamente pequeño (4.9s), lo que lo hace un método rápido y potente.

8.3.2 K-Nearest Neighbours

Anteriormente hemos visto este método con variaciones aplicado a distintos métodos de undersampling y oversampling. Ahora vamos a verlo en acción para un problema de clasificación.

La **primera prueba**, como ya se ha hecho en el primer método visto, será con los parámetros por defecto. Estos son: 5NN, pesos uniformes entre vecinos cercanos (no dependientes de la distancia), el algoritmo se selecciona por defecto (exploraremos las opciones más adelante), y métrica de distancia Minkowski. Para acelerar el proceso se ha activado el multiprocesamiento (n_jobs=-1):

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Tabla 2: Scores KNN prueba 1

acc@k	Tiempo predicción	Accuracy score
acc@1	144.5s	0.21487417001393558
acc@5		0.49372899417985083
acc@10		0.49506107057955573

Veamos la *tabla 14*: una acc@10 de casi 0.5 no está mal. El problema que tiene KNN es que el tiempo se lo lleva la predicción, no el entrenamiento, ya que se realiza un cálculo de distancias entre puntos del espacio en tiempo de ejecución. Se podría realizar un cambio de la estructura de datos que almacena los puntos del dataset, pero el problema es que nuestra cantidad de datos es tan grande que las matrices son dispersas (sparse), es decir, para un artículo tienes un vector que tiene una dimension igual al número de terminos en el vocabulario, en muchos casos tienen peso cero (el termino no aparece en el documento). Los algoritmos que íbamos a ver (KD Tree y Ball Tree) se basan en matrices densas, y nosotros no tenemos, por lo que vamos a emplear Brute Force, el método de cálculo de distancias estándar (método muy lento).

Como **segunda prueba** voy a probar con varios números de vecinos: 1NN, 25NN y 50NN, y haré la comparación a ver cuál es más rápido en predecir. Los resultados son:

Tabla 15: Scores KNN prueba 2

KNN	acc@K	Accuracy score	Tiempo de pred.
1NN	acc@1	0.3161119763915075	106.9s
	acc@5	0.3170546766128371	
	acc@10	0.31967784244610215	
25NN	acc@1	0.2445487335027461	148.9s
	acc@5	0.47126813673251905	
	acc@10	0.5855602918271989	
50NN	acc@1	0.2473358472005902	145.4s
	acc@5	0.49213050250020496	
	acc@10	0.5991269776211164	

Fijándonos en la *tabla 15*, el tiempo de predicción en los tres ejemplos han sido de 1m 30s aproximadamente, así que por ese lado no hay ninguna razón por la que escoger uno u otro.

Por otra parte, vemos que la calidad de la predicción apenas varía para 1NN. Esto se puede deber a que los datos están divididos en grupos de manera medianamente uniforme, pero posiblemente el entrenamiento no sea bueno debido a la calidad de los datos. Esto lo voy a mostrar con unos dibujos.

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Supongamos que tenemos un problema de tres clases: rojo, verde y azul, y queremos clasificar con 1NN. En el ejemplo siguiente la clase correcta es la roja.

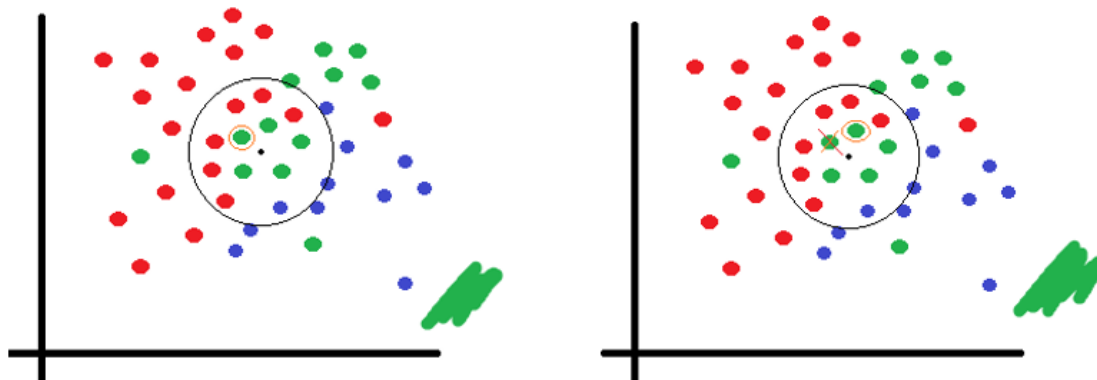


Ilustración 15 Explicación KNN 1

Como vemos en la *ilustración 15*, al principio clasificamos para el más cercano de los vecinos, y el ranking sería el segundo candidato, tercero, etc. Si recuperamos los 5 más probables en este ejemplo vemos cómo clasificaría igual que recuperando 1, y eso puede estar pasando: al tener una cantidad de datos muy grande puede estar pasando que la clase a la que pertenece está un poco más lejos y no clasifica correctamente.

Para 25NN y 50NN la razón por la que está aumentando la accuracy conforme aumenta el número de revistas recuperadas es porque su funcionamiento es por el número de artículos de cada revista que hay en el radio que forma el artículo más lejano del top K elementos. En el ejemplo de abajo (*ilustración 16*) la clase correcta es la roja.

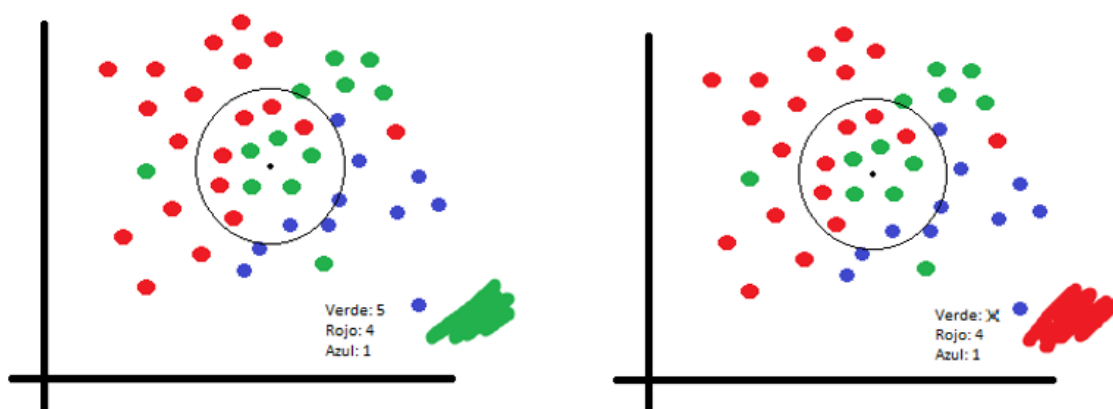


Ilustración 16 Explicación KNN 2

Vemos cómo para $acc@1$ tomaría como predicción verde, lo que es erróneo, pero si hacemos la $acc@2$ tomaría el rojo también, lo que sí sería correcto para nuestro problema. Esta

es la razón por la que cuando tomamos valores más altos de K la predicción mejora para nuestro ranking: porque hay más artículos que tomar en el conteo.

El MRR es muy similar en 1NN, 25NN y 50NN. Se encuentra rondando entre 0.31 y 0.35. No vale la pena hacer una tabla, ya que no nos da información muy relevante dado que es un método lento.

Este método es muy bueno cuando no hay una similitud muy grande entre clases y se encuentran en el espacio muy separadas las unas de las otras. Cuando no están divididos en grupos con poco ruido este algoritmo es eficaz, pero en este caso los datos están muy mezclados como para realizar una buena predicción.

8.3.3 Random Forest

Para las pruebas con este método vamos a trabajar con los tres parámetros siguientes:

- `n_estimators`: Número de árboles que se busca crear para realizar los sondeos. A mayor valor mayor necesidad de prestaciones más altas.
- `max_features`: Número de características a explorar en cada árbol individual. Se puede especificar dando un número exacto como tres opciones predefinidas:
 - None: `max_features = n_features`
 - `sqrt/auto`: `max_features = sqrt(max_features)`
 - `log2`: `max_features = log2(max_features)`
- `criterion`: Algoritmo utilizado para decidir qué splits realizar. Puede especificarse el criterio Gini, Entropy o Log-loss.
- `max_depth`: Profundidad máxima de los árboles creados. Dado que los recursos con mi máquina personal son limitados he podido crear los árboles con una profundidad de 200.

La **primera prueba** la realizaremos con el parámetro `n_estimators`. Voy a utilizar los parámetros por defecto (10 estimadores), 50 y 150 estimadores. Los resultados son los siguientes:

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Tabla 3: Scores RandomForest prueba 1

n_estimators	acc@k	Accuracy score	MRR@10	Tiempo
10	acc@1	0.30006557914583165	0.35351470553793396	61.8s
	acc@5	0.41638658906467746		
	acc@10	0.4900196737437495		
50	acc@1	0.3639027789163046	0.44249988940025553	182.5s
	acc@5	0.5472784654479875		
	acc@10	0.6123452742028035		
150	acc@1	0.37843265841462415	0.47087735366027217	1002.8s
	acc@5	0.5941470612345274		
	acc@10	0.6753422411673088		

Veamos la *tabla 16*: estos resultados indican que, a mayor número de árboles para nuestro problema, mayor es la calidad de la predicción. Esto tiene un umbral el cual al ser alcanzado la calidad disminuirá, pero hemos conseguido mejorar la calidad de la predicción hasta alcanzar un score de 0.67 al recuperar 10 artículos. Esto está mucho mejor.

Viendo los tiempos de la prueba de 150 estimadores podemos observar que la espera no es larga. En cuanto a MRR, se aprecia una tendencia al alza a mayor número de estimadores. Un 0.47 de score MRR para nuestro problema, contando con que hay un acc@10 de 0.67 es un buen resultado. Dado que 150 estimadores da buenas predicciones continuaré las pruebas con esta cifra.

La **segunda prueba** va a ser probando distintas configuraciones con max_features. Para mi problema el parámetro None no me ha sido posible testarlo, ya que en mi máquina no he podido entrenarlo debido a la cantidad de memoria que requiere, y en el servidor tras mucho tiempo con el contenedor levantado se me cancelaba la operación sin ningún código de error, así que los ajustes del parámetro serán log2 y max_features=16000, ya que se acerca al total de atributos de nuestro dataset, que es de 16.686. No necesitamos probar el ajuste sqrt, ya que sqrt es el parámetro por defecto y ya lo hemos visto anteriormente:

Tabla 17: Scores RandomForest prueba 2

max_features	acc@k	Accuracy score	Tiempo
Log2	acc@1	0.3325477498155587	355.9s
	acc@5	0.526866956307894	
	acc@10	0.6104803672432166	
16000	acc@1	0.37986720222969095	9373.9s
	acc@5	0.5837363718337568	
	acc@10	0.6569595868513812	

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Los resultados de la *tabla 17* han sido bastante inesperados. Antes de realizar la prueba pensaba que el número de atributos iba a influir en el resultado, y la calidad iba a ser directamente proporcional al número de atributos que tengamos en cuenta. Se han utilizado 129 atributos para sqrt y 14 atributos para log2.

Sorprendentemente el caso de mayor número de atributos ha dado peores resultados que utilizando 129. Esto puede deberse a que tenemos una profundidad máxima de 200, por lo que realmente no van a aprovecharse los 16.000 atributos disponibles. Esta razón es probablemente por la cual estamos teniendo una mayor calidad de predicción con menos atributos, pero si aumentáramos la profundidad máxima probablemente tendríamos una calidad mayor con 16.000 atributos. Naturalmente nos vamos a quedar con `max_features="sqrt"` para la prueba final.

Finalmente, para la **tercera prueba** vamos a ver el parámetro criterion. La función Gini es la que asignada por defecto, por lo que las pruebas serán con Entropy y Log-Loss:

Tabla 18: Scores RandomForest prueba 3

criterion	acc@k	Accuracy score	Tiempo
entropy	acc@1	0.30022952701041067	571.6s
	acc@5	0.47417821132879745	
	acc@10	0.5486105418476924	
log_loss	acc@1	0.30014755307812113	575.5
	acc@5	0.4712066562833019	
	acc@10	0.545536519386835	

En la *tabla 18* concluimos cómo el criterio de split Gini es el mejor que se puede utilizar, el que mejor clasifica, y el más rápido de los tres criterios que hay disponibles. No voy a hablar del MRR porque, naturalmente, es mucho menor para estos casos.

Concluimos que para nuestro conjunto de datos y nuestras especificaciones la mejor opción que tenemos es la de tener un total de 150 estimadores, con 129 atributos a tener en cuenta, y usando el criterio Gini para realizar los split.

8.3.4 Multinomial Logistic Regression

Para este método los parámetros con los que vamos a trabajar son los siguientes:

- solver: Permite elegir el algoritmo de reducción de la función de coste, y nos sirve para calcular los coeficientes óptimos para nuestro problema. Los algoritmos que vamos a ver son los siguientes [49]:

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

- Liblinear: Usa el algoritmo “Coordinate Descent” [50] para resolver la optimización de la función de coste. Este algoritmo funciona minimizando de una en una cada variable de la función. Tiende a ser lento, pero es muy eficiente para grandes datasets.
- Lbfgs: Extensión del método de Newton, el cual ha ganado mucha popularidad debido a que utiliza poca memoria y es potente. Es el método por defecto de SKLearn.
- SAGA: Es el más utilizado para el caso de grandes volúmenes de datos dispersos. Variación de Stochastic Average Gradient, que añade soporte para penalty L1.
- multi_class: Indica qué tipo de Logistic Regression utilizaremos de las disponibles: Multinomial u One VS Rest (utiliza solver=“liblinear”).
- penalty: Ajusta la penalización con varias configuraciones distintas. Necesario para la regularización, que consiste en realizar ajustes para evitar sobreentrenamiento penalizando coeficientes de regresión altos. Las opciones son [51]:
 - None: Ningún tipo de penalización.
 - L1: Penaliza la suma de los valores absolutos de los coeficientes. Los valores más cercanos a cero se ponen a cero, ya que no se consideran importantes.
 - L2: No se eliminan coeficientes, pero los reduce en la misma medida. Penaliza la suma de los coeficientes al cuadrado. Los coeficientes más cercanos a cero no desaparecen, pero su influencia es menor.
 - Elastic net: Combina L1 y L2 en uno. Aplica los dos métodos de forma que algunos coeficientes pasan a cero y otros se reducen. Debemos poner el argumento l1_ratio a 0.5, ya que así habrá una influencia intermedia entre L1 y L2 ($0 \leq l1_ratio \leq 1$).

Para Logistic Regression no trabajaremos con los parámetros por defecto, ya que el parámetro solver es importante ajustarlo dependiendo del penalty utilizado (debido a que no todos los algoritmos son compatibles). Especificaré qué algoritmo del parámetro solver estoy utilizando en todo momento. De todas maneras cabe mencionar que los parámetros por defecto son los siguientes:

- max_iter=100: Máximo de iteraciones de los solvers hasta que converjan.
- multi_class=“auto”: Auto es multinomial, a no ser que solver sea liblinear.
- solver=“lbfgs”: Explicado anteriormente.
- penalty=“l2”: Explicado anteriormente.
- l1_ratio=None: No sirve porque no estamos usando Elastic Net.

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Para la **primera prueba** realizamos una comparación entre utilizar One VS Rest y Multinomial (con solver lbfgs y saga) para ver cuál de los dos métodos es mejor usar, dado que son las dos alternativas que tomar frente al problema multiclase. Como se ha venido haciendo, utilizaremos `n_jobs=-1` para multiprocessing:

Tabla 19: Scores LogisticRegression prueba 1

multi_class	Tiempo	solver	acc@k	Accuracy score	MRR@10
Multinomial	1880.4s	lbfgs	acc@1	0.29100745962783836	0.40881459675983367
			acc@5	0.5670751700959095	
			acc@10	0.6881711615706205	
Multinomial	832s	saga	acc@1	0.2910894335601279	0.40919813392199994
			acc@5	0.5688171161570621	
			acc@10	0.6916345602098533	
OVR	4379.4s	liblinear	acc@1	0.28754406098860563	0.4059261704381339
			acc@5	0.5655176653824084	
			acc@10	0.6895647184195426	

En la *tabla 19* vemos cómo con OVR alcanzamos un score recuperando 10 documentos de casi 0.69, el cual es mejor que Multinomial por defecto, pero vemos que al usar el solver SAGA no solo tenemos mejor `acc@10`, sino que el tiempo es el más bajo de todos. En términos de `MRR@10` hablamos de un casi 0.41, lo cual no está nada mal. El solver SAGA es el escogido para las pruebas siguientes.

Para la **segunda prueba**, comprobaremos qué penalización es la mejor para nuestro problema. Al haber escogido el solver SAGA, tenemos disponibles todos los tipos de `penalty` que el método soporta (`none`, `l1`, `l2` y `elasticnet`).

La prueba anterior fue con `penalty="l2"`, ya que es el valor por defecto, por lo que realizaremos una tabla con `l1`, `elasticnet` y `none`. Para `elasticnet` vamos a ajustar el parámetro `l1_ratio` para que los dos tipos de `penalty` tengan la misma influencia (`l1_ratio=0.5`). Los resultados son los siguientes:

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Tabla 20: Scores+MRR LogisticRegression prueba 2

multi_class	Tiempo	acc@k	Accuracy score	MRR@10
L1	6806s	acc@1	0.3156611197639151	0.42770599366069206
		acc@5	0.5791253381424707	
		acc@10	0.6870235265185671	
Elastic Net	4297.4s	acc@1	0.28713419132715795	0.3998180390143993
		acc@5	0.5510697598163784	
		acc@10	0.6670833674891384	
None	3930.6	acc@1	0.39781949340109846	0.5055689495105653
		acc@5	0.6484342978932699	
		acc@10	0.7456348881055824	

En la *tabla 20* vemos cómo sin ningún tipo de penalty obtenemos una acc@10 de 0.74, un buen score para nuestro problema. Es actualmente el valor más alto de todos y lo acompaña un MRR@10 de 0.5. Concluimos así que el mejor ajuste de parámetros que hemos encontrado es el de multi_class="multinomial", penalty="none" y solver="saga".

8.3.5 Complement Naive Bayes

Los argumentos de este método son los mismos que para MultinomialNB, sólo que añade un parámetro de normalización de pesos, con el cual no trabajaremos.

Probaremos con los mismos parámetros con los que probamos en MultinomialNB para poder observar si hay algún cambio al respecto. Los parámetros por defecto son Alpha=1.0 (suavizado), sin probabilidad a priori y sin normalización de pesos. Los resultados son los siguientes:

Tabla 21: Scores ComplementNB

alpha	acc@k	Accuracy score	MRR@10
1.0	acc@1	0.19581523075661939	0.31067547821382524
	acc@5	0.46481268956471844	
	acc@10	0.6091687843265842	
0	acc@1	0.203766702188704	0.3185265885051976
	acc@5	0.472764160996803	
	acc@10	0.6153783096975162	
0.004	acc@1	0.20335683252725634	0.3183933402035542
	acc@5	0.47280514796294776	
	acc@10	0.6158086728420362	

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

En la *tabla 21* vemos que, a pesar de que este método es mejor para datos desbalanceados, los scores son peores. La mejoría es muy leve ajustando el parámetro Alpha, y con el parámetro por defecto la predicción es superior a la de Multinomial Naive Bayes.

8.3.6 Resumen de los métodos utilizados

A modo de resumen veremos en una tabla las mejores configuraciones para cada método, junto con la conclusión y algunos comentarios finales. En términos generales para nuestro problema no ha habido una mala predicción y hemos conseguido ajustar los parámetros eficientemente (*tabla 22*).

Tabla 22: Resumen métodos Score+MRR dataset 1

Método	Parámetros	acc@k	Accuracy score	MRR@10
Multinomial Naive Bayes	alpha=0.004	acc@1	0.2952905975899664	0.40837721541059285
		acc@5	0.5593901139437659	
		acc@10	0.6749118780227887	
K-Nearest Neighbors	n_neighbors=50	acc@1	0.2473358472005902	0.3517757928375802
		acc@5	0.49213050250020496	
		acc@10	0.5991269776211164	
Random Forest	n_estimators=150 max_features="sqrt" criterion="gini"	acc@1	0.37843265841462415	0.47087735366027217
		acc@5	0.5941470612345274	
		acc@10	0.6753422411673088	
Logistic Regression	multi_class="multinomial" solver="saga" penalty="none"	acc@1	0.39781949340109846	0.5055689495105653
		acc@5	0.6484342978932699	
		acc@10	0.7456348881055824	
Complement Naive Bayes	alpha=0.004	acc@1	0.20335683252725634	0.3183933402035542
		acc@5	0.47280514796294776	
		acc@10	0.6158086728420362	

Logistic Regression está por encima de todos los demás métodos mencionados anteriormente, tanto de acc@10 como de MRR@10. El tiempo de entrenamiento no es muy alto (menos de 1 hora), y tiene una predicción buena en términos generales.

Uno de los métodos que, aunque diera buena predicción, se descartaría, es K-Nearest Neighbors, y la razón es muy simple: poniéndonos en el caso de tener que crear una página web de recomendación, el hecho de que el tiempo de predicción supere el minuto y medio en la mayoría de los casos es muy malo, ya que no se puede tardar en predecir minuto y medio en una web con una búsqueda que se debe de realizar lo más rápida posible.

En los siguientes dataset, ya que he explicado toda la información de interés para cada algoritmo, simplemente haré una tabla con las accuracy score de cada método con los parámetros que han resultado ser superiores. La razón de esta decisión es que no hace falta repetirse, y dado que todo parte del mismo dataset los parámetros óptimos no van a variar prácticamente.

8.4 Resultados del dataset unigramas/bigramas

El primer dataset consistía en una prueba con los datos preprocesados iniciales. En estas pruebas utilizamos el dataset al que le añadimos los bigramas y unigramas al aplicar el vectorizado Tf-Idf. Vamos a realizar una comparación directa con los resultados anteriores. Contamos con 110.431 atributos frente a los 16.686 del dataset anterior.

8.4.1 Resumen de resultados

Tabla 23: Resumen métodos Score+MRR dataset 2

Método	Parámetros	acc@k	Accuracy score	MRR@10
Multinomial Naive Bayes	alpha=0.004	acc@1	0.34855316009508974	0.4533832951438956
		acc@5	0.591667349782769	
		acc@10	0.7011230428723666	
K-Nearest Neighbors	n_neighbors=50	acc@1	0.25596360357406345	0.3604736905965669
		acc@5	0.4992417411263218	
		acc@10	0.6045987376014428	
Random Forest	n_estimators=150 max_features="sqrt" criterion="gini"	acc@1	0.36486597262070664	0.45568747015433747
		acc@5	0.5774038855643905	
		acc@10	0.6585785720140995	
Logistic Regression	multi_class="multinomial" solver="saga" penalty="none"	acc@1	0.41849741782113287	0.5292815196275559
		acc@5	0.6768997458808099	
		acc@10	0.7734445446348062	
Complement Naive Bayes	alpha=0.004	acc@1	0.26270595950487746	0.3848290404365882
		acc@5	0.5529141732928928	
		acc@10	0.6653209279449135	

Vamos a hablar un poco de la *tabla 23*: en términos generales los scores de los métodos han mejorado comparándolos con el primer dataset. Contar con cadenas de 2 palabras y con palabras aisladas ha conseguido que se obtenga una predicción superior a la prueba anterior.

Donde más se ve mejoría es en Multinomial Naive Bayes y Complement Naive Bayes, ya que a pesar de que no tengan el mejor score su MRR ha aumentado considerablemente (*tabla 24*), lo que significa que también ha aumentado la posición media en el ranking de la revista correcta.

Tabla 24: Comparación MultinomialNB y ComplementNB MRR datasets 1 y 2

Método	Dataset 1 MRR@10	Dataset 2 MRR@10
Multinomial Naive Bayes	0.40837721541059285	0.4533832951438956
Complement Naive Bayes	0.3183933402035542	0.3848290404365882

Como ha ocurrido en el primer dataset, el mejor método vuelve a ser Logistic Regression, pero Multinomial Naive Bayes también tiene un buen score, por lo que si buscamos un método rápido la diferencia es enorme (51 minutos y 50 segundos de Logistic Regression frente 5 segundos de Multinomial Naive Bayes). Podemos decir que la mejor manera de atajar el problema, dada la información que hemos obtenido, por ahora es contando con el uso de bigramas y unigramas en nuestro dataset.

8.5 Resultados de los dataset de desbalanceo

Para ponernos en situación, el tercer dataset consistía en aplicarle la técnica de undersampling a los datos con los que ya contábamos del conjunto de entrenamiento (no de test, ya que este conjunto ha de ser el mismo para todos), y el cuarto dataset consistía en aplicar oversampling a los datos, para poder realizar una comparación de qué método es mejor para atacar el problema. A continuación vamos a realizar la elección del algoritmo a utilizar para cada uno de los datasets. Al principio contamos en los dos con 16.686 atributos distintos.

Las notebooks utilizados en las elecciones figuran, al igual que para las pruebas, en el *apéndice 3.1*.

8.5.1 Undersampling: selección del método a utilizar

He escogido los algoritmos RandomUnderSampler y AllKNN, ya que me parece interesante ver cómo actúan. Para realizar la comparación he escogido Multinomial Naive Bayes porque es rápido y nos dará una idea de con cual nos puede ir mejor en términos generales. Los parámetros utilizados son los dados por defecto para cada algoritmo:

- AllKNN: Aplica 3NN, toma como estrategia remuestrear (extraer muestras) en todas las clases menos en las clases pequeñas, con que uno de los vecinos esté de acuerdo en excluir una muestra se excluye, y no permite que las clases grandes se terminen convirtiendo en las clases pequeñas.
- RandomUnderSampler: Sampling sin reemplazo. Significa que un punto de datos de una muestra NO puede aparecer en otras muestras.

Utilizamos los mismos parámetros que con el dataset inicial para aplicar TfidfVectorizer. Las métricas utilizadas para realizar la comparación son el acc@1 y el tiempo:

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Tabla 25: Comparación UnderSampling

Método	Tiempo	acc@1
AllKNN	383.3s	0.2175998032625625
RandomUnderSampler	3.2s	0.21155422575620952

En la *tabla 25* vemos que AllKNN tiene una acc@1 superior, aunque el tiempo de cómputo es bastante mayor al RandomUnderSampler. Puede aumentar mucho la calidad de la predicción escogiendo otros métodos, pero esto ha servido para realizar una estimación. Por lo pronto escogeré como algoritmo el AllKNN para nuestro dataset, ya que tenemos la mayor calidad, aunque el proceso dure bastante más. Nos quedamos con un total de 51.735 artículos.

Tabla 26: Número de artículos por revista con AllKNN

Revista	Artículos
acm int. conf. proc. ser.	25
acs omega	45
acta biomed	210
acta cir. bras.	70
acta paediatr	41
acta pharmacol sin	33
adv differ equ	35
adv exp med biol	24
advances in experimental medicine and biology	26

Volviendo al caso de AllKNN, en la *tabla 26* observamos cómo no todas las revistas tienen el mismo número de artículos, pero esto se debe a que estos métodos borran los artículos que se encuentren en un grupo al que no pertenece, además de que es un algoritmo bastante estricto a la hora de borrar atributos, por lo que podemos deducir que todos esos artículos pertenecientes a algunas de las revistas se clasificaron erróneamente, y por ende fueron descartados.

8.5.2 Undersampling: resumen de resultados

En la *tabla 27* no vemos mejoría en ninguno de los métodos habiendo aplicado undersampling. No se puede hablar mucho más de este dataset, ya que los resultados no muestran ninguna peculiaridad aparente, y concluimos que para nuestro problema se descarta esta opción.

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Tabla 27: Resumen métodos Score+MRR dataset 3

Método	Parámetros	acc@k	Accuracy score	MRR@10
Multinomial Naive Bayes	alpha=0.004	acc@1	0.2175998032625625	0.30821522743864477
		acc@5	0.4275350438560538	
		acc@10	0.5355356996475121	
K-Nearest Neighbors	n_neighbors=50	acc@1	0.20763997048938437	0.29078937481215805
		acc@5	0.4040904992212476	
		acc@10	0.48415853758504795	
Random Forest	n_estimators=150 max_features="sqrt" criterion="gini"	acc@1	0.25873022378883515	0.34334170801147224
		acc@5	0.459013033855234	
		acc@10	0.5444093778178539	
Logistic Regression	multi_class="multinomial" solver="saga" penalty="none"	acc@1	0.2768259693417493	0.37899708309425845
		acc@5	0.5151446839904911	
		acc@10	0.6194974997950652	
Complement Naive Bayes	alpha=0.004	acc@1	0.18649069595868514	0.27329648735195633
		acc@5	0.3863226493974916	
		acc@10	0.4922739568817116	

El método de undersampling es un arma de doble filo, ya que nos puede servir en ciertas ocasiones, pero normalmente es un método que no vale la pena debido a la cantidad de datos que se pierden por el camino, y que son dependientes de la clase con menos muestras. Esta pérdida de calidad en la predicción es debida claramente a que el dataset ha disminuido drásticamente en tamaño (de 146.385 artículos a 51.735).

8.5.3 Oversampling: selección del método a utilizar

La comparación es entre el método RandomOverSampler con el método SMOTE, ya que interesa ver la potencia de este método en su forma original de cara al más popular. Sus parámetros por defecto son los siguientes:

- RandomOverSampler: Las muestras que se saquen no van a tener suavizado ninguno. Esto cuando está activado busca reducir la “sampling variability”, que es cuanto varía la predicción entre muestras.
- SMOTE: Utiliza 5NN para hacer la elección del vecino más cercano, y hace remuestreo de todas las clases excepto de las clases grandes.

Las pruebas las realizamos de la misma forma que antes, es decir, mediante el método MultinomialNB y utilizando las métricas acc@1 y tiempo. Los resultados son los siguientes:

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Tabla 28: Comparación OverSampling

Método	Tiempo	acc@1
RandomOverSampler	36.4s	0.29813919173702763
SMOTE	53.9s	0.29664316747274366

Sobre la *tabla 28*: revisando los tiempos podemos observar que difieren en 17.5s, lo que para nuestro problema no es muy significativo, y la acc@1 es muy similar entre los dos algoritmos, aunque por décimas RandomOverSampler supera a SMOTE. Dado que buscamos la mejor calidad de predicción continuaremos utilizando RandomOverSampler para las pruebas de oversampling, y nuestro dataset pasa a tener un total de 3.324.100 artículos.

En términos generales podemos observar cómo aplicando técnicas de balanceo de clases conseguimos que no haya clases predominantes sobre otras a la hora de entrenar el método escogido. Se recomienda utilizar oversampling como primera opción ante el problema de desbalanceo siempre que sea posible, ya que, a pesar que se reduce el problema con undersampling, se pierde mucha información, y normalmente da peores resultados.

8.5.4 Oversampling: resumen de resultados

Ahora vamos a pasar a ver el dataset con oversampling, y compararemos el rendimiento de los dos, además de finalmente compararlo con el dataset de mejores resultados (primer dataset).

Tabla 29: Resumen métodos Score+MRR dataset 4

Método	Parámetros	acc@k	Accuracy score	MRR@10
Multinomial Naive Bayes	alpha=0.004	acc@1	0.2985695548815477	0.40230337804166816
		acc@5	0.5401057463726535	
		acc@10	0.6478194934010985	
K-Nearest Neighbors	n_neighbors=50	acc@1	0.1982129682760882	0.2907680680956286
		acc@5	0.4230674645462743	
		acc@10	0.4316747274366751	
Random Forest	n_estimators=150 max_features="sqrt" criterion="gini"	acc@1	0.4033117468644971	0.49063572248311144
		acc@5	0.6061972292810887	
		acc@10	0.6870440200016394	
Complement Naive Bayes	alpha=0.004	acc@1	0.15568899090089353	0.24125525999399736
		acc@5	0.35443478973686365	
		acc@10	0.46102139519632757	

Para este dataset no hemos podido explorar el método de LogisticRegression debido a que el dataset alcanzaba los 3 millones de artículos y no entraba en la memoria, ni en mi máquina ni en el servidor bahía.ugr.es, del cual he hablado anteriormente. Por otro lado, he decidido no realizar cambios en el vectorizado limitando el número de atributos del vectorizado, ya que considero que

realizar eso provocará una diferencia en los preprocesados y la prueba ya no será significativa a la hora de realizar comparaciones.

En general los resultados (*tabla 29*) han mejorado frente a undersampling, pero no frente a los demás dataset. Podemos apreciar para KNN que los resultados han sido peores, y puede ser debido al tamaño tan grande del dataset, que provoca una alta dispersión de los distintos artículos del problema. e impide conseguir buenos resultados.

A pesar de ello, KNN no es un algoritmo a tener en cuenta en exceso, ya que aunque se diera el caso de que su score fuera el más alto, el tiempo que tarda importa en la etapa de predicción, y escala con el tamaño del dataset y el número de clases que contenga. El tiempo de predicción para KNN en este caso ha sido de más de 5 horas. Naturalmente un usuario medio no va a esperar 5 horas para obtener un resultado, así que este método no es relevante.

Observamos también que para Complement Naive Bayes el score es peor para oversampling. De todas formas observamos cómo ninguno de estos métodos han mejorado la calidad de predicción respecto al dataset con ngramas, el cual tiene un score@10 con Logistic Regression de 0.7734445446348062.

8.6 Resultados del dataset Deep Learning

En este apartado entrenaremos una Red Neuronal Long Short Term Memory (LSTM). Se explicará todo el proceso de creación y entrenamiento de la red, y finalmente se realizarán pruebas y se comparará con los demás datasets. Los notebooks relacionados se encuentran en el *apéndice 3.4*.

Los pasos a realizar durante el proceso son los siguientes:

- Tokenizado de palabras y generación de un vocabulario
- Padding de artículos
- Creación de una matriz de embeddings
- Indexado de las clases existentes
- Asignación de capas al modelo
- Compilación y entrenamiento del modelo

8.6.1 Construcción de vocabulario y padding

Tras la lectura de los datos, en la cual se leen los dataset de entrenamiento y de test preparados para Deep Learning (*punto 3.2*), se realiza la separación de palabras de todos los

artículos y se almacenan en una variable mediante una función sencilla, para luego ser tokenizado por la función Tokenizer de Keras. Tras esto se asigna el tamaño del vocabulario, el cual es dado por el tamaño de una variable `word_index` (lista de palabras tokenizadas) dada por la propia función, y se le suma 1 a ese total.

Ahora pasamos al proceso de padding. El padding es una técnica que ajusta todas las frases del dataset al mismo número de palabras. Si se escoge un largo de frase menor al máximo las frases se truncan, y si no se añaden palabras con valor 0 (no son relevantes). He escogido un tamaño máximo de 400 para reducir el tamaño del modelo, ya que con el tamaño máximo de palabra el largo es de 1366, lo que ralentizaría mucho el proceso de entrenamiento.

8.6.2 Creación de matriz de embeddings

El embedding aplica técnicas de Machine Learning o Deep Learning para aprender las relaciones entre las palabras, y se guardan en una matriz de embeddings.

Para realizar el embedding utilicé un embedding proveniente de internet. Hay muchos embedding de distintos temas y es sencillo encontrar uno que se ajuste a las necesidades del usuario. En mi caso utilizo un embedding creado mediante Word2Vec sacado de una serie de textos completos y abstract provenientes de publicaciones de PubMed, los cuales se ajustan al tema del dataset utilizado (<http://evexdb.org/pmresources/vec-space-models/>). El tamaño del embedding es de 200.

El embedding está en formato binario, y para convertirlo a un TXT utilicé un script de C. Tras aplicar la técnica de embedding correspondiente la información se añade mediante una función a una matriz de embeddings para que posteriormente pueda ser aplicada en la red LSTM. Puedes encontrar los archivos involucrados en el *apéndice 3.3*.

8.6.4 Construcción del modelo

Una red neuronal se compone de varias capas que se alimentan de una entrada y generan una salida. Las capas que vamos a ver serán las siguientes:

- Capa Embedding: Comprime el espacio de atributos a menor tamaño. Se prepara una matriz de embeddings y se le aplica al vocabulario tokenizado con el que contamos. En términos generales reduce el vocabulario que tenemos separando por distintas categorías. Los argumentos que utilizaremos son los siguientes [52]:
 - `input_dim`: Tamaño del vocabulario.
 - `output_dim`: Dimensión de salida del embedding.

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

- weights: Pesos del embedding (se le pasa la matriz de embedding).
- trainable: Escoge si los pesos se actualizan durante el entrenamiento. Puede ocasionar sobreentrenamiento.
- Capa LSTM: Explicado anteriormente. Capa central del modelo de red LSTM. Los argumentos con los que vamos a trabajar son los siguientes [53]:
 - units: Dimensionalidad del output. Define el input de la siguiente capa.
 - return_sequences: Permite que se devuelvan todos los estados ocultos (*hidden states*) de las capas. En algunos casos es requerido para entrenar al modelo, ya que a veces se necesita información adicional, y si se especifica la capa como bidireccional es necesario e imprescindible.
 - dropout: Método de regularización que ignora aleatoriamente neuronas durante el proceso de entrenamiento. Es utilizado para prevenir el sobreentrenamiento de clases. Se especifica una fracción de unidades que ignorar.
- Capa Bidirectional: Implementa una comunicación en dos direcciones: hacia delante y hacia atrás. Utiliza la información proveniente de los dos frentes, y ayuda a modelar las dependencias secuenciales entre palabras y frases, tomando información del pasado y del presente [54].
- Capa Dense: Capa normal. Se aplica una función de activación al producto escalar entre la entrada y la matriz de pesos de la capa, y sumado el coeficiente bias. Un coeficiente bias es especificado si se quiere realiza algún ajuste u optimización al modelo, y sirve para ajustar la función de activación hacia la derecha o a la izquierda.

La función de activación es una parte importante del proceso. Vamos a trabajar con dos funciones de activación distintas [55]:

- Activation="relu": La función de activación Rectified Linear se utiliza para las capas ocultas (hidden layers), que son las capas que se encuentran entre las capas de entrada y salida de la red neuronal. Simple y eficiente frente a otras funciones de activación existentes.

Se calcula con $\max(0.0, x)$: si el valor de x (pesos del input de la capa anterior) es negativo se devuelve 0.0.

- Activation="softmax": Es la función argmax pero llevada a probabilidades. Devuelve un vector de valores de 0.0 a 1.0, que pueden ser interpretados como probabilidades de que los atributos pertenezcan a una clase concreta. Es adecuado para problemas multiclase. Se calcula para cada input la siguiente función:

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

$$\text{softmax}(x_n) = \frac{e^{x_n}}{\sum_{i=1}^w e^{x_i}}$$

Siendo x_n uno de los inputs de la capa anterior, la cual devuelve un vector como forma de output, y en el denominador se realiza la sumatoria de todos los inputs.

Si la capa Dense es la última de la red neuronal se debe especificar (para SoftMax) la dimensionalidad del output igual al número de clases existentes en el modelo, ya que devuelve la distribución de probabilidad de cada una de las clases.

Por otro lado, si la capa Dense está entre el conjunto de capas intermedias (*hidden layers*), esta capa actúa como selector de atributos para decidir qué atributos son relevantes para una clase y cuales no.

El primer modelo construido tiene las siguientes capas:

1. Capa Embedding para reducir la dimensionalidad del problema. He utilizado un embedding sacado de PubMed mencionado anteriormente. Sus parámetros iniciales son:
 - Tamaño de vocabulario: 390.053
 - Tamaño de padding: 400
 - Tamaño de embedding: 200
 - Trainable=False
2. Capa LSTM con los siguientes parámetros:
 - Dimensionalidad del output: 128
 - return_sequences=True
 - dropout=0.2
3. Capa LSTM con los siguientes parámetros:
 - Dimensionalidad del output: 64
 - dropout=0.2
4. Capa Dense con los siguientes parámetros:
 - Dimensionalidad del output: 32
 - Función de activación RELU
5. Capa Dense con los siguientes parámetros:

- Dimensionalidad del output: 650
- Función de activación SoftMax

8.6.3 Indexado de clases

Para que las distintas clases sean tratadas correctamente por la red LSTM debemos de indexarla, es decir, asignarle un número único a cada una de las clases. Primero se genera un set de clases único y se convierte en un diccionario que guarda un índice por clase. Seguidamente hace el inverso, y luego se crea una función que convierte cada una de las clases de los distintos artículos en su índice correspondiente, y es aplicado al conjunto de entrenamiento y test.

8.6.5 Compilación del modelo

La compilación del modelo se realiza con el método `compile()` de Keras: aquí se define la función de pérdida (*loss*), el optimizador y la métrica a utilizar.

Una función de pérdida es una función que evalúa la desviación entre las predicciones realizadas y los valores reales. A menor resultado de la función mejor es la red [56]. La función que vamos a utilizar es Sparse Categorical Cross-entropy, la cual realiza la función de coste Cross-entropy pero aplicada a probabilidades.

El optimizador es una función que se utiliza para minimizar la función de pérdida, y por ende para maximizar la eficiencia del modelo. Nosotros utilizaremos Adaptive Moment Estimation (ADAM). Es uno de los algoritmos más populares y fáciles de implementar, además de ser computacionalmente eficiente y pedir pocos requisitos de memoria [57].

Finalmente, la métrica que utilizaremos será la Top K Categorical Accuracy recuperando 10 revistas, y calcularemos el Mean Reciprocal Rank. El número de épocas de entrenamiento será de 20.

8.6.6 Resultados y ajustes de modelo

Ahora vamos a pasar a poner a prueba el modelo creado, habiendo realizado todo el proceso de construcción de este. Veremos resultados e iremos ajustando el modelo de cara a mejorar la calidad de la predicción.

Los resultados con el modelo inicial son los siguientes:

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Tabla 30 Métricas calculadas modelo 1

Métrica	Valor	Tiempo
acc@10	0.5718706	37544.6s
MRR@10	0.3265923	

Observamos en la *tabla 30* una precisión intermedia. Esto significa que nuestro embedding está siendo eficaz. Recordemos que hay desbalanceo de clases, por lo que la precisión se ve muy afectada por esto.

En la etapa de validación se ve que ha estado aumentando todavía, por lo que probablemente con 20 épocas este modelo no consiga estabilizarse, pero para no haberlo hecho tenemos un buen punto de partida.

A continuación vamos a convertir las capas LSTM en bidireccionales. Para refrescar el concepto, una capa bidireccional propaga información hacia delante y hacia atrás, contando con información sobre el futuro y el presente, y así en la mayoría de los casos mejorando el aprendizaje. Nos quedamos con el siguiente modelo:

1. Capa Embedding
2. Capa LSTM bidireccional
3. Capa LSTM bidireccional
4. Capa Dense intermedia
5. Capa Dense final

No se realizaron cambios en los parámetros. Los resultados son los siguientes:

Tabla 31 Métricas calculadas modelo 2

Métrica	Valor	Tiempo
acc@10	0.71893185	41042.9s
MRR@10	0.45381323	

En la *tabla 31* vemos gran mejoría aplicando capas bidireccionales. Demostramos cómo efectivamente una comunicación bidireccional puede ayudar al entrenamiento muy eficientemente. En la etapa de validación de cada época se estaba prácticamente estabilizando. Entre la época 19 y la 20 la diferencia era de 0.0003, por lo que podemos decir que usando capas bidireccionales el modelo converge más rápidamente, y ahorra tiempo en forma de épocas.

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Adicionalmente se aprecia un aumento en el tiempo de entrenamiento (1 hora más aproximadamente), pero esto se debe a que las capas bidireccionales requieren un poco más de tiempo de entrenamiento por época.

Para la siguiente prueba voy a probar a borrar la capa intermedia Dense. Esta capa funciona como selector de atributos para decidir qué atributos son relevantes para una clase y cuales no, inmediatamente antes de sacar la distribución de probabilidad de la capa final.

Nos quedamos con el siguiente modelo:

6. Capa Embedding
7. Capa LSTM bidireccional
8. Capa LSTM bidireccional
9. Capa Dense final

Los resultados son los siguientes:

Tabla 32 Métricas calculadas modelo 3

Métrica	Valor	Tiempo
acc@10	0.7464751	40317.6s
MRR@10	0.47302544	

En la *tabla 32* hemos obtenido los mejores resultados hasta ahora utilizando una red neuronal. Es un resultado muy bueno y el MRR tiene un score adecuado. Esto nos indica que posiblemente la capa Dense estuviera evitando que algunos atributos fueran incluidos correctamente, por lo que seguiremos adelante sin contar con esa capa. El tiempo de entrenamiento es similar al de la segunda prueba.

Como última prueba probaremos activando la actualización de pesos durante el entrenamiento (`trainable=True`) en la capa Embedding para ver si mejora nuestra predicción. Tenemos el siguiente modelo:

1. Capa Embedding
 - `trainable=True`
2. Capa LSTM bidireccional
3. Capa LSTM bidireccional
4. Capa Dense final

Los resultados son los siguientes:

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Tabla 33 Métricas calculadas modelo 4

Métrica	Valor	Tiempo
acc@10	0.62019426	55949.8s
MRR@10	0.42617935	

Observamos en la *tabla 33* cómo ha disminuido la precisión. Durante el entrenamiento el score daba 0.9, pero a la hora de la predicción con el conjunto de test ha disminuido drásticamente, por lo que significa que ajustar los pesos en la etapa de entrenamiento provoca sobreentrenamiento y, por ende, un mal entrenamiento.

Además de una peor precisión, se aprecia que actualizar los pesos añade un tiempo de entrenamiento bastante mayor a los modelos vistos anteriormente. Aunque esto no fuera un grave impedimento, ya que para el entrenamiento los tiempos son relativamente asumibles tratándose de una red neuronal, es algo a tener en cuenta.

El tercer modelo visto ha sido el que mejor resultado nos ha dado. A pesar de tener un score más bajo que algunos resultados de las pruebas de Machine Learning, probablemente un ajuste en los parámetros podría aumentar bastante el rendimiento de nuestra red neuronal. En este tipo de modelos se trabaja mucho mediante prueba y error, ya que las redes suelen ser específicas para el problema.

Como conclusión final de este apartado escogería el Deep Learning por encima de cualquier técnica de Machine Learning debido a que, a pesar de haber obtenido un score un poco peor, un correcto ajuste podría darnos unos mejores resultados para resolver nuestro problema, y probablemente aumentar el número de épocas también.

Capítulo 9

Dataset con texto completo

En este capítulo vamos a explorar la última de las opciones que tenemos: estudiar si influye contar con el texto completo de los artículos de cara a mejorar la calidad de la predicción y aumentar el Mean Reciprocal Rank.

Como ya se ha mencionado anteriormente, el texto completo según la bibliografía explorada sobre sistemas de recomendación no ha sido tenido en cuenta, y este tipo de clasificación textual con este dataset no se ha explorado, por lo que las conclusiones que saquemos de esta parte será un punto importante en nuestro estudio.

Recordemos que los datos estaban almacenados por separado: un archivo CSV con información “reducida” sobre los artículos, y un conjunto de archivos JSON con el texto completo de cada artículo. En el punto siguiente vemos cómo hemos juntado los distintos dataset con los que contamos para obtener el dataset inicial pero con texto completo añadido a los artículos que cuenten con el.

El archivo final y los involucrados se encuentran en el *apéndice 1*.

9.1 Transformaciones de los datos

Aquí veremos las transformaciones realizadas al dataset, las cuales serán similares a las que hemos visto en el capítulo 3. Se parte del conjunto de entrenamiento derivado del primer dataset de todos, y solamente se realizarán cambios en la parte de entrenamiento, y el conjunto de test se mantendrá intacto para que la comparación sea significativa. Todas las transformaciones se han recopilado en notebooks, los cuales se encuentran en el *apéndice 2.2*.

Los pasos que se han seguido han sido los siguientes:

1. Extracción del texto completo
 - Creación de estructura para almacenar documento JSON
 - Extraer artículos JSON a CSV
2. [Deep Learning] Preprocesado de datos
 - Combinación dataset con texto completo
 - Conversión de texto a minúsculas y borrado de caracteres no ASCII

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

- Borrado de stopwords
- Unión de abstract y texto completo

3. [Machine Learning] Preprocesado de datos

- Stemming del texto disponible
- Tokenizado del texto aplicando Tf-Idf

9.1.1 Extracción de texto de los artículos

Para realizar la extracción del texto y el ID del artículo he creado una clase que a partir de un fichero JSON referente a un artículo me extraiga tanto el texto completo como su ID.

Tras esto realicé un bucle que recorriera la carpeta pdf_json, extraje toda la información de cada uno para luego juntarlo todo en un dataframe y guardarlo en un archivo csv.

9.1.2 Deep Learning - Preprocesado

El preprocesado parte leyendo el csv extraído anteriormente, junto con el dataset preprocesado para Deep Learning (*punto 6.3*) y el dataset de entrenamiento para Deep Learning (*punto 6.4*). Los campos que contienen cada uno son los siguientes (*tabla 34*):

Tabla 34: Campos de los archivos csv

CSV	Campos
Documentos JSON	full_text sha
Preprocesado DL	abstract journal sha
Train DL	abstract journal

El siguiente paso es combinar los archivos csv para conseguir el dataset deseado. Primero borramos las tuplas nulas provenientes del preprocesado DL, luego juntamos los datos de entrenamiento DL con los datos preprocesado DL por abstract y journal, y finalmente juntamos el resultante con el texto completo mediante la columna sha.

De esta manera obtenemos lo que necesitamos y sin perder ninguno de los datos con los que contábamos anteriormente. Por último borramos la columna sha, ya que no la requerimos.

El siguiente paso es pasar todo el texto a minúsculas y quitar los caracteres no ASCII del texto completo. Seguido de esto se borrarán todos los stopwords del texto completo, al igual que

con el dataset original, y por último juntaremos el abstract con el texto completo para tener solamente dos columnas: abstract y journal.

El conjunto final tiene un total de 1.924.306 palabras únicas y 208.629.448 palabras totales, y ya tenemos nuestro dataset preparado para la prueba de Deep Learning.

9.1.3 Machine Learning - Preprocesado

Este apartado puede verse como una ampliación del anterior, ya que los pasos realizados hasta ahora se vuelven a repetir de manera idéntica. Adicionalmente se realiza el aplicado del stemming Porter, al igual que para el preprocesado sin texto completo. Hasta este punto contamos con los siguientes dataset:

- Dataset de entrenamiento (Deep Learning):
 - 146.385 artículos de texto completo.
 - 1.924.306 palabras únicas.
 - 208.629.448 palabras totales.
- Dataset de entrenamiento (Machine Learning):
 - 146.385 artículos de texto completo.
 - Se realiza stemming de la bolsa de palabras.
 - 1.034.681 palabras únicas.
 - 70.404.911 palabras totales.

Como ya se ha mencionado anteriormente, el conjunto de test se mantiene intacto para que las comparaciones sean significativas. Finalmente se aplica el vectorizado Tf-Idf, de iguales características que el dataset original, y nos quedamos con un total de 56.588 atributos, lo que es menos que para el dataset con n-gramas pero más que para los demás casos.

9.2 Machine Learning - Resultados

Este dataset es el penúltimo que veremos. Como hemos mencionado anteriormente, consiste en la integración del texto completo al primer dataset que hemos visto, añadiendo a los artículos que lo tuvieran el texto que les falta, y juntando todo en el campo abstract. Veamos qué tal se ha comportado en términos generales. Los notebooks utilizados en las pruebas se encuentran en el *apéndice 3.2*.

A continuación veremos los resultados obtenidos con los algoritmos de Machine Learning (*tabla 35*):

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Tabla 35: Resumen métodos Score+MRR dataset 5

Método	Parámetros	acc@k	Accuracy score	MRR@10
Multinomial Naive Bayes	alpha=0.004	acc@1	0.2588121977211247	0.36929815187819587
		acc@5	0.5178293302729732	
		acc@10	0.6394376588244938	
K-Nearest Neighbors	n_neighbors=50	acc@1	0.22618657266989098	0.33333536641699335
		acc@5	0.47585867694073286	
		acc@10	0.5855602918271989	
Random Forest	n_estimators=150 max_features="sqrt" criterion="gini"	acc@1	0.24317567013689648	0.3115467196114307
		acc@5	0.4025534879908189	
		acc@10	0.4748749897532585	
Logistic Regression	multi_class="multinomial" solver="saga" penalty="none"	acc@1	0.3333060086892368	0.43611364026716487
		acc@5	0.5718911386179195	
		acc@10	0.6718993360111485	
Complement Naive Bayes	alpha=0.004	acc@1	0.19054840560701697	0.3033182689968288
		acc@5	0.4525575866874334	
		acc@10	0.5885728338388393	

Observamos que a pesar de tener un número bastante más alto de atributos con los que contar, esto ha afectado negativamente al rendimiento. Para Multinomial Naive Bayes contamos con el segundo peor acc@10, por encima de undersampling.

Su mejor score es con Logistic Regression, como lleva siendo con los demás datasets, aunque con peores predicciones exceptuando undersampling. Para el caso de Random Forest vemos que predice al mismo nivel que otros dataset, como el primer dataset y el que incluye bigramas, por lo que para ese algoritmo, al no tener profundidad indefinida (max_depth=200), no se pueden aprovechar al máximo el número de atributos.

En términos de MRR el mejor score lo tiene Logistic Regression con 0.37, lo que también es más bajo que la media. Esto demuestra que por mucho que añadamos una cantidad de términos grande, si estos no son relevantes o discriminatorios no mejorará la calidad de la predicción, y en muchos casos la empeorará.

9.3 Deep Learning - Resultados

Ahora veremos el último dataset de todos. Para realizar las pruebas utilizaremos el mismo modelo LSTM ajustado en el apartado 8.6. Para contextualizar, contábamos con un modelo de red neuronal con las siguientes capas:

1. Capa Embedding
2. Capa LSTM bidireccional
3. Capa LSTM bidireccional

4. Capa Dense final

En cuanto a los parámetros y la construcción del modelo, la única diferencia es haber aumentado el tamaño máximo de padding a 800. Los notebooks utilizados en las pruebas se encuentran en el *apéndice 3.4*, y este dataset puedes encontrarlo en el *apéndice 2.2*.

Veremos los resultados y los compararemos con los resultados de las pruebas que no contienen texto completo. Tras el entrenamiento nuestro modelo nos ha dado los siguientes resultados:

Tabla 36 Métricas calculadas modelo 1

Métrica	Valor	Tiempo
acc@10	0.32170668	88185.3s
MRR@10	0.16903153	

Observamos en la *tabla 36* que el rendimiento es de los más bajos que hemos obtenido. En el score de entrenamiento da 0.7947, y la diferencia es tan grande que significa que se está dando el caso de un alto sobreentrenamiento debido al texto completo.

En cuanto a los tiempos, era de esperar que estos aumentaran considerablemente (aproximadamente el doble), debido a que hemos aumentado el padding al doble anterior, y se tienen en cuenta más atributos distintos a la hora de entrenar nuestra red.

Este score podría haber mejorado aumentando el tamaño del padding, ya que contamos con un artículo que contiene hasta 31.770 palabras, pero esto implicaría unos tiempos que no pueden ser tomados por el hardware con el que contamos, y un uso de memoria que posiblemente sea muy alto. Para esta prueba he requerido 24 horas, frente a 8 horas aproximadamente en el caso de padding 400, por lo que el tiempo de entrenamiento aumentaría exponencialmente.

En este problema podemos concluir que el uso del texto completo aporta muy poco si lo que queremos es mantener unos tiempos y unos recursos similares, y en términos generales no nos ha beneficiado el uso de los textos completos.

Capítulo 10

Conclusión

Durante este trabajo hemos podido ver una puesta en práctica del procesado y entrenamiento de un dataset, de cara a crear un sistema de recomendación con un gran número de clases. Hemos podido comprobar el rendimiento de varios algoritmos de Machine Learning y Deep Learning, y podido a raíz de esos sacar conclusiones y aprender un poco más sobre cada uno de estos diversos métodos.

10.1 Vista general del trabajo realizado

Para empezar, daremos un repaso a los mejores algoritmos de Machine Learning para cada dataset distinto con el que contamos:

Tabla 37: Comparación mejores métodos datasets

Dataset	Mejor método	acc@10	MRR@10
Normal	Logistic Regression	0.7456348881055824	0.5055689495105653
Unigramas y bigramas	Logistic Regression	0.7734445446348062	0.5292815196275559
Undersampling	Logistic Regression	0.6194974997950652	0.37899708309425845
Oversampling	Random Forest	0.6870440200016394	0.49063572248311144
Texto completo	Logistic Regression	0.6718993360111485	0.43611364026716487

Viendo los resultados de la *tabla 37* podemos observar que el texto completo no es una opción viable a la hora de realizar un entrenamiento de Machine Learning. La técnica de Oversampling obtiene mejores resultados que el texto completo a costa de almacenamiento y tiempos altos.

Por otro lado, vemos que contar con unigramas y bigramas en nuestro resumen obtenemos un mayor score, y un buen MRR. El número de atributos ha aumentado de 16.686 a 110.431, lo que provoca un aumento en el tiempo de entrenamiento pero no es significativo.

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

Recordemos también que el algoritmo K-Nearest Neighbours es un algoritmo inviable de cara a tener que contar con requisitos de tiempos inmediatos a la hora de predecir. Esto es debido a que los cálculos se hacen en la predicción y no en el entrenamiento, y la respuesta puede demorarse minutos, e incluso horas como es el caso del dataset que aplica Oversampling.

En la siguiente tabla veremos los resultados que hemos obtenido empleando una red Long-Short Term Memory (LSTM) para entrenar nuestros dataset:

Tabla 38: Rendimiento LSTM datasets

Dataset	acc@10	MRR@10
Normal	0.7464751	0.47302544
Texto completo	0.32170668	0.16903153

Observamos nuevamente (*tabla 38*) un rendimiento inferior en el caso del texto completo. Como se ha mencionado antes, esto se podría mejorar aumentando el tamaño del padding, pero supondría un aumento exponencial en el tiempo de entrenamiento con el que no contamos.

Con el dataset normal obtenemos score de 0.74 y un MRR de 0.47, lo que es un poco inferior al caso de Logistic Regression empleando n-gramas, pero dado que, realizando un ajuste óptimo de la red, aumentando el número de épocas, aumentando el tamaño de padding o cambiando el embedding podríamos alcanzar un mejor score, utilizaría para mi problema la red LSTM frente a los algoritmos de Machine Learning. Cabe destacar que no contamos con requisitos de tiempo de entrenamiento bajos, y todo esto es ignorando el aumento en este tiempo y el posible aumento de las prestaciones que requiera entrenar la red, por lo que en el caso de contar con pocos recursos las técnicas de Machine Learning tienden a ser menos costosas en tiempo y recursos de computación que el Deep Learning.

Como conclusión de este trabajo, hemos visto cómo una mayor cantidad de texto no es siempre equivalente a un mejor rendimiento. De hecho, utilizar más texto requiere más almacenamiento y recursos para su correcto entrenamiento, por lo que no es computacionalmente viable contar con los textos completos de un artículo a la hora de crear un sistema de recomendación, sino que con simplemente un resumen del artículo tenemos suficiente para que nuestro modelo de aprendizaje pueda realizar buenas predicciones.

10.2 Experiencia personal

Antes de la realización de este TFG había tenido un pequeño contacto con el Machine Learning en la asignatura de Inteligencia de Negocio, impartida con el profesor D. Francisco Herrera Triguero. Fue una asignatura que disfruté especialmente, y a pesar de estar especializado en la rama de Sistemas de Información me empecé a interesar por la Inteligencia Artificial. Otra asignatura que me ha ayudado a comprender más a fondo los conceptos vistos en este trabajo ha sido la asignatura de Recuperación de Información, impartida por uno de mis tutores de TFG: D. Juan Francisco Huete Guadix, ya que en esta asignatura se ve la fase de preprocesamiento de los datos, cosa que se ha utilizado, y se ve una forma alternativa a la hora de afrontar el problema de crear un Sistema de Recomendación, la cual no he incluido.

Me ha fascinado poder escoger un TFG que de verdad me entusiasmara, y he aprendido de manera autodidacta muchas de las metodologías empleadas. Nunca había sido consciente del potencial que tienen las matemáticas y la Inteligencia Artificial hasta haber entrado en materia. Ya contaba con experiencia previa utilizando Python y los notebooks de Jupyter, lo que me ha facilitado mucho la realización del trabajo. Librerías como Tensorflow o ImbLearn eran desconocidas para mí, pero me ha resultado relativamente sencillo comprenderlas y adaptarme a ellas.

En cuanto al Deep Learning, no había tenido contacto previo, por lo que es la parte que más me ha costado de todo mi TFG. Gracias a haber realizado este trabajo me he dado cuenta de lo que realmente me gusta la informática, y ha despertado en mí la curiosidad por aprender más sobre todo lo relacionado con la Inteligencia Artificial. Estoy seguro de que esto ha sido un antes y un después en mi futuro profesional.

Apéndice

Apéndice 1: Archivos de datos

1. Archivos troncales

Tabla 39: Archivos troncales

Descripción	Carpeta	Nombre del archivo
Dataset original	/data	data.csv (Descargar)
Artículos JSON	/data/json-files	*.json (Descargar)
Dataset limpio	/data	data-limpia.csv (Descargar)
Deep Learning - Dataset preprocesado	/data	data-preprocesada-lstm.csv (Descargar)

2. Deep Learning - Archivos de entrenamiento

Tabla 40: Deep Learning - Archivos de entrenamiento

Descripción	Carpeta	Nombre del archivo
Dataset train	/data/train	data-train-lstm.csv (Descargar)
Dataset train texto completo	/data/train	data-fulltext-train-lstm.csv (Descargar)

3. Machine Learning - Archivos de entrenamiento

Tabla 41: Machine Learning - Archivos de entrenamiento

Descripción	Carpeta	Nombre del archivo
Dataset train	/data/train	data-train.csv (Descargar)
Dataset train texto completo	/data/train	data-fulltext-train.csv(Descargar)

4. Archivos de test

Tabla 42: Archivos de test

Descripción	Carpeta	Nombre del archivo
Deep Learning - Dataset test	/data/test	data-test-lstm.csv (Descargar)
Machine Learning - Dataset test	/data/test	data-test.csv (Descargar)

Apéndice 2: Código de preprocesado

1. Pipeline (/src/pipeline)

Tabla 43: Pipeline (/src/pipeline)

Descripción	Secciones	Nombre del archivo
Limpieza de dataset	6.1	1-data-limpia.ipynb
Deep Learning - Preprocesado	6.2.1	2-data-preprocesada.ipynb
División train/test	6.2.2	3-data-division-traintest.ipynb
Machine Learning - Preprocesado	6.3.1	4-stemming.ipynb

2. Pipeline de texto completo (/src/pipeline-fulltext)

Tabla 44: Pipeline de texto completo (/src/pipeline-fulltext)

Descripción	Secciones	Nombre del archivo
Estructura de artículos	9.1.1	CovidArticle.py
Recopilar JSON en CSV	9.1.1	1-pasar-json-csv.ipynb
Deep Learning - Preprocesado	9.1.2	2-juntar-text-abstract.ipynb
Machine Learning - Preprocesado	9.1.3	3-stemming.ipynb

Apéndice 3: Archivos de pruebas

1. Elección del método de desbalanceo (/src/elección)

Tabla 45: Elección del método de desbalanceo (/src/elección)

Descripción	Secciones	Nombre del archivo
RandomUnderSampler	8.5.1	undersampling-random.ipynb
AllKNN	8.5.1	undersampling-allknn.ipynb
RandomOverSampler	8.5.3	oversampling-random.ipynb
SMOTE	8.5.3	oversampling-smote.ipynb

2. Machine Learning - Archivos de pruebas (/src)

Tabla 46: Machine Learning - Archivos de pruebas (/src)

Descripción	Secciones	Nombre del archivo
Dataset normal	8.3	dataset-normal.ipynb
Dataset con unigramas/bigramas	8.4	dataset-ngram.ipynb
Dataset undersampling	8.5.2	dataset-undersampling.ipynb
Dataset oversampling	8.5.4	dataset-oversampling.ipynb
Dataset texto completo	9.2	dataset-fulltext.ipynb

3. Deep Learning - Embedding (/src/lstm/embedding)

Tabla 47: Deep Learning - Embedding (/src/lstm/embedding)

Descripción	Secciones	Nombre del archivo
Conversor embedding BIN a TXT	8.6.2	distance.c
Embedding BIN	8.6.2	PubMed-w2v.bin (Descargar)
Embedding TXT	8.6.2	PubMed-w2v.txt (Descargar)

4. Deep Learning - Archivos de pruebas (/src/lstm)

Tabla 48: Deep Learning - Archivos de pruebas (/src/lstm)

Descripción	Secciones	Nombre del archivo
Dataset normal	8.6	lstm.ipynb
Dataset texto completo	9.3	lstm-fulltext.ipynb

Ilustraciones

Ilustración 1 Sistema de Recomendación Elsevier JournalFinder	10
Ilustración 2 Campos rellenos Elsevier JournalFinder	10
Ilustración 3 Resultados de la búsqueda Elsevier JournalFinder	11
Ilustración 4 Pipeline de la clasificación de texto [8]	19
Ilustración 5 Ejemplo espacio KNN [20]	25
Ilustración 6 Ejemplo árbol de decisión [n]	26
Ilustración 7 Scatter plot con sigmoide [31]	27
Ilustración 8 Estructura interna de red LSTM [35]	29
Ilustración 9 Gráfica de artículos de CORD-19 hasta 2020 [38]	31
Ilustración 10 Ejemplo archivo CSV dataset	34
Ilustración 11 X: Revistas Y: Artículos (log2) antes de quitar nulos	36
Ilustración 12 X: Revistas Y: Artículos (log2) tras quitar nulos	37
Ilustración 13 X: Revistas Y: Artículos (log2) tras ajustar revistas	38
Ilustración 14 X palabras distintas, Y frecuencia de las palabras Demostración Ley de Zipf ..	40
Ilustración 15 Explicación KNN 1	54
Ilustración 16 Explicación KNN 2	54

Bibliografía

- [1] R. Muñoz, «Fundación iS+D,» 30 05 2022. [En línea]. Available: <https://isdfundacion.org/2021/07/02/el-origen-y-evolucion-de-la-ciencia-de-datos-data-science/>.
- [2] Grapheverywhere, «Sistemas de recomendación | Qué son, tipos y ejemplos,» 02 12 2019. [En línea]. Available: <https://www.grapheverywhere.com/sistemas-de-recomendacion-que-son-tipos-y-ejemplos/>.
- [3] J. G. y. A. J. H. Dehdarirad, «Scholarly publication venue recommender systems:A systematic literature review,» *Data Technologies and Applications*, vol. 54, nº 2, pp. 169-191, 2020.
- [4] M. J. S. y. J. A. Kors, «Jane: suggesting journals, finding experts,» *Bioinformatics*, vol. 24, nº 5, pp. 727-728, 2008.
- [5] Z. Y. y. B. D. Davison, «Distinguishing venues by writing styles,» de *Proceedings of the 12th ACM/IEEE-CS joint conference on Digital Libraries - JCDL '12*, Washington, DC, USA, 2012.
- [6] H. K. K. y. T. Tyagi, «ACM Venue Recommendation System CS6604 - Digital Libraries Final Project Report,» Blacksburg, Virginia, 2019.
- [7] B. Droste, «Google Colab Pro+: Is it worth \$49.99?,» Towardsdatascience, 18 1 2022. [En línea]. Available: <https://towardsdatascience.com/google-colab-pro-is-it-worth-49-99-c542770b8e56>. [Último acceso: 4 7 2022].
- [8] J. M. H. M. B. y. B. Kowsari, «Text Classification Algorithms: A Survey,» *Information*, vol. 10, nº 4, p. 150, 2019.
- [9] J. Weng, «NLP Text Preprocessing: A Practical Guide and Template,» 9 1 2021. [En línea]. Available: <https://towardsdatascience.com/nlp-text-preprocessing-a-practical-guide-and-template-d80874676e79> . [Último acceso: 4 7 2022].
- [10] L. J. Sarica S, «Stopwords in technical language processing,» *PLoS ONE*, vol. 16, nº 8, 2021.
- [11] M. E. P. y. T. Abbas, «Development for Performance of Porter Stemmer Algorithm,» *Eastern-European Journal of Enterprise Technologies*, vol. 1, nº 2, pp. 6-13, 2021.
- [12] SKLearn, «6.2.3.4. Tf-idf term weighting,» SKLearn, [En línea]. Available: https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction. [Último acceso: 4 7 2022].

- [13 S. Kapadia, «Language Models: N-Gram,» Towardsdatascience, 19 8 2019. [En línea].
] Available: <https://towardsdatascience.com/introduction-to-language-models-n-gram-e323081503d9>. [Último acceso: 4 7 2022].
- [14 D. N. Jeevanandam, «What Does Machine Learning Embedding Mean?,» Analytics India
] Magazine, 5 9 2021. [En línea]. Available: <https://analyticsindiamag.com/machine-learning-embedding/> . [Último acceso: 4 7 2022].
- [15 D. Karani, «Introduction to Word Embedding and Word2Vec,» Towardsdatascience, 2 9
] 2020. [En línea]. Available: <https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa>. [Último acceso: 4 7 2022].
- [16 KDnuggets, «Implementing Deep Learning Methods and Feature Engineering for Text
] Data: The Skip-gram Model,» [En línea]. Available:
<https://www.kdnuggets.com/implementing-deep-learning-methods-and-feature-engineering-for-text-data-the-skip-gram-model.html/> . [Último acceso: 4 7 2022].
- [17 A. Akdogan, «Word Embedding Techniques: Word2Vec and TF-IDF Explained,»
] Towardsdatascience, 22 7 2021. [En línea]. Available:
<https://towardsdatascience.com/word-embedding-techniques-word2vec-and-tf-idf-explained-c5d02e34d08>. [Último acceso: 4 7 2022].
- [18 Shriram, «Multinomial Naive Bayes Explained: Function, Advantages & Disadvantages,
] Applications in 2022,» UpGrad, 3 1 2021. [En línea]. Available:
<https://www.upgrad.com/blog/multinomial-naive-bayes-explained/>. [Último acceso: 4 7 2022].
- [19 S. Patwardhan, «KNN Algorithm | What is KNN Algorithm | How does KNN Function,»
] Analyticsvidhya, 21 4 2021. [En línea]. Available:
<https://www.analyticsvidhya.com/blog/2021/04/simple-understanding-and-implementation-of-knn-algorithm/>. [Último acceso: 4 7 2022].
- [20 «Figure 11. K-Nearest Neighbor (KNN) classification principle.,» ResearchGate, [En
] línea]. Available: https://www.researchgate.net/figure/K-Nearest-Neighbor-KNN-classification-principle_fig4_343080916. [Último acceso: 4 7 2022].
- [21 H. Marius, «Tree algorithms explained: Ball Tree Algorithm vs. KD Tree vs. Brute Force,»
] Towardsdatascience, 10 12 2021. [En línea]. Available:
<https://towardsdatascience.com/tree-algorithms-explained-ball-tree-algorithm-vs-kd-tree-vs-brute-force-9746debcd940>. [Último acceso: 4 7 2022].
- [22 J. Brownlee, «A Gentle Introduction to Sparse Matrices for Machine Learning,» Machine
] Learning Mastery, 13 3 2018. [En línea]. Available:
<https://machinelearningmastery.com/sparse-matrices-for-machine-learning/>. [Último acceso: 4 7 2022].
- [23 «Distancia euclidiana,» Wikipedia, 7 6 2022. [En línea]. Available:
] https://es.wikipedia.org/w/index.php?title=Distancia_euclidiana&oldid=144036433.
[Último acceso: 4 7 2022].

- [24 «Taxicab geometry,» Wikipedia, 10 6 2022. [En línea]. Available:
] https://en.wikipedia.org/w/index.php?title=Taxicab_geometry&oldid=1092514385.
[Último acceso: 4 7 2022].
- [25 «Distancia de Minkowski,» Interactive Chaos, [En línea]. Available:
] <https://interactivechaos.com/es/manual/tutorial-de-machine-learning/distancia-de-minkowski>. [Último acceso: 4 7 2022].
- [26 P. Gupta, «Decision Trees in Machine Learning,» Towardsdatascience, 12 11 2017. [En
] línea]. Available: <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052>. [Último acceso: 4 7 2022].
- [27 Simplilearn, «Random Forest Algorithm - Random Forest Explained | Random Forest in
] Machine Learning | Simplilearn,» YouTube, 12 3 2018. [En línea]. Available:
<https://www.youtube.com/watch?v=eM4uJ6XGnSM>. [Último acceso: 4 7 2022].
- [28 P. Aznar, «Decision Trees: Gini vs Entropy,» Quantdare, 2 12 2020. [En línea]. Available:
] <https://quantdare.com/decision-trees-gini-vs-entropy/>. [Último acceso: 4 7 2022].
- [29 DANB, «What is Log Loss?,» Kaggle, 19 4 2018. [En línea]. Available:
] <https://kaggle.com/code/dansbecker/what-is-log-loss>. [Último acceso: 4 7 2022].
- [30 W. Monroe, «Logistic regression,» *Lecture Notes*, nº 22, 2017.
]
- [31 Conecta Software, «Técnicas de data mining - Regresión logística,» Conecta Software, 2 3
] 2020. [En línea]. Available: <https://conectasoftware.com/analytics/tecnicas-de-data-mining-regresion-logistica/>. [Último acceso: 4 7 2022].
- [32 L. Newman, «ML From Scratch: Logistic and Softmax Regression,» Towardsdatascience,
] 9 8 2021. [En línea]. Available: <https://towardsdatascience.com/ml-from-scratch-logistic-and-softmax-regression-9f09f49a852c>. [Último acceso: 4 7 2022].
- [33 J. Brownlee, «One-vs-Rest and One-vs-One for Multi-Class Classification,» Machine
] Learning Mastery, 12 4 2020. [En línea]. Available:
<https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/>. [Último acceso: 4 7 2022].
- [34 Javatpoint, «Complement Naive Bayes (CNB) Algorithm,» Javatpoint, [En línea].
] Available: <https://www.javatpoint.com/complement-naive-bayes-algorithm>. [Último
acceso: 4 7 2022].
- [35 Colah, «Understanding LSTM Networks,» Colah's blog, 27 8 2015. [En línea]. Available:
] <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. [Último acceso: 4 7 2022].
- [36 hH1sG0n3, «Answer to "Difference between LSTM cell state and hidden state",» Stack
] Exchange, 23 10 2020. [En línea]. Available:
<https://datascience.stackexchange.com/questions/82808/difference-between-lstm-cell-state-and-hidden-state>. [Último acceso: 4 7 2022].

- [37 N. C. M. R. OSTP, «COVID-19 Open Research Dataset Challenge (CORD-19),» Kaggle,
] 16 3 2020. [En línea]. Available:
<https://www.kaggle.com/datasets/08dd9ead3afd4f61ef246bfd6aee098765a19d9f6dbf514f0142965748be859b>. [Último acceso: 4 7 2022].
- [38 L. L. W. et al., «CORD-19: The COVID-19 Open Research Dataset,» arXiv, 10 7 2020.
] [En línea]. Available: <http://arxiv.org/abs/2004.10706>. [Último acceso: 4 7 2022].
- [39 NLTK, «Source code for nltk.stem.porter,» NLTK Project, 25 3 2022. [En línea].
] Available: https://www.nltk.org/_modules/nltk/stem/porter.html. [Último acceso: 4 7 2022].
- [40 P. Hart, «he condensed nearest neighbor rule (corresp.),» *IEEE transactions on*
] *information theory*, vol. 14, nº 3, pp. 515-516, 1968.
- [41 D. L. Wilson, «Asymptotic properties of nearest neighbor rules using edited data,» *IEEE*
] *Transactions on Systems, Man, and Cybernetics*, pp. 408-421, 1972.
- [42 I. Tomek, «An experiment with the edited nearest-neighbor rule,» *IEEE Transactions on*
] *systems, Man, and Cybernetics*, vol. 6, nº 6, pp. 448-452, 1976.
- [43 I. Tomek, «Two modifications of CNN,» *IEEE Trans. Systems, Man and Cybernetics*, vol.
] 6, nº 6, pp. 769-772, 1976.
- [44 K. W. B. L. O. H. a. W. P. K. Nitesh V Chawla, «Smote: synthetic minority over-sampling
] technique,» *Journal of artificial intelligence research*, vol. 16, pp. 321-357, 2002.
- [45 Y. B. E. A. G. a. S. L. A. Haibo He, «Adaptive synthetic sampling approach for
] imbalanced learning,» *IEEE International Joint Conference on Neural Networks*, pp. 1322-1328, 2008.
- [46 Wikipedia, «Mean reciprocal rank,» Wikipedia, 15 6 2021. [En línea]. Available:
] https://en.wikipedia.org/w/index.php?title=Mean_reciprocal_rank&oldid=1028668019.
[Último acceso: 4 7 2022].
- [47 N. Craswell, «Mean Reciprocal Rank,» de *Encyclopedia of Database Systems*, Boston,
] MA, Springer US, 2009, pp. 703-1703.
- [48 SKLearn, «3.3. Metrics and scoring: quantifying the quality of predictions,» SKLearn, [En
] línea]. Available: https://scikit-learn/stable/modules/model_evaluation.html. [Último
acceso: 4 7 2022].
- [49 Yahya, «Answer to "Logistic regression python solvers' definitions",» Stack Overflow, 18
] 9 2018. [En línea]. Available: <https://stackoverflow.com/a/52388406>. [Último acceso: 4 7 2022].
- [50 R. D. Peng, «Coordinate Descent,» Bookdown, [En línea]. Available:
] <https://bookdown.org/rdpeng/advstatcomp/coordinate-descent.html>. [Último acceso: 4 7 2022].

Recomendación de revistas científicas
utilizando técnicas de aprendizaje automático

- [51 Kassambara, «Penalized Logistic Regression Essentials in R: Ridge, Lasso and Elastic Net,» STHDA, 11 3 2018. [En línea]. Available: <http://www.sthda.com/english/articles/36-classification-methods-essentials/149-penalized-logistic-regression-essentials-in-r-ridge-lasso-and-elastic-net/>. [Último acceso: 4 7 2022].
- [52 K. Team, «Keras documentation: Embedding layer,» Keras, [En línea]. Available: https://keras.io/api/layers/core_layers/embedding/. [Último acceso: 4 7 2022].
- [53 K. Team, «Keras documentation: LSTM layer,» Keras, [En línea]. Available: https://keras.io/api/layers/recurrent_layers/lstm/. [Último acceso: 4 7 2022].
- [54 E. Zvornicanin, «Differences Between Bidirectional and Unidirectional LSTM,» Baeldung, 5 2 2022. [En línea]. Available: <https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm>. [Último acceso: 4 7 2022].
- [55 J. Brownlee, «How to Choose an Activation Function for Deep Learning,» Machine Learning Mastery, 17 1 2021. [En línea]. Available: <https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/>. [Último acceso: 4 7 2022].
- [56 Ediciones eni, «Funciones de pérdida (Loss function),» Ediciones eni, [En línea]. Available: <https://www.ediciones-eni.com/open/mediabook.aspx?idR=8dd2ca32769cb24b49648b15ef8e777e>. [Último acceso: 4 7 2022].
- [57 Mustafa, «Optimizers in Deep Learning,» Medium, 12 2 2022. [En línea]. Available: <https://medium.com/mlarning-ai/optimizers-in-deep-learning-7bf81fed78a0>. [Último acceso: 4 7 2022].
- [58 Scikit-learn, «6.2.3.4. Tf-idf term weighting,» [En línea]. Available: https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction.