Trabajo de Fin de Máster

Estudio de técnicas de inteligencia artificial aplicadas a datos recopilados en Twitter

Study of artificial intelligence techniques applied to data collected on Twitter

Manuel Alexander Arzola Santos

La Laguna, 6 de julio de 2023

Dña. **María Belén Melián Batista**, con N.I.F. 44.311.040-E Caterática de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

D. **J. Marcos Moreno Vega**, con N.I.F. 42.841.047-M Catedrático de Universidad adscrito al Departamento deIngeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Estudio de técnicas de inteligencia artificial aplicadas a datos recopilados en Twitter*”

ha sido realizada bajo su dirección por D. **Manuel Alexander Arzola Santos**, con N.I.F. 45.865.120-S.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 6 de julio de 2023

Agradecimientos

Agradezco a mi familia por todo el apoyo recibido y la comprensión mostrada durante todo el tiempo que ha durado el máster, especialmente en el tramo final.

No quiero olvidarme de mis amigos, colegas y cómplices en la sombra, aquellos que no son de mi familia pero han estado ahí todo el tiempo y me han ayudado de una forma u otra.

Sería injusto no agradecer también a mis tutores, María Belén Melián Batista y J. Marcos Moreno Vega, por estar ahí hasta el final.

Licencia

Resumen

En el presente TFM se trata el tema de la obtención de datos en redes sociales, su tratamiento y la extracción de conocimiento mediante diversos métodos. La red social elegida es Twitter, y el tipo de dato con el que se trabajará mayoritariamente, tuits pertenecientes a diversos usuarios. Se mostrará el criterio seguido a la hora de conseguir los datos, de acuerdo al objetivo que se pretende lograr, así como todos los pasos seguidos para la recopilación de datos y su tratamiento y limpieza.

Los métodos estudiados incluyen sistemas recomendadores basados en contenido, por una parte mediante el uso del valor TF-IDF asociado a cada palabra y por otra parte haciendo uso del modelo Word2Vec, cuya arquitectura emplea redes neuronales, tocando así el deep learning.

Se empleará también el machine learning, pues se usarán algoritmos de clasificación supervisada para poder clasificar usuarios de acuerdo a temas de interés para el usuario.

Por último, se realizará un análisis de sentimientos sobre parte de los tuits recogidos, midiendo la positividad o negatividad de los tuits, observando su evolución temporal, la actividad del usuario, porcentaje de tuits negativos, positivos o neutros y empleando también algoritmos de clasificación que permitan etiquetar a los tuits como positivos o negativos.

Palabras clave: Twitter, sistemas de recomendación, Word2Vec, algoritmos de clasificación

Abstract

This TFM deals with the issue of obtaining data on social networks, their treatment and the extraction of knowledge from the data through various methods. The social network chosen is Twitter, and the type of data with which we will mainly work is tweets belonging to various users. The criteria followed when obtaining the data will be shown, according to the objective to be achieved, as well as all the steps followed for data collection and its treatment and cleaning.

The studied methods include content-based recommendation systems. On the one hand by using the TF-IDF value associated with each word and on the other hand making use of the Word2Vec model, whose architecture uses neural networks, that are part of deep learning.

Machine learning will also be used, since supervised classification algorithms will be used to classify users according to topics of interest to the user.

Finally, a sentiment analysis will be carried out on part of the collected tweets, measuring the positivity or negativity of the tweets, observing their temporal evolution, user activity, percentage of negative, positive or neutral tweets and also using classification algorithms that allow tweets to be labeled as positives or negatives.

Keywords: Twitter, Recommendation systems, Word2Vec, classification algorithms

Índice general

Índice de Figuras

Capítulo 1

Introducción

La época contemporánea se caracteriza sobre todo por las ingentes cantidades de información de diferentes tipos que generan los individuos, empresas, gobiernos, etc. Con el desarrollo de la radio y la televisión como precedentes de transmisión de la información, es con la llegada de internet que se da la explosión que permite que todo aquel con acceso a la red pueda expresar sus opiniones y generar sus propios datos. No en vano se llama a esta era la era digital o de la información.

Dentro del ecosistema que es internet, las redes sociales constituyen una valiosa fuente de información, pues es donde se deposita tanto la información personal como las opiniones de los usuarios, y en suma, el sentir colectivo acerca de los diferentes temas de actualidad, dependiendo del formato de la red social. Algunas de las más famosas son Youtube, Facebook o Instagram, pero una que llama poderosamente la atención es Twitter.

Twitter es una red social denominada de microblogueo, lo cual quiere decir que la actividad principal es el intercambio de mensajes cortos. El límite de dichos mensajes es de 140 caracteres, lo cual limita a la hora de expresar opiniones demasiado complejas, pero que es especialmente útil para expresar con inmediatez y de forma concisa opiniones sobre los eventos diarios.

Dichos eventos pueden referirse a experiencias recibidas por diferentes establecimientos, a interacciones con otras personas o empresas, opiniones políticas, o bien de diferentes productos o servicios. Resulta obvio que si existiese una forma de analizar dichos datos sería realmente provechoso para las empresas de todos los tamaños así como para los gobiernos.

Cabe entonces hacerse la siguiente pregunta: ¿pueden las distintas herramientas del campo de la inteligencia artificial ayudar a extraer conocimiento de toda esa información?

La respuesta es un rotundo sí. Por ejemplo, se pueden crear sistemas de recomendación de usuarios que permita a las empresas medir la similitud entre usuarios, bien para el servicio de recomendar usuarios al usuario en concreto, bien para conocer qué productos recomendar, como puede verse en Weng et al. (2023). De igual manera, se pueden analizar las tendencias de las personas como en Błajda et al. (2023), por mostrar tan sólo algunas posibilidades.

Para profundizar en la respuesta y en algunas de las posibilidades se han definido varios objetivos en este trabajo, los cuales se detallan en la siguiente sección. De igual forma, también se habla someramente de la estructura del presente trabajo en la última sección de este capítulo.

1.1. Objetivos

El objetivo de este TFM es estudiar la aplicación de diferentes herramientas en el ámbito de la inteligencia artificial. Partiendo desde la obtención de los datos, su tratamiento y preprocesado para su posterior utilización y el empleo de diferentes métodos de los ámbitos del *machine learning* y *deep learning* para poder así extraer conocimiento.

Puesto que son varios pasos, se puede dividir el objetivo global en los siguientes objetivos específicos:

- Recopilar datos de *Twitter*, así como tratarlos y limpiarlos para poder ser usados apropiadamente.
- Crear un sistema recomendador basado en contenido empleando el concepto de *Term Frequency - Inverse Document Frequency* (TF-IDF).
- Usar algoritmos de aprendizaje supervisado para clasificar usuarios en base a temas de su interés.
- Crear un sistema recomendador basado en contenido usando el modelo *Word2Vec*.
- Realizar un análisis de sentimientos.

1.2. Estructura

Para la consecución de los objetivos previamente comentados, se ha estructurado este trabajo en diferentes capítulos, en cada uno de los cuales se desarrollan los diferentes conceptos clave necesarios para comprender los procesos realizados y para detallar dichos procesos.

- En el capítulo 2 se trata la recolección de los datos de la red social *Twitter*, pormenorizando el uso de su API y también de la librería *Tweepy*, que permite interactuar con la API mediante el lenguaje *Python*. Se detallan además los pasos y el criterio seguido a la hora de descargar los datos, así como los diferentes problemas surgidos y su solución.

  Debido a que al final de este capítulo se guardarán los datos que se usarán en el resto de capítulos, se limpiará el texto, eliminando mayúsculas, símbolos, signos de puntuación, emojis, largas cadenas de texto, etc; en resumen, elementos que cumplen una función en el lenguaje natural, el que usamos los humanos, pero que afectan negativamente a la hora de ser procesados por un ordenador.

- En el capítulo 3 se habla de los diferentes sistemas recomendadores, sus características y problemas principales, y debido a las herramientas empleadas se elige usar uno basado en contenido. Se basará este sistema en los conceptos de *Term Frequency* (TF), *Inverse Document Frequency* (IDF), *Term Frequency - Inverse Document Frequency* (TF-IDF) y el método de similitud del coseno.

- El capítulo 4 versa acerca del uso de distintos algoritmos de aprendizaje supervisado para clasificar usuarios en base a dos temas de interés. Previamente a ello se comentan las particularidades de los algoritmos y también del análisis de componentes principales, un método de reducción de la dimensionalidad. También se habla del preprocesado realizado y se muestran los resultados obtenidos por cada algoritmo.

- En el capítulo 5 se elabora un sistema recomendador basado en contenido, pero esta vez en base al modelo *Word2Vec*, que pertenece al ámbito del *deep learning*. Para ello se empieza detallando qué es la neurona artificial como unidad funcional de las redes neuronales, cómo se organizan dichas redes, los tipos de red neural usados y las arquitecturas que conforman el modelo. Se muestra también la representación de las palabras como nube de puntos y los resultados del sistema recomendador.

- El capítulo 6 trata del análisis de sentimientos. Para ello se explica qué es un sentiment lexicon, el cual permite asignar sentimientos positivos o negativos a diferentes palabras dentro de un tuit, otorgando así un valor numérico que exprese el sentimiento de dicho tuit. También se hace un análisis exploratorio de los datos, donde se muestra el porcentaje de tuits positivos, negativos y neutros de dos usuarios, cada uno afín a cada uno de los dos temas seleccionados. En el susodicho análisis también se muestra la tendencia de la evolución temporal de publicaciones y sentimientos por tuit para cada uno de los dos usuarios. Además, se usan los algoritmos y métodos descritos en el capítulo 4 para clasificar tuits de acuerdo a si son positivos o negativos.

- Por último, en los capítulos 7 y 8 se analizan los resultados obtenidos y se proponen líneas de trabajo futuras en español e inglés, respectivamente.

Capítulo 2

Recolección de datos

Los datos que aparecen en el TFM se han recopilado específicamente para este estudio, sin recurrir a un dataset previamente realizado y preprocesado. Se ha buscado el contenido deseado de la red social Twitter mediante su API. Se explicarán los conceptos clave y luego la parte práctica.

2.1. API

El término API es el acrónimo de Application Programming Interfaces, que en español significa interfaz de programación de aplicaciones. Se trata de un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones a través de un conjunto de reglas.

Así pues, podríamos definir una API como una especificación formal que establece cómo un módulo de un software se comunica o interactúa con otro para cumplir determinadas funciones.

Twitter permite realizar diferentes acciones con su API, dependiendo del *endpoint* que se utilice:

- Búsquedas de contenido
- Actividad de una cuenta
- Mensajes directos
- Contenido en tiempo real
- Anuncios

La API de Twitter cuenta con diferentes servicios y suscripciones: esencial, elevada, investigación académica, etc. Con cada uno de ellos se puede acceder a diferentes *endpoints*. Para este proyecto he usado el nivel de acceso esencial, el cual es gratuito y que puede emplear cualquier usuario, sin ningún tipo de requisito. Sin embargo, todos los niveles tienen limitaciones de diferentes tipos, y en el caso del nivel esencial, las relevantes para nosotros son las siguientes:

- Posibilidad de crear 1 único proyecto
- 1 aplicación por proyecto

- Se puede descargar un máximo de 500.000 tuits mensuales por cada aplicación
- Acceso a los tuits de tan sólo los 7 últimos días

Los proyectos permiten organizar el trabajo en función de cómo se pretenda usar la API de Twitter para que poder administrar de manera efectiva el acceso a la misma y monitorear su uso. Cada proyecto puede contener una o varias aplicaciones según su nivel de acceso. Se usarán estas aplicaciones para generar credenciales de autenticación, como claves y secretos de API, *tokens* de acceso de usuario y *token* de acceso a la aplicación.

Además, se ha de tener en cuenta que el 12 de Agosto de 2020 salió la versión 2 de la API de Twitter, la cual cambia el cómo los usuarios y las aplicaciones se relacionan con ella, de tal forma que los métodos de autenticación y las limitaciones que tiene cada nivel de acceso cambian con respecto a las versiones anteriores; de hecho, el nivel esencial sólo permite acceder de manera efectiva para nuestros fines a la versión 2 de la API, por lo que me centraré en dicha versión.

Un concepto que debemos tener en cuenta es el de requests o solicitudes, las cuales también están limitadas en esta versión de la API a cierta cantidad por cada 15 minutos. Por defecto, 1 solicitud nos devuelve 10 tweets, cantidad ampliable hasta a 100 tweets por solicitud.

Esta versión de la API establece límites más generosos en cuanto a solicitudes si nos autenticamos mediante la aplicación de nuestro proyecto que como usuario, por lo que el método elegido para la autenticación en la API será el de la aplicación.

Teniendo en cuenta cada uno de los criterios anteriormente mencionados, los pasos necesarios para poner a punto el entorno que nos permita obtener los datos será el siguiente:

- Registrarse en Twitter
- Darse de alta en la plataforma para desarrolladores de Twitter
- Crear una app
- Crear los tokens de autenticación necesarios

Debido a que nos autenticaremos mediante la app, el token que debemos generar será el llamado App only Access Token, o Bearer Token. Una vez realizados estos pasos, estaremos listos para acceder a la API.

2.2. Tweepy

A pesar de que podríamos usar directamente la API de Twitter haciendo las solicitudes a las url de sus diferentes *endpoints*, emplearemos la librería de Python Tweepy, la cual cuenta con métodos para acceder a cada uno de los *endpoints* de la API, y cuyo uso resulta más sencillo y llevadero que las solicitudes directas.

Hay que tener en consideración que sólo se cuenta con soporte para la versión 2 de la API de Twitter de la versión 4.0 de Tweepy en adelante. En este proyecto se ha usado la versión 4.4 de Tweepy.

Un concepto clave a la hora de recopilar datos en Twitter es el de la paginación. Es una función de los *endpoint* de la v2 de la API de Twitter que devuelve más resultados

de los que se pueden devolver en una sola respuesta. Cuando eso sucede, los datos se devuelven en una serie de 'páginas'. La paginación se refiere a los métodos para solicitar programáticamente todas las páginas, con el fin de recuperar todo el conjunto de datos de resultados.

Los distintos métodos de Tweepy usados son los siguientes:

- client: nos permite autenticarnos en la API.
- search_recent_tweets: El *endpoint* de búsqueda reciente devuelve tuits de los últimos siete días que coinciden con un término o consulta a nuestra elección. El límite de *request* de esta función es de 450.
- get_users_tweets: Devuelve tuits escritos por un solo usuario, especificado por el ID de usuario solicitado. De forma predeterminada, se devuelven los diez tuits más recientes por solicitud. Mediante la paginación, se pueden recuperar hasta los 3200 tuits más recientes. El límite de *request* de esta función es de 1500.
- paginator: Permite la paginación. Se le pasa el método que queremos emplear, en caso de que se pueda, y distintos parámetros; como fechas inicial y final entre las que queramos recopilar los tuits, la posibilidad de excluir retuits y *replies*, el máximo de tuits que queremos, etc. Cabe destacar que cada límite establecido es un máximo solicitado, Twitter puede enviar menos datos.

Debido a las limitaciones que nos impone el tener que usar la versión esencial de la API, la metodología seguida para la recolección de datos ha sido la siguiente:

1. Seleccionar dos temas lo suficientemente diferentes entre sí en base a los cuales se clasificará a los usuarios o bien se comprobará si la recomendación de usuarios ha sido aceptable. Los dos temas seleccionados han sido "Xbox", la popular consola de videojuegos, y "War", guerra, tema tristemente célebre estos días. El lenguaje en el que se han buscado los tuits para cada uno de los temas ha sido el inglés.
2. Para cada uno de esos temas, descargar el número de id y el nombre de usuario de los últimos 1000 tuits publicados.
3. Se comprueba que no haya coincidencias entre los usuarios seleccionados de ambos temas, para que no se repitan. En caso de haber coincidencias, se eliminan de ambas listas de usuarios, por considerarlos afines a ambos temas.
4. Usando la función get_users_tweets, se descargan los últimos 1000 tuits publicado por cada uno de esos usuarios, excluyendo retuits y *replies*, para quedarnos con la información relevante de cada usuario. Esto será hecho creando un archivo por usuario, que contenga sólo los tuits del usuario en concreto.
5. Se limpia el texto.
6. Cuanta más información se tenga disponible del usuario, mejores serán las tareas de clasificación y recomendación, por lo que se hace un filtrado para obtener aquellos usuarios de los que haya descargado un número de tuits en inglés mayor a 400.
7. Se obtiene el nombre y username para cada uno de los usuarios que quedan al final de todo este proceso.

Hay un problema que surgió a la hora de confeccionar el conjunto de datos: la presencia de tuits en diferentes idiomas. El inglés fue el idioma escogido, pero debido a que es el idioma preponderante en internet, se da que la lengua materna de muchos usuarios puede ser distinta, pero hayan publicado tuits sobre estos temas en inglés. Ya que se han elegido 1000 tuits de cada uno de los temas para hallar así la id de los usuarios, es de suponer que el tuit por el que se eligió la id estaba casualmente escrito en inglés, pero que no toda la actividad del usuario sea en el mismo idioma.

Existe una gran variedad de librerías y APIs que permiten traducir texto, como por ejemplo Google Cloud Translate, la API oficial del traductor de Google, y DeepL. Google Cloud translate no tiene acceso gratuito a la API, mientras que DeepL sí tiene una versión gratuita. Sin embargo, tiene un límite de 500.000 caracteres traducibles al mes. Esto hace que si, por ejemplo, la cantidad total de caracteres de todos los tuits de un usuario sea de 70.000, tardaríamos aproximadamente 6 años en traducir los tuits de todos los usuarios que hemos recopilado. Ambas opciones quedan descartadas.

En cuanto a las posibilidades gratuitas, la más destacada es Googletrans, una API no oficial que recurre al traductor de Google. Sin embargo, el problema es el gran volumen de texto a traducir, ya que al superar el límite de solicitudes hechos al *endpoint* de google, este nos arroja un error y requiere de mucho tiempo de espera antes de poder solicitar la siguiente traducción.

Existen otras alternativas, como Goslate, Py-translate o TextBlob, pero todas ellas se apoyan en el mismo endpoint que Googletrans, por lo que no constituyen alternativas en realidad. El resto de proyectos que se han encontrado en la red, o bien no tienen detección automática de idioma, algo fundamental ya que a priori no sabemos el idioma del que se deberá traducir al inglés, o bien están ya descontinuados y obsoletos

Una opción que podría solucionar ambos problemas sería detectar los tuits escritos en un idioma diferente al inglés y eliminarlos, pero para ello también hay que mandar una solicitud por tuit usando cualquiera de las librerías anteriores, lo que ocasionaría el mismo problema que para la traducción.

Sin embargo, se descubrió que al descargar los tuits de cada usuario uno de los parámetros que puede devolver Tweepy es el idioma en el que está el tuit, por lo que a la hora de descargarlos se etiquetará a cada uno de los tuits con el idioma que le corresponde según twitter.

Otro parámetro que se ha descargado de cada uno de los tuits es la fecha en la que fue publicado, la cual nos será útil en la sección de análisis de sentimientos.

2.3. Limpieza de texto

Tras recopilar los tuits, se hace necesario preprocesar los datos limpiando el texto, para así eliminar palabras, expresiones o símbolos que no aporten significado alguno. A pesar de que existen varias librerías dedicadas a tal fin, como por ejemplo NLTK (Natural Language ToolKit), se ha preferido hacerlo combinando varias librerías así como expresiones regulares, debido la flexibilidad que permite el ir afrontando los distintos problemas que surgían, uno detrás de otro.

Las distintas operaciones realizadas para ello fueron las siguientes:

- Pasar todo el texto a minúsculas.
- Eliminación de todas las páginas web.

- Eliminación de los signos de puntuación y símbolos tales como €@#+...
- Eliminación de los números.
- Eliminación de los espacios en blanco múltiples.
- Eliminación de hashtags y menciones a usuarios.
- Eliminación de direcciones de email
- Eliminación de los emojis presentes.
- Eliminación de largas cadenas de texto.

Para la mayoría de las operaciones mencionadas más arriba he usado la librería de código abierto cleantext, sobre todo porque permite eliminar emojis y también debido a su facilidad de uso.

Con respecto a la eliminación de largas cadenas de texto, he usado la función de la librería NLTK nltk.tokenize.casual.reduce_lengthening(), la cual permite reemplazar las secuencias de caracteres repetidas de 3 o más de longitud con secuencias de 3 de longitud. Por ejemplo, si tenemos la secuencia “jaaaaa”, la reemplazará por “jaaa”. Esto es especialmente útil con texto proveniente de redes sociales, ya que es común que los usuarios empleen más letras de las necesarias para enfatizar la emoción a la que hace referencia la palabra.

En el caso de los emails y las menciones a usuarios surge un problema, ya que ambas estructuras comparten el símbolo “@”. Las menciones tienen la estructura "@usuario", mientras que la de los emails es "email@email.com". Por ello, si se eliminan las cadenas que empiecen por "@", se borrarán las menciones pero también la parte final de los email, dejando medio email sin borrar. Por ello primero se eliminan los emails usando la librería clean-text y luego las menciones usando expresiones regulares.

Ocurre lo mismo con el símbolo "#", que es con el que empiezan los hashtags. Al eliminar todos los símbolos y signos de puntuación quedaría el hashtag como palabra, lo cual no aporta demasiado al ser generalmente los hashtag conjunciones de palabras o palabras inventadas. Por lo que se eliminan primero todas las cadenas que empiecen con dicho símbolo mediante expresiones regulares y luego todos los simbolos.

Los pasos realizados han sido los siguientes:

1. Eliminación de emails
2. Eliminación de hashtags, menciones a usuarios y largas cadenas de texto
3. Realización del resto de operaciones anteriormente mencionadas

Lo que no se realizará en esta fase de limpieza del texto es el tokenizado del texto, esto es, el proceso de tokenizar o dividir una cadena de texto en una lista de *tokens*, siendo constituido cada *token* por cada una de las diferentes palabras del documento. Esto se debe a que, por ejemplo, en el sistema recomendador mediante TF-IDF no es necesario presentar el texto *tokenizado*, mientras que para el sistema recomendador mediante Word2Vec la librería Gensim permite un *tokenizado* sencillo previamente al entrenado del modelo.

Capítulo 3

Sistema recomendador basado en contenido

3.1. Marco teórico

Existe una gran variedad de sistemas recomendadores:

1. **Sistemas recomendadores colaborativos:** aprovechan la información sobre el comportamiento pasado o las opiniones de una comunidad de usuarios existente para predecir en qué artículos es probable que esté interesado el usuario actual del sistema. Este sistema no necesita saber nada del producto para ser recomendado. Puede estar basado en el usuario, en el cual se identifica a usuarios que hayan tenido preferencias similares con respecto al usuario activo o bien basado en el ítem, en el cual no se busca la similitud entre los usuarios, sino entre los propios productos en base a valoraciones realizadas por los usuarios. La principal ventaja es que se evita al sistema la pesada tarea de proporcionar descripciones detalladas del producto. Sin embargo, con este enfoque no es posible seleccionar productos basados en sus características y en las preferencias específicas de un usuario. Otro problema asociado a este sistema es el de arranque en frío, ya que si no existe una comunidad de usuarios o de valoraciones de un producto, es decir, si hay escasez de datos, no se podrá efectuar ninguna recomendación. Una posibilidad para resolver este problema es la de recabar datos adicionales, pero de esta forma se perdería el enfoque colaborativo puro.

2. **Sistemas recomendadores basados en contenido:** en este sistema se determinarán los ítems que mejor se ajusten a las preferencias del usuario. En este caso, el contenido al que hace referencia el nombre del sistema consistiría en la descripción de las características del ítem dado. Por ejemplo, el usuario indicará sus preferencias, y en base a la descripción del producto, la cual puede proporcionar el propio fabricante, se efectuará la recomendación. En este caso, se evita el anteriormente comentado problema de arranque en frío pues no necesitamos la existencia previa de una comunidad o de un historial de valoraciones. Sin embargo, también se deberá procurar al sistema una descripción detallada de las características del producto.

3. **Sistemas recomendadores basados en conocimiento:** este sistema responde a los problemas de los dos sistemas anteriores para situaciones en que los sistemas no pueden actuar de forma adecuada, por ejemplo, cuando existen productos con bajo número de valoraciones disponibles, o cuando los clientes quieren definir explícitamente sus requerimientos. Esto se debe a que no existe el problema de arranque en frío y las recomendaciones no se hacen en base a las valoraciones de los

usuarios. Las recomendaciones se hacen en forma de similitudes entre los requisitos del cliente y los ítems o en base a reglas explícitas de recomendación.

4. **Sistemas recomendadores híbridos:** enfoque que combina las técnicas anteriores de distintas formas. Puede basarse en modelos de conocimiento, en el perfil del usuario, en características del producto, etc.

Conceptualmente, surge la duda de cómo denominar al sistema recomendador a usar, puesto que comparte características tanto de los sistemas colaborativos como de los basados en contenido. Esto es debido a que:

- Existe el problema de arranque en frío, ya que la existencia de tuits no depende de una descripción emitida por un fabricante, sino que son los propios usuarios los que publican sus tuits. Si un usuario tiene pocos tuits o no tiene, no podrá ser recomendable.
- Las recomendaciones serán efectuadas basadas en las características propias de cada usuario, ya que lo que se toma como descripción del producto es la forma de expresarse, las palabras que usa y los temas sobre los que habla el usuario. Se da que el producto y el usuario son la misma cosa.

En base a lo previamente comentado, podría pensar que lo que se usará será un sistema recomendador híbrido, ya que sin una comunidad de Twitter activa no sería posible recabar estos datos, pero como las técnicas empleadas son puramente de sistemas recomendadores basados en contenido, ese es el nombre que va a recibir.

Para efectuar la recomendación se medirá la similitud entre usuarios. De igual manera, la similitud comprobada será en base al contenido de cada usuario, mediante una matriz *Term Frequency - Inverse Document Frequency* (TF-IDF).

En un sistema recomendador basado en contenido, se llamaría documento a la descripción con las características del producto, y en el caso que nos ocupa, los documentos con los que se trabajará son líneas de texto que hacen referencia a las *timelines* de los diferentes usuarios a recomendar. Estos serán el contenido en el que se basará la recomendación.

Por ello, se hace necesario tratar el texto de alguna forma, y para ello se obtendrán los vectores TF-IDF, que serán los que se usarán para hallar la similitud entre documentos.

Esta operación la realizará una función de la librería Scikit-Learn, la función *TfidfVectorizer()*; en este paso es donde realizamos el paso de eliminación de *stopwords*.

Las *stopwords* son palabras muy comunes en el idioma, como árticulos, etc, que no aportan significado al texto. Para eliminarlas se hace uso de la librería NLTK, que incluye conjuntos de *stopwords* para diferentes idiomas, siendo el inglés el usado en este estudio.

A la función *TfidfVectorizer()* se le puede pasar como argumento una lista de *stopwords*, para que las elimine a la hora de crear la matriz TF-IDF,

La susodicha función retornará una serie de vectores, uno por cada documento, cuyos elementos serán cada una de las diferentes palabras del total del texto, y cuyos valores serán el peso TF-IDF asociado a cada palabra.

Sin embargo, antes deben ser explicados los conceptos TF e IDF para comprender en base a qué se obtiene la similitud. Ya que para un ordenador es imposible comprender palabras, se hace necesario que las palabras sean traducidas a valores numéricos con los que pueda operar matemáticamente.

3.1.1. TF (Term frequency, o frecuencia del término)

La frecuencia del término hace referencia a la cantidad de veces que aparece una palabra en un texto.

Es una forma de normalizar el texto que evita que los textos largos adquieran una mayor importancia que los cortos, ya que usando esta métrica no importará la longitud del documentos, sino que se tendrá un vector multidimensional, con tantas dimensiones como términos, en el que aparecerá la frecuencia de las diferentes palabras del documento.

La fórmula para hallar el valor TF de un determinado término es la siguiente:

$$TF(i,j) = \frac{freq(i,j)}{maxOther(i,j)} \tag{3.1}$$

Donde freq(i, j) es el número de ocurrencias del término i en el documento, y maxOthers(i,j) el número de términos diferentes que hay en el documento.

3.1.2. IDF (Inverse document frequency, o frecuencia inversa del documento)

La idea detrás de la frecuencia inversa del documento es reducir el peso de las palabras que aparecen más a menudo en todos los documentos. Esto es debido a que, por lo general, las palabras que más se repiten no son de tanta utilidad como las que menos a la hora de discriminar entre documentos, por lo que se asignarán pesos diferentes a las palabras en base a ello.

Existen varias fórmulas diferentes para calcular IDF, pero la que usa internamente la función *TfidfVectorizer()* es la siguiente:

$$IDF(i) = log\left(\frac{1+n}{1+df(i)}\right) + 1 \tag{3.2}$$

Donde n es el número total de documentos del conjunto de documentos y df(i) es el número de documentos del conjunto de documentos que contienen el término i.

3.1.3. TF-IDF

De hallar los valores TF se tendrá como resultado una matriz de valores, mientras que se obtendrá un vector de valores IDF. Multiplicando vectorialmente:

$$TF - IDF(i,j) = TF(i,j) * IDF(i) \tag{3.3}$$

De esta manera se obtienen los valores TF-IDF, los cuales sirven para cuantificar la relevancia de una palabra dentro de un determinado documento, ya que se contabiliza la frecuencia de las palabras con el componente TF (Term Frequency) y se reduce el peso de las más frecuentes gracias al término IDF (Inverse Document Frequency). Así se beneficia a las palabras que aparezcan menos, las cuales presumiblemente tendrán mucha más relevancia en los distintos documentos.

Puesto que se realiza un producto vectorial de vector por matriz, el resultado será una matriz de valores TF-IDF, en la cual cada una de las filas será el vector de pesos TF-IDF para cada uno de los documentos. De esta manera, una lista de frases se habrá convertido en una serie de valores numéricos con los que podremos trabajar para hallar la similitud entre documentos.

3.1.4. Método de similitud del coseno

Una vez hallados los valores TF-IDF usando *TfidfVectorizer()*, se tendrá como resultado un vector por documento, con tantas dimensiones como palabras diferentes haya en la totalidad de los documentos.

Para medir el nivel de coincidencia entre ellos usaremos el método de similitud del coseno, el cual mide la similitud de dos vectores n-dimensionales basándose en el ángulo que forman.

La fórmula detrás del método es la siguiente, siendo a y b dos vectores de los que calcular la similitud:

$$sim(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\mid \vec{a} \mid * \mid \vec{b} \mid} \tag{3.4}$$

En el programa este método será realizado por la función, también perteneciente a la librería Scikit-Learn, *cosine_similarity()*.

3.2. Implementación práctica

Se emplea la función *TfidfVectorizer()* para hallar la matriz TF-IDF y se fija que el índice de la misma sea el username del usuario en cuestión. Las variables son todo el conjunto de palabras de todos los tuits combinados:

Username	aa	aaa	aaaaa	aaaaaa	aaaaacdvv	aaafuckoffgodno	aaagh	aaaghhh
morefrecklespls (war)	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
modus5959 (war)	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
ATree02645601 (war)	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
neuralogic (war)	0.008682	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
caitoz (war)	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...
AlasdairPlays (xbox)	0.000000	0.005986	0.0	0.0	0.0	0.0	0.0	0.0
MattsGeekCorner (xbox)	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
that_neomai_guy (xbox)	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
anchordown902 (xbox)	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
DeezyDevs (xbox)	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0

496 rows × 126810 columns

Figura 3.1: Matriz TF-IDF

Tras ello, se selecciona un usuario al azar y se comprueba la similitud usando el método de similitud del coseno, obteniendo los 10 usuarios más similares al seleccionado:

```
El usuario seleccionado es: MyaAlex92810555 (war)

Los 10 usuarios con mayor similitud al usuario MyaAlex92810555 (war) son:

La similitud con el usuario barlas_o (war) es: 0.7829087689663654.
La similitud con el usuario yoseefbasha (war) es: 0.7814178348522803.
La similitud con el usuario Jessica17390323 (war) es: 0.7811834809743862.
La similitud con el usuario 0214Tiff (war) es: 0.7801779378146478.
La similitud con el usuario RauseoJaimar (war) es: 0.7796183113745293.
La similitud con el usuario Brianna41683904 (war) es: 0.7788564544614788.
La similitud con el usuario micamartinez345 (war) es: 0.777845706709635.
La similitud con el usuario DiannaBrow98834 (war) es: 0.77679906398532.
La similitud con el usuario jonathanshunge3 (war) es: 0.7748503159332217.
La similitud con el usuario Cannon2082901 (war) es: 0.7726124177254051.
```

Figura 3.2: Recomendaciones con alta similitud

En este caso las recomendaciones son muy buenas, pues todos los usuarios recomendados pertenecen al mismo tema, y el grado de similitud es muy alta. Esto no siempre es así, como puede verse:

```
El usuario seleccionado es: artofmanliness (war)

Los 10 usuarios con mayor similitud al usuario artofmanliness (war) son:

La similitud con el usuario FonzGaming (xbox) es: 0.2788456977268521.
La similitud con el usuario GeneralJ501st (xbox) es: 0.2612399904903153.
La similitud con el usuario coach4kindness (war) es: 0.2135440855035479.
La similitud con el usuario DoktaVODKA_ (xbox) es: 0.1943632711875174.
La similitud con el usuario PureDeadGaming (xbox) es: 0.1822546544646473.
La similitud con el usuario GoodReading (war) es: 0.17322088367332583.
La similitud con el usuario SpookyTBG (xbox) es: 0.1699307113891736.
La similitud con el usuario xZer0ex (xbox) es: 0.12096571141758382.
La similitud con el usuario DiannaBrow98834 (war) es: 0.12093414050160431.
La similitud con el usuario JaneEyrePilled (war) es: 0.1201341921897047.
```

Figura 3.3: Recomendaciones con baja similitud

En este caso las recomendaciones son mucho peores, pues tan sólo 4 de los 10 usuarios pertenecen al mismo tema y ni siquiera esos usuarios se encuentran entre las primeras recomendaciones. Esto era de esperarse, pues el grado de similitud con respecto al usuario seleccionado es muy bajo.

Capítulo 4

Algoritmos de clasificación

Ya que se ha seleccionado a los usuarios mediante temas específicos, se les puede asignar un valor de variable de clase y emplear algoritmos de clasificación, con el objetivo de obtener modelos que puedan clasificar usuarios de forma exitosa. El método empleado es el de One Hot-Encoding, etiquetando con un 0 a los usuarios pertenecientes el tema "War" y con un 1 a los pertenecientes al tema "Xbox".

Esto podría servir para varios cometidos, como recomendar usuarios en base a ello, personalizar publicidad, etc.

A pesar de que los valores de salida de la variable de clase sean números, estos cuentan como variables categóricas, por lo que no serán usados algoritmos de regresión, los cuales emplean variables cuantitativas. Los algoritmos utilizados son los siguientes:

- Árboles de decisión
- Clasificador Naive-Bayes
- Support Vector Classifier
- Random Forest

Existe un problema inicial debido a la naturaleza de los datos obtenidos, pues los diferentes algoritmos no pueden procesar palabras, por lo cual se hace necesario traducir las palabras obtenidas a valores numéricos. Para ello se ha empleado la matriz TF-IDF obtenida en el capítulo 3.

4.1. Marco teórico

4.1.1. Análisis de componentes principales

El dataset usado para realizar estas tareas de clasificación cuenta con miles de atributos, lo cual conllevaría grandes esfuerzos computacionales a la hora de elaborar modelos, mucho tiempo para tratar los datos, y, en general, análisis inviables de los datos. Por ello se hacen necesarias técnicas que permitan reducir los datos manteniendo a su vez la mayor integridad posible de los datos. El objetivo es usar una herramienta para producir prácticamente el mismo resultado cuando se aplica sobre un reducido conjunto de datos en lugar de los datos originales.

El análisis de componentes principales (Principal Component Analysis, PCA) es un método de reducción de la dimensionalidad que permite simplificar la complejidad de

espacios multidimensionales intentando, a su vez, que se mantenga la mayor cantidad de información posible.

El punto de partida de esta técnica es, por ejemplo, un conjunto de datos en los que contamos con n observaciones y v variables. Usando el PCA, lograremos condensar la información ocurriendo que del número inicial de variables v, obtendremos un número de componentes Z, de tal manera que $Z < v$.

Tal y como se explica en VanderPlas (2022) y en James (2013), cada componente se calcula como una combinación lineal de las variables. Se define el conjunto de variables como $X = (X_1, X_2, ..., X_v)$, por lo que la componente Z_1 sería igual a:

$$Z_1 = Y_{11}X_1 + Y_{21}X_2 + ... + Y_{v1}X_v \tag{4.1}$$

siendo los términos $Y_{11}, ..., Y_{v1}$ coeficientes que definen las componentes, y que van asociados a cada variable, estableciendo, de esta manera, el peso que cada variable tiene dentro de la componente.

Un concepto imprescindible en el PCA es el de varianza. Según Matus (2010), la variabilidad o dispersión de los datos puede venir dada por la varianza, la cual se expresa de la siguiente manera:

$$VAR(X) = \frac{\sum_{i=1}^{n}(x_i - \bar{X})^2}{n} \tag{4.2}$$

donde n es el número de muestras y $\bar{X}$ la media.

Cada componente tiene un valor de varianza de los datos asociados, por lo que calculando la varianza acumulada se puede llegar a seleccionar un número de componentes que conserve suficiente variabilidad de los datos, despreciando el resto de componentes.

Aquella componente con mayor escala dominará sobre el resto, por lo cual se hace imprescindible normalizar los datos para que esto no ocurra.

Como punto positivo, tenemos que cada dimensión o componente principal generada por PCA será una combinación lineal de las variables originales, y serán además independientes o no correlacionadas entre sí.

4.1.2. Arboles de decisión

Los árboles de decisión pueden usarse tanto para tareas de clasificación como de regresión, y como su nombre indica tiene una estructura semejante a un árbol, empezando con un nodo raíz, el cual lleva a diferentes nodos internos que representarán decisiones; derivando en sucesivos nodos hoja que darán lugar a las predicciones sobre aquello que queremos clasificar. En la figura 4.1 se muestra un árbol de clasificación para la tarea de clasificar a un paciente en las clases Enfermo, Sano usando las variables cp, ca y thal.

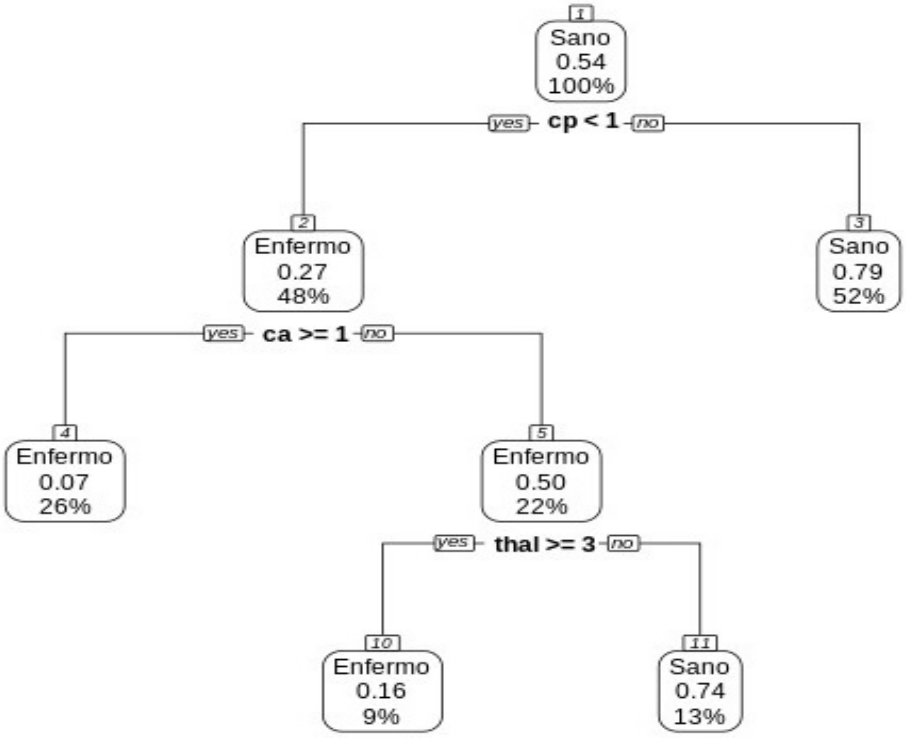

Figura 4.1: Ejemplo de árbol de decisión

En el libro Tan (2014) hay más información al respecto sobre este algoritmo, en su capítulo correspondiente.

4.1.3. Clasificador Naive-Bayes

El clasificador Naive-Bayes permite resolver problemas de clasificación mediante el teorema de Bayes: teniendo un atributo de clase C y un conjunto de variables predictoras $A_1, ..., A_n$, podemos calcular la probabilidad de que se de el atributo de clase en base a las variables predictoras.

$$P(C \mid A_1, ..., A_n) = \frac{P(A_1, ..., A_n \mid C) \cdot P(C)}{P(C \mid A_1, ..., A_n)} \tag{4.3}$$

Por otra parte, la etiqueta “Naive” (o ingenuo) del clasificador se debe a la suposición del clasificador de que, conocido el valor de la clase, los atributos son independientes entre sí, lo cual no tiene necesariamente por qué ser así.

En el libro Tan (2014) hay más información al respecto sobre este algoritmo, en su capítulo correspondiente.

4.1.4. Support Vector Machine

El Support Vector Machine es un algoritmo ideado en los años 90 que en un principio se usaba para clasificación binaria y luego se extendió su uso para problemas de clasificación con múltiples variables y problemas de regresión. El objetivo del SVM es el de poder separar dos o más clases diferentes de puntos de datos que se hallan en un espacio n-dimensional.

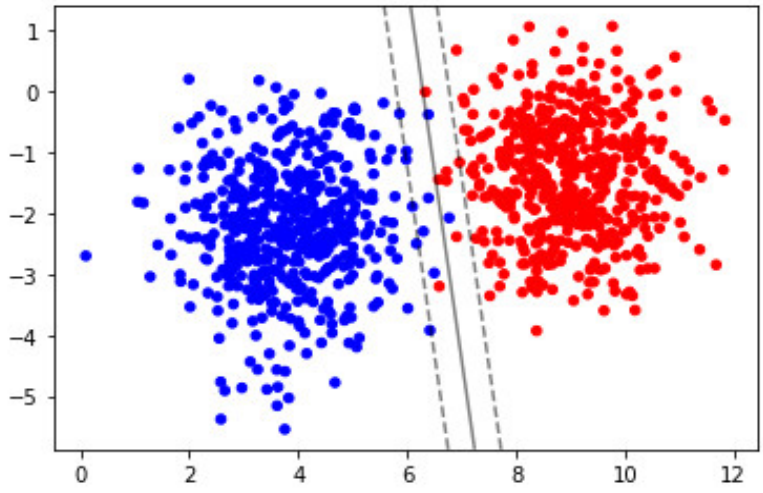

Figura 4.2: Conjunto de datos linealmente separable

Cuanta peor sea la separación entre las clases, peor será la clasificación:

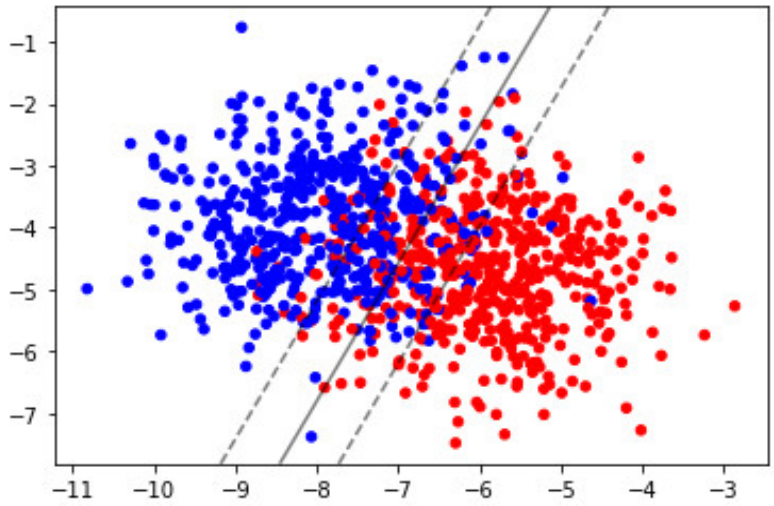

Figura 4.3: Conjunto de datos linealmente no separable

Sin embargo, existe una solución para el caso de que los datos no sean linealmente separables, y esta solución son las funciones kernel, las cuales permiten mapear el espacio de características original a uno de mayor dimensionalidad donde el conjunto de datos sí que sea separable. Esto, de forma contraria a otras operaciones como el análisis de componentes principales, donde se busca reducir la dimensionalidad.

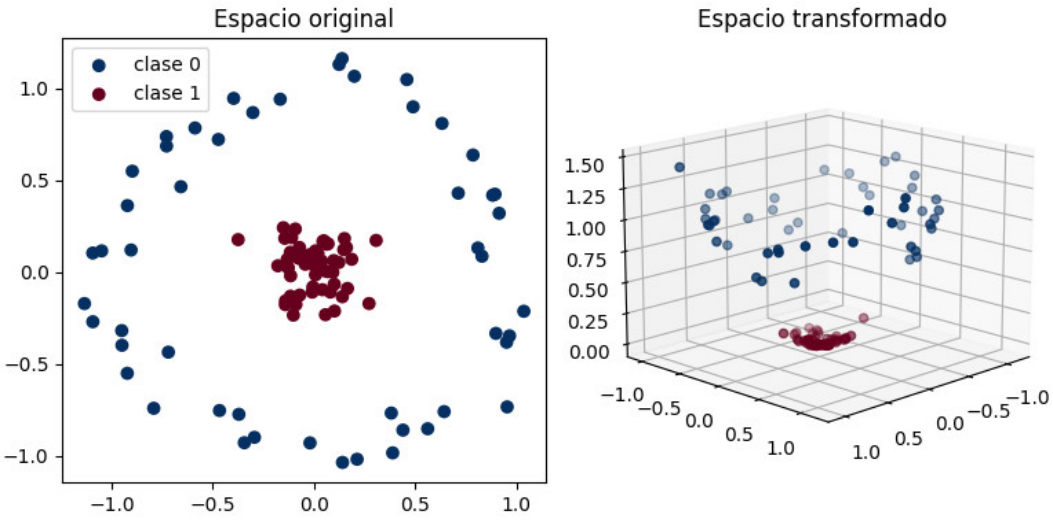

Figura 4.4: Espacio transformado mediante kernel

Como puede verse, los datos en el espacio original no serían apropiadamente separados de forma lineal, pero sí en el espacio transformado.

Existen distintas funciones kernel, como la lineal, polinómica, sigmoide y función de base radial o gaussiana.

Puede ampliarse la información sobre este algoritmo en el artículo Hsu et al. (2003).

4.1.5. Random Forest

El modelo Random Forest toma como base al árbol de decisión. Partiendo de este modelo se forma un conjunto de árboles de decisión, cada uno de ellos entrenado con una muestra ligeramente distinta de los datos. De igual modo, la predicción será igual a la media de las predicciones del conjunto de árboles.

Existe más información disponible sobre este algoritmo en el capítulo correspondiente del libro Müller and Guido (2016).

4.2. Parte práctica

4.2.1. Consideraciones previas

Los modelos han sido entrenados realizando previamente reducción de dimensionalidad mediante PCA y también sin realizarla, para poder comparar ambos resultados en términos de accuracy.

4.2.2. Preprocesado

Mezclado

Debido a cómo está construida la matriz TF-IDF obtenida en el capítulo 3 hay que aplicar un *shuffler*, ya que al unir los dataframes de ambos temas, se ponen, por decirlo de alguna manera, uno encima de otro. Si se fueran a elaborar así los modelos de clasificación, se beneficiaría más a la clase con la que se encontrara primero. Para evitarlo, se aplica un *shuffler* que mezcle las instancias de ambas clases en todo el dataframe.

Hiperparámetros

Por otra parte, los algoritmos de SVM y Random Forest hacen uso de hiperparámetros que deben ser definidos antes de crear el modelo y de los que no se puede conocer a priori su valor óptimo. Se hace uso de la herramienta GridSearchCV, la cual entrena al modelo en cuestión con diferentes valores de dichos hiperparámetros para hallar así la combinación que brinde el mejor modelo posible y ahorrar tiempo.

Balanceo

Un problema que surgió en un primer momento fue el acabar, tras todos los pasos de recolección de datos, con un número dispar de usuarios de ambos temas. Por ejemplo, 280 usuarios de un tema y 140 del otro. Esto supone un problema a la hora de aplicar algoritmos de clasificación, ya que existiría un desbalanceo entre las clases, es decir, que una de las clases fuese minoritaria con respecto a la otra. El algoritmo con mucha más frecuencia categorizaría las instancias como pertenecientes a la clase mayoritaria, ya que así el índice de acierto sería mucho mayor. Esto es un error, ya que no se clasificaría correctamente a la clase minoritaria.

Por tanto, se repitió el proceso de adquisición de datos cuantas veces fuera necesario para conseguir una cantidad pareja de usuarios de ambos temas. Finalmente la cantidad de usuarios ha sido de 237 para el tema War y 259 para el tema Xbox, resultando en un balanceo casi perfecto.

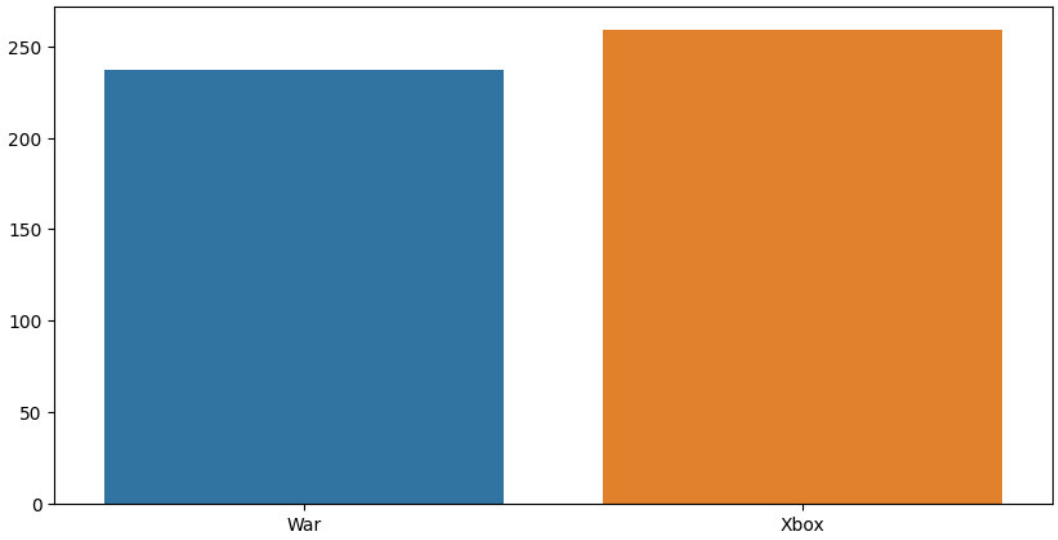

Figura 4.5: Balanceo de las clases

División del conjunto de entrenamiento

A la hora de crear los modelos se ha dividido el dataset en un conjunto de entrenamiento y otro de validación, siendo cada uno del 70 % y 30 % del conjunto de datos total, respectivamente.

Normalizado

Debido a que los valores TF-IDF están contenidos dentro del intervalo [0,1], no es necesario normalizar los datos.

Análisis de componentes principales

En el caso del PCA, primero se lleva a cabo sin indicarle un número de componentes, para calcularlos todos, lo que permite construir el dataframe en el que figuran los coeficientes que asignan el peso que tiene cada una de las variables por cada una de las componentes, tal y como se mostró en la ecuación 4.1. Esto puede verse en la figura 4.6, en la que, una vez hecho el PCA se obtiene un dataframe en el que las columnas son las variables, en este caso el conjunto total de palabras de todos los tuits, y en las filas no figuran los usuarios, sino los distintos componentes. Los datos de este dataframe son los coeficientes:

	aa	aaa	aaaaa	aaaaaa	aaaaacdvv
PC1	0.002234	0.009462	-0.000005	-0.000069	-0.000070
PC2	0.001110	0.015360	0.000251	0.000005	0.000004
PC3	0.000336	0.010964	0.000197	-0.000016	-0.000018
PC4	-0.000555	0.005213	-0.000133	-0.000198	-0.000198
PC5	-0.000417	0.001468	-0.000051	-0.000128	-0.000130
...	...	...	...	...	...
PC492	0.001461	0.004403	-0.000231	-0.000053	0.000073
PC493	-0.000119	-0.000392	-0.000007	-0.000038	-0.000059
PC494	-0.001055	-0.005493	-0.000274	0.000022	0.000008
PC495	0.001492	0.010286	0.000140	0.000078	0.000035
PC496	-0.040678	0.254237	-0.032115	0.009711	0.039111

496 rows × 126810 columns

Figura 4.6: Componentes

Luego, se emplean los atributos $explained\_variance\_ratio\_$, $n\_components\_$ y $prop\_varianza\_acum$ del modelo para hacer una gráfica que muestre la suma de los valores de varianza de cada componente para hallar el porcentaje de varianza acumulada, lo que facilitará el seleccionar más adecuadamente el número óptimo de componentes:

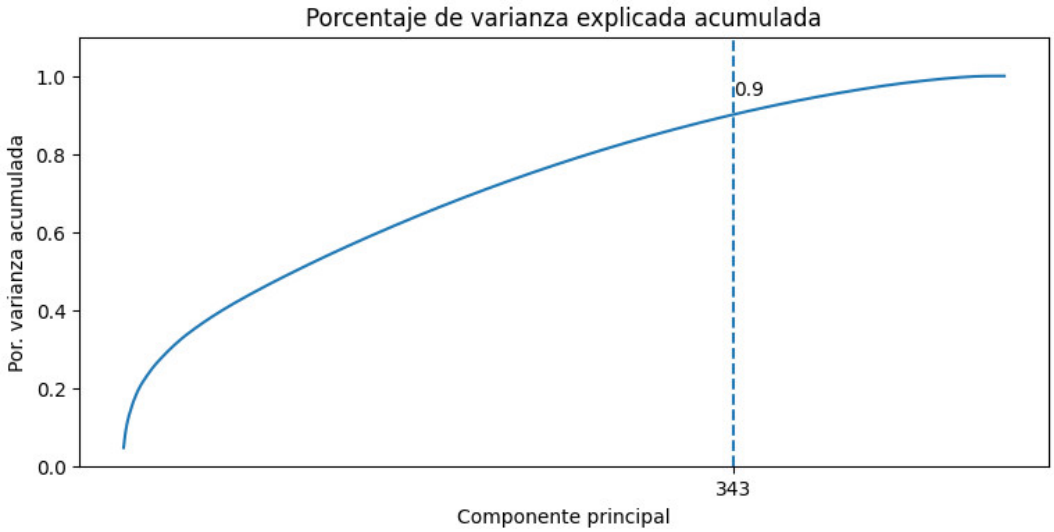

Figura 4.7: Gráfica Varianza

Como puede verse, con 343 componentes se conserva el 90 % de la varianza, por lo que se volverá a realizar el PCA, este vez especificándole que el número de componentes será 343. Se obtendrá un dataframe parecido al de la figura 4.6, pero en vez de 496 filas contará con 343, cada una de las cuales será uno de los componentes que se conservará para mantener ese 90 % de varianza.

Tras ello, se transformará la matriz de componentes mediante el método $transform()$ de la clase $sklearn.decomposition.PCA$ para obtener el dataframe definitivo que se usará para entrenar los modelos de clasificación:

	0	1	2	3	4
0	0.298564	0.222225	0.112709	-0.002636	-0.038683
1	0.261284	0.179481	0.107264	0.046343	-0.015593
2	0.340508	-0.130795	-0.168220	-0.020925	0.011399
3	-0.078737	0.209387	0.026382	-0.054446	0.017396
4	0.261689	-0.182593	0.042201	0.081204	0.065586
...	...	...	...	...	...
491	0.306497	0.390977	0.185223	0.049428	-0.019170
492	0.223342	-0.133969	-0.125965	-0.069508	-0.002842
493	-0.161399	0.110235	-0.012475	-0.028475	-0.045129
494	0.244380	-0.081813	-0.079590	0.000990	-0.008176
495	0.267549	-0.108615	-0.004676	0.002470	0.009827

496 rows × 343 columns

Figura 4.8: Dataframe final

Puede observarse que de las 126810 columnas iniciales ha quedado un dataframe con

tan sólo 343, habiéndose reducido drásticamente la dimensionalidad.

A continuación se muestran los resultados obtenidos al aplicar las técnicas de clasificación descritas anteriormente sobre el dataframe mostrado en la figura 4.8.

4.2.3. Arbol de decisión

Dimensionalidad	Valores
Accuracy sin PCA	86.58 %
Accuracy con PCA	81.21 %

4.2.4. Clasificador Naive-Bayes

Dimensionalidad	Valores
Accuracy sin PCA	84.56 %
Accuracy con PCA	70.47 %

4.2.5. Support Vector Machine

La serie de valores entre los que GridSearchCV buscará los hiperparámetros óptimos para el clasificador será la siguiente:

Hiperparámetro	Valores
kernel	(’linear’, ’rbf’, ’poly’, ’sigmoid’)
C	(0.1, 1, 10, 100, 1000)
gamma	(’auto’, ’scale’)

Los hiperparámetros indican lo siguiente:

- Kernel: función que permitirá cambiar la dimensionalidad del conjunto de características con el objetivo de establecer hiperplanos más eficientes a la hora de clasificar.
- C: compensa la clasificación errónea de los ejemplos de entrenamiento frente a la simplicidad de la superficie de decisión. Una C baja suaviza la superficie de decisión, mientras que una C alta apunta a clasificar correctamente todos los ejemplos de entrenamiento.
- Gamma: define cuánta influencia tiene un solo ejemplo de entrenamiento. Cuanto mayor sea gamma, más cerca deben estar otros ejemplos para verse afectados.

Los resultados obtenidos son los siguientes:

Dimensionalidad	Valores
Accuracy sin PCA	87.25 %
Accuracy con PCA	87.25 %

4.2.6. Random Forest

La serie de valores entre los que GridSearchCV buscará los hiperparámetros óptimos para el clasificador será la siguiente:

Hiperparámetro	Valores
n_estimators	150
max_features	(5, 7, 9)
max_depth	(None, 3, 10, 20)
criterion	(’auto’, ’scale’)

Los hiperparámetros indican lo siguiente:

- n_estimators: El número de árboles en el bosque.
- max_features: Cantidad de características a considerar al buscar la mejor división.
- max_depth: La profundidad máxima del árbol. Si es Ninguno, los nodos se expanden hasta que todas las hojas sean puras o hasta que todas las hojas contengan menos de un valor determinado de muestras.
- criterion: La función para medir la calidad de una división.

Los resultados obtenidos son los siguientes:

Dimensionalidad	Valores
Accuracy sin PCA	85.23 %
Accuracy con PCA	80.54 %

Puede verse el desempeño de los distintos algoritmos en la siguiente tabla comparativa:

Dimensionalidad	Arbol	Naive-Bayes	SVC	Random forest
Accuracy sin PCA	86.58 %	84.56 %	87.25 %	85.23 %
Accuracy con PCA	81.21 %	70.47 %	87.25 %	80.54 %

En general, la eficacia al clasificar sin reducir la dimensionalidad ha sido buena, siendo el menor valor de accuracy un 84.56 %. Por otro lado, habiendo realizado previamente el PCA se obtiene que excepto el *Support Vector Classifier*, la eficacia del resto del algoritmos ha descendido, siendo un ligero descenso en el caso del árbol de decisión y el *random forest* y un dramático descenso en el caso del clasificador *Naive-Bayes*

Capítulo 5

Sistema recomendador usando Word2Vec

El *deep learning* es un conjunto de algoritmos del *machine learning*, una rama de la inteligencia artificial. Estos algoritmos están especialmente enfocados en el uso de redes neuronales, con el objetivo de extraer patrones o características de los datos que puedan ser de utilidad.

Para este sistema recomendador se usará *Word2Vec*, una popular técnica para el procesamiento de lenguaje natural que emplea un modelo de red neuronal. Esta técnica fue creada y publicada por un equipo dirigido por Tomas Mikolov en Google, en 2013. Pero antes de explicarla, se hará un breve repaso a los fundamentos del *deep learning*, pasando por los conceptos clave hasta llegar a ella.

5.1. Marco teórico

5.1.1. Neurona

Las redes neuronales son estructuras cuya unidad fundamental es la neurona. Estas se organizan en capas, y cada una de las neuronas es capaz de comunicarse con las neuronas de las capas anterior y posterior.

Cada una de las neuronas tiene dos variables asociada: el valor de entrada x_i y el peso asociado a dicha neurona w_i. Como se puede ver en la siguiente figura, no es tan sólo un valor de entrada y un peso, sino varios de ellos:

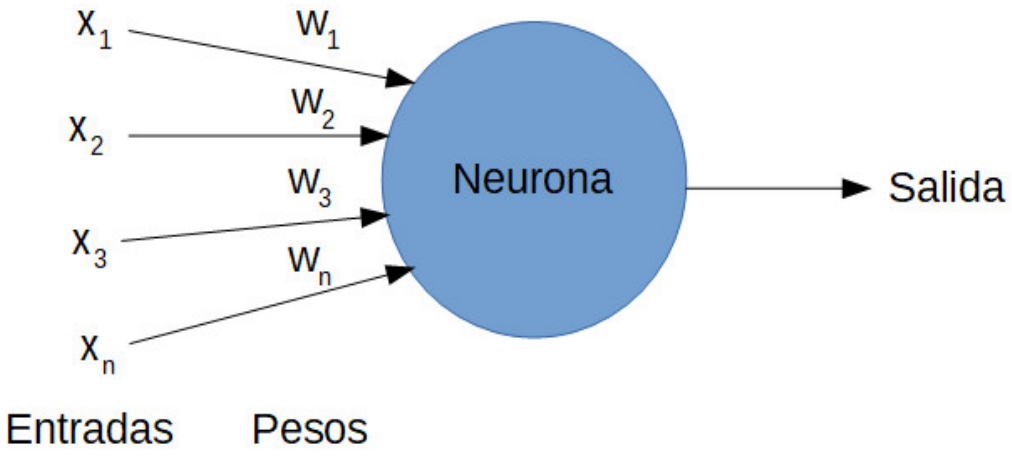

Figura 5.1: Neurona simple

Por lo tanto, se tiene un vector X=[x_1, x_2, x_3,...,x_n], que representa los valores de entrada y un vector W=[w_1, w_2, w_3,...,w_n] que representa a los pesos. Partiendo de aquí, puede verse que la entrada de una neurona será igual al producto vectorial de X y W, más el valor de bias b:

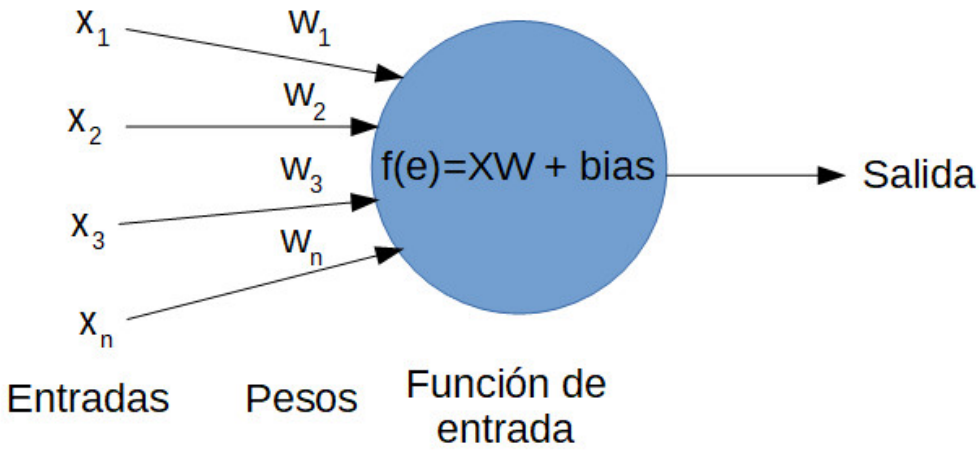

Figura 5.2: Función de entrada

Sin embargo, si esto fuese lo único operando en las neuronas, estas no serían capaces de aprender relaciones no lineales, puesto que, como puede verse, la función de entrada antes definida es puramente lineal. El factor necesario para superar esa barrera es la función de activación g, la cual transforma a la entrada y devuelve el valor de salida de la neurona a, que es lo que se conoce como valor de activación de la neurona:

$$a = g(entrada) = g(XW + bias) \tag{5.1}$$

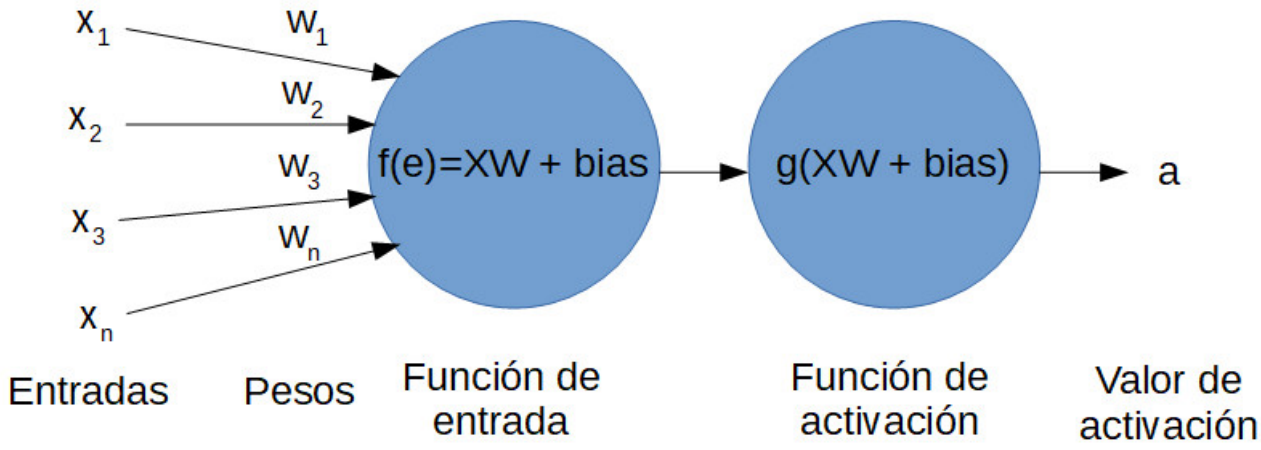

Figura 5.3: Neurona completa

5.1.2. Función de activación

Estas funciones, las cuales tienen por lo general carácter no lineal, son las que transforman la entrada a la neurona en la salida de la misma, el llamado valor de activación.

Es debido a estas funciones y a la existencia de una o más capas de neuronas que la red es capaz de aprender relaciones no lineales más o menos complejas. Las funciones de activación más usuales son las siguientes:

ReLU (Rectified Linear Unit)

Consiste en una función lineal que devuelve 0 si el valor de entrada es menor o igual a 0, mientras que si la entrada es positiva, la salida será igual a la entrada, conservando así únicamente los valores positivos:

$$f(x) = max(0, x) = \begin{cases} x_i & \text{si } x_i > 0 \\ 0 & \text{si } x_i < 0 \end{cases} \tag{5.2}$$

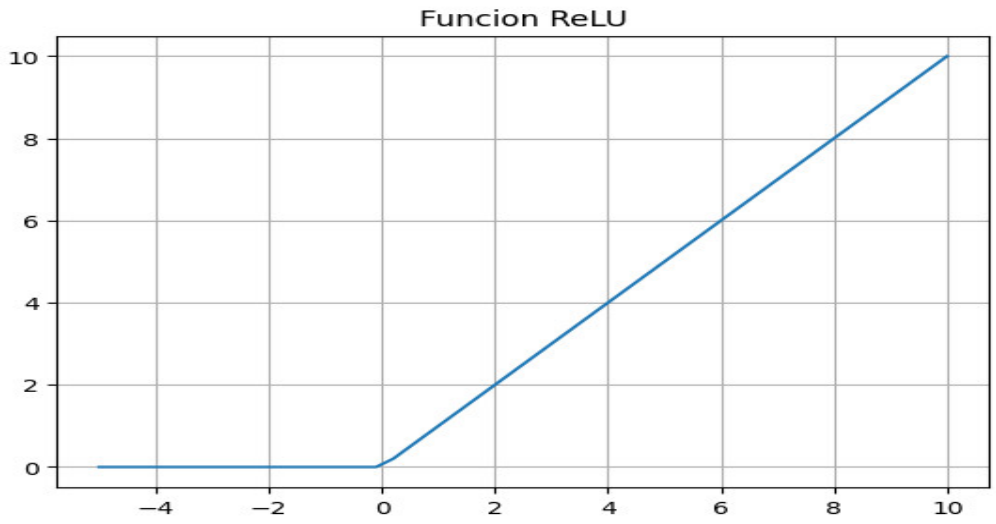

Figura 5.4: Rectified Linear Unit

Función sigmoide

La función sigmoide se encarga de transformar la variable de entrada, la cual puede estar en el rango (-∞, +∞) al rango [0,1]. La fórmula que define a la función es la siguiente, puede verse la representación de la misma:

$$sigmoide(x) = \frac{1}{1 + e^{-x}} \tag{5.3}$$

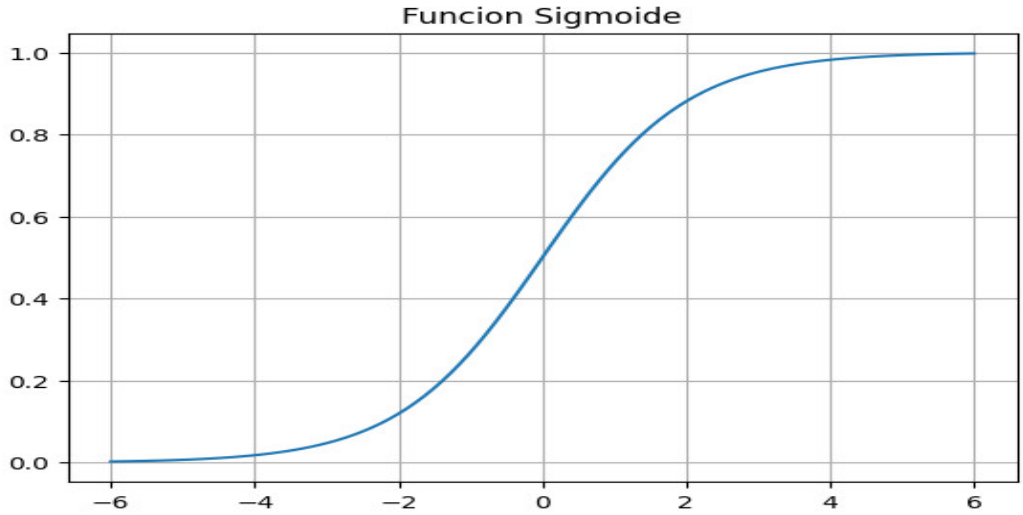

Figura 5.5: Función sigmoide

Función tangente hiperbólica

La función tangente hiperbólica se encarga de transformar la variable de entrada, la cual puede estar en el rango (-∞, +∞) al rango [-1,1]. La fórmula que define a la función es la siguiente, puede verse la representación de la misma:

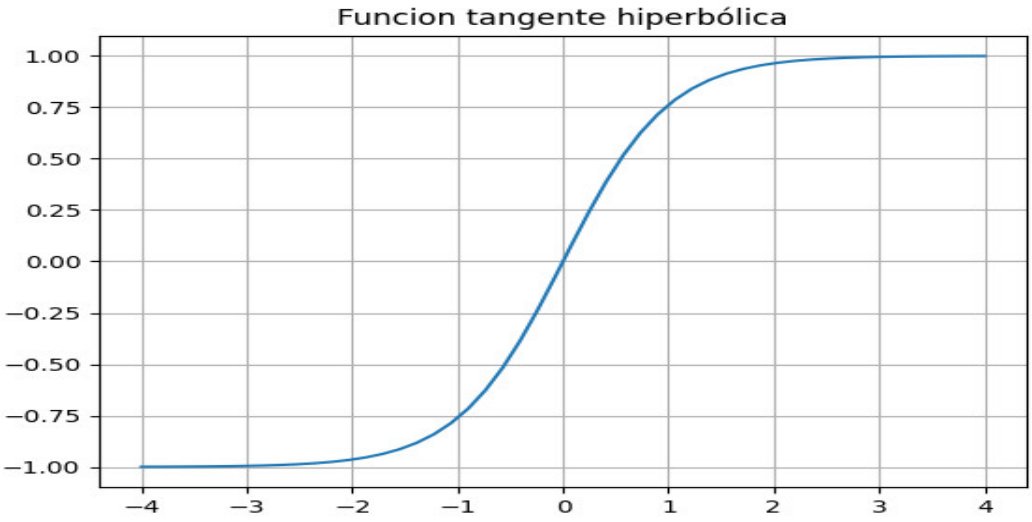

Figura 5.6: Función tangente hiperbólica

5.1.3. Bias

Una vez comprendido qué son las funciones de activación, se puede explicar el último término que actúa en la neurona, el cual ya ha sido mencionado previamente. Este se llama sesgo o bias. Mientras que los datos de entrada no pueden ser modificados y los pesos hacen que varíe la inclinación de la curva, el valor de bias permite que la función de activación se desplace hacia la izquierda o hacia la derecha, cambiando la curva para que se ajuste mejor a los datos.

Para comprender mejor este concepto, se representa gráficamente lo que es el bias tomando para ello la función sigmoidea:

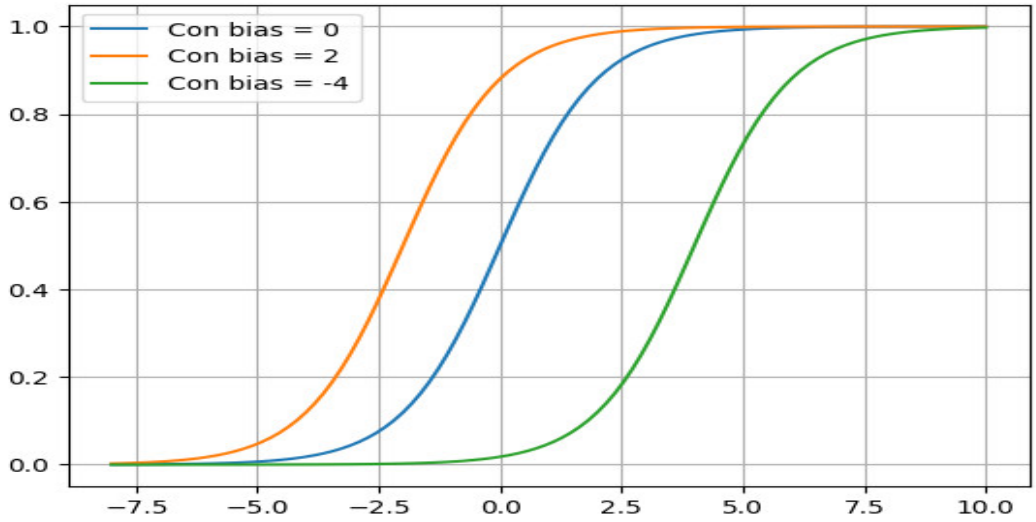

Figura 5.7: Bias

Como puede verse, en la curva azul el bias es 0, lo que da lugar a la función sigmoidea normal. En la curva naranja el valor de bias aumenta, por lo que el valor de x requerido para que la función pase a 1 será menor. Al contrario ocurre con la gráfica verde, en la cual el bias es negativo y que requerirá valores más elevados de x para pasar a 1. Por lo tanto, se puede asociar el bias a la capacidad de hacer que una neurona se active:

- Con bias alto, la neurona se activará (pasará a 1) más fácilmente, puesto que la curva está desplazada hacia la izquierda.
- Con bias bajo, la neurona se activará (pasará a 1) más difícilmente, puesto que la curva está desplazada hacia la derecha.
- Con bias cercano o igual a 0, el valor de la salida dependerá sólo de las entradas y pesos.

5.1.4. Red neuronal

Las redes neuronales se organizan siguiendo una estructura de capas, una a continuación de la otra, tal y como puede verse:

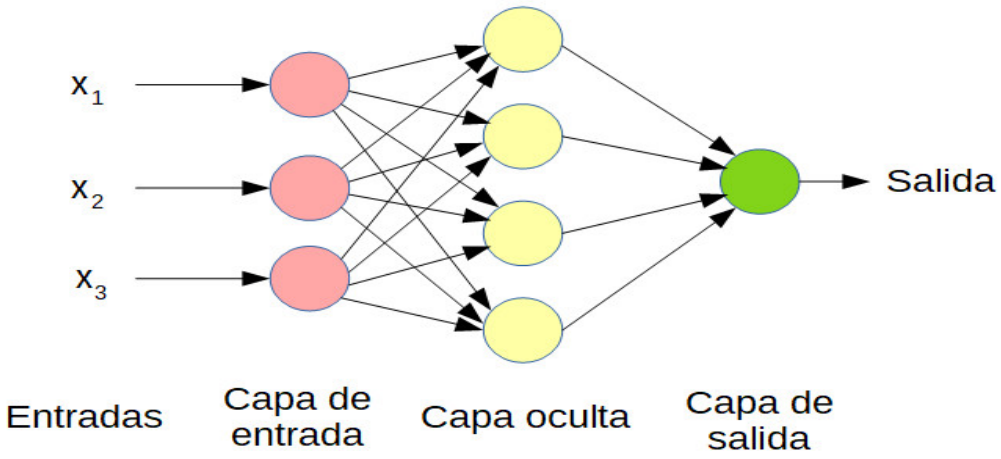

Figura 5.8: Red neuronal simple

En esta estructura se pueden apreciar tres capas diferentes. La capa de entrada consiste en neuronas que toman los valores de entrada y los transmiten sin alteración alguna a la siguiente capa. Para ello, la función de activación de dichas neuronas habrá de ser la unidad, para que salga lo mismo que entre.

En la capa oculta se reciben los valores de la capa de entrada y se combinan con los diferentes pesos de cada neurona. A la hora de entrenar un modelo lo que se busca es hallar los vectores de pesos, así como el bias.

Por último, la capa de salida transforma los valores de salida de la capa oculta para dar lugar a la salida de la red.

Cuantas más neuronas puedan añadirse a las capas y más capas se añadan, mayor será la complejidad de los patrones que la red puede aprender.

5.1.5. Modelos neuronales

A pesar de que existe variedad de modelos para la representación contínua de palabras, en el ártículo Mikolov et al. (2013), que dio lugar a *Word2Vec*, se basan en los modelos que emplean redes neuronales, más concretamente en el *Feedforward Neural Net Language Model* y el *Recurrent Neural Net Language Model*, que son los modelos más parecidos a los desarrollados en el artículo.

Feedforward Neural Net Language Model

El modelo probabilístico *Feedforward Neural Net Language* consiste en una capa de entrada, una de proyección (P), una o más capas ocultas (H) y la capa de salida (V).

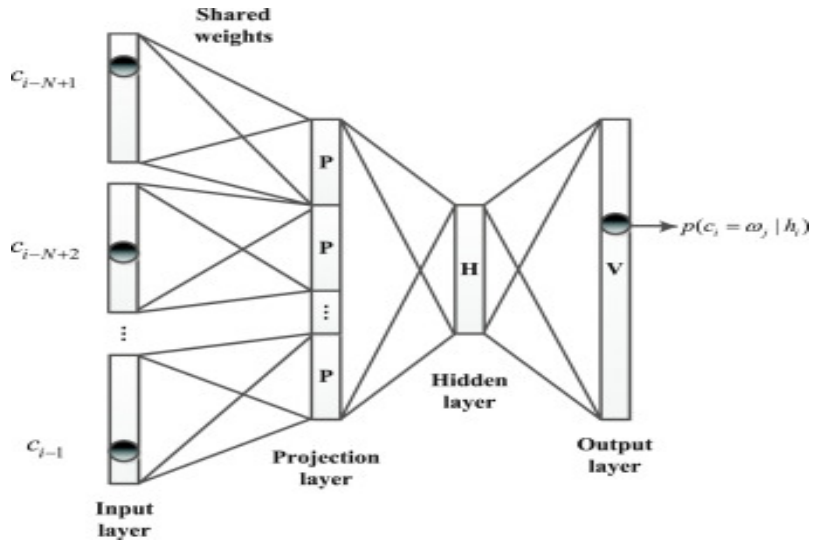

Figura 5.9: Feedforward NNLM. Fuente: Wu et al. (2017)

En la capa de entrada, se codifican N palabras anteriores utilizando la codificación *1-of-V*, donde V es el tamaño del vocabulario. Seguidamente, la capa de entrada se proyecta a un capa de proyección P que tiene dimensionalidad $N \times D$. Debido a que sólo N entradas están activas en un momento dado, la composición de la capa de proyección se produce a un bajo coste computacional.

Recurrent Neural Net Language Model

RNN son las siglas de Recurrent Neural Network, o Red Neural Recurrente. Mientras que en las redes neuronales clásicas un vector de entrada produce un vector de salida y las secuencias de datos son tratadas de una sola vez, en el caso de las redes recurrentes tendremos que el tratamiento de cada uno de los elementos del vector entrada constituirá un paso temporal. Por lo tanto, cada elemento del vector salida exceptuando el primero, estará relacionado con el estado oculto del paso temporal anterior. Esto implica cierta memoria con respecto al instante previo, y es lo que le da a la red neuronal el nombre de recurrente. De esta forma, los datos se tratan secuencialmente.

Este modelo permite superar ciertas limitaciones del feedforward NNLM, como la necesidad de especificar la longitud del contexto (el orden del modelo N), y porque teóricamente los RNN pueden representar de forma mas eficiente patrones complejos que usando redes neuronales clásicas.

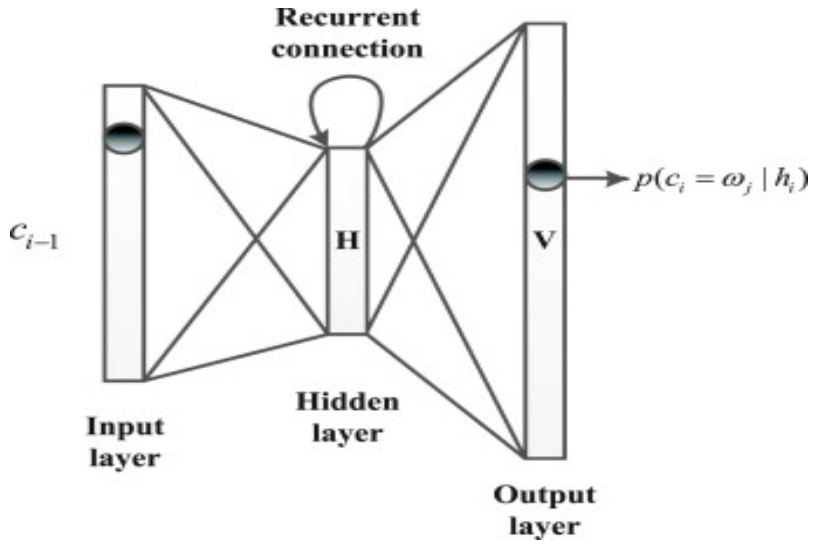

Figura 5.10: Recurrent NNLM. Fuente: Wu et al. (2017)

5.1.6. Word2Vec

Como se explica en Zhang et al. (2021), el lenguaje natural es un sistema usado para expresar significados, cuya unidad básica son las palabras. Para mostrarlas de forma numérica, de tal manera que puedan ser procesadas computacionalmente, se crean vectores de palabras, los cuales se usan para representar palabras. La técnica para ello se conoce como *word embedding*.

Una primera aproximación a la creación de dichos vectores sería la de los vectores *one-hot*. La representación de diferentes palabras usando este método sería la siguiente:

$$\begin{aligned} gato &= (0, 0, 1) \\ perro &= (0, 1, 0) \\ zepelin &= (1, 0, 0) \end{aligned}$$

Como puede verse, la representación *one-hot* crea vectores unitarios que cuya similitud es igual a cero en todos los casos, sin importar la palabra que sea. Esto hace que sea necesario recurrir a otros métodos que puedan expresar más apropiadamente las palabras y su significado.

El modelo Word2Vec resuelve este problema ya que es capaz de hallar relaciones sintácticas y semánticas de las palabras, como se explica en el artículo Mikolov et al. (2013). Esto permite representaciones vectoriales mucho más fidedignas de las palabras.

En dicho artículo se proponen dos modelos que permiten poder representar vectores continuos de palabras para datasets muy grandes, los cuales son el modelo de bolsa de palabras continua (*Continous Bag Of Words*, CBOW) y el modelo *continous skip-gram*, ambos con enfoques diferentes.

Skip-Gram

De acuerdo a Zhang et al. (2021), este modelo considera que una palabra puede usarse para generar las palabras que pueden ir asociadas a ella en una secuencia de texto, es decir, para deducir el contexto de una palabra. Si se tiene la palabra *w(t)*, se obtendrán las

palabras $(w(t-n), ..., w(t-1), w(t+1, ..., w(t+n))$, donde *n* es la ventana de contexto, es decir, el número de palabras anteriores y posteriores en una frase asociadas a la palabra en cuestión.

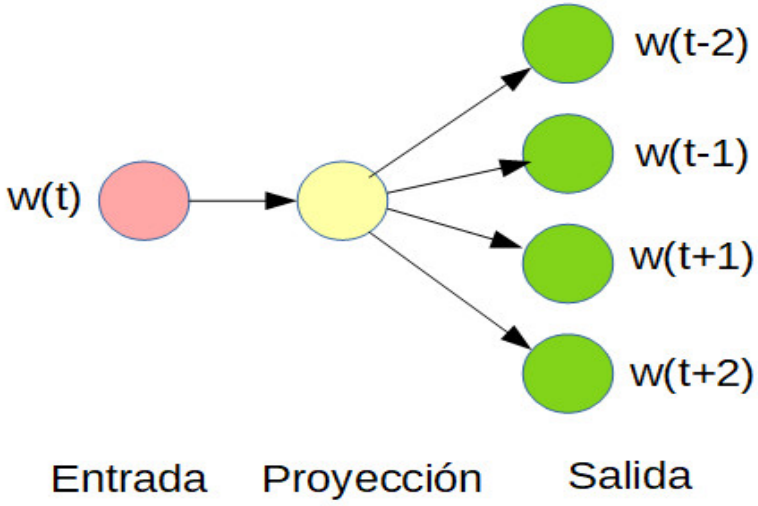

Figura 5.11: Modelo Skip-Gram con n = 2

Como ejemplo se puede tomar la secuencia de texto “el hombre ama su hijo”. Se elige “ama” como palabra central y se establece el tamaño de la ventana de contexto en 2, por lo que sólo se tomarán las dos palabras más cercanas. Al establecer como palabra central “ama”, el modelo skip-gram considera la probabilidad condicional para generar las palabras de contexto: “el”, “hombre”, “su” e “hijo”, que no están a más de 2 palabras de distancia de la palabra central:

$$P(el, hombre, su, hijo \mid ama) \tag{5.4}$$

Si se supone que las palabras son condicionalmente independientes, la probabilidad condicional puede expresarse de la siguiente forma:

$$P(el \mid ama) \cdot P(hombre \mid ama) \cdot P(su \mid ama) \cdot P(hijo \mid ama) \tag{5.5}$$

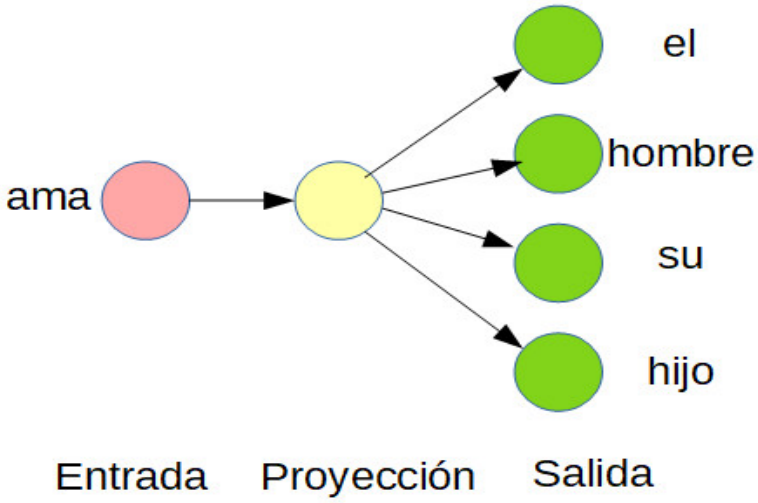

Figura 5.12: Ejemplo Skip-Gram

Continous Bag Of Words, CBOW

Esta arquitectura es similar al modelo *Skip-gram*, siendo la principal diferencia que, en lugar de intentar deducir el contexto de una palabra, intenta deducir la palabra a partir de un contexto dado, tal y como se muestra en la siguiente figura:

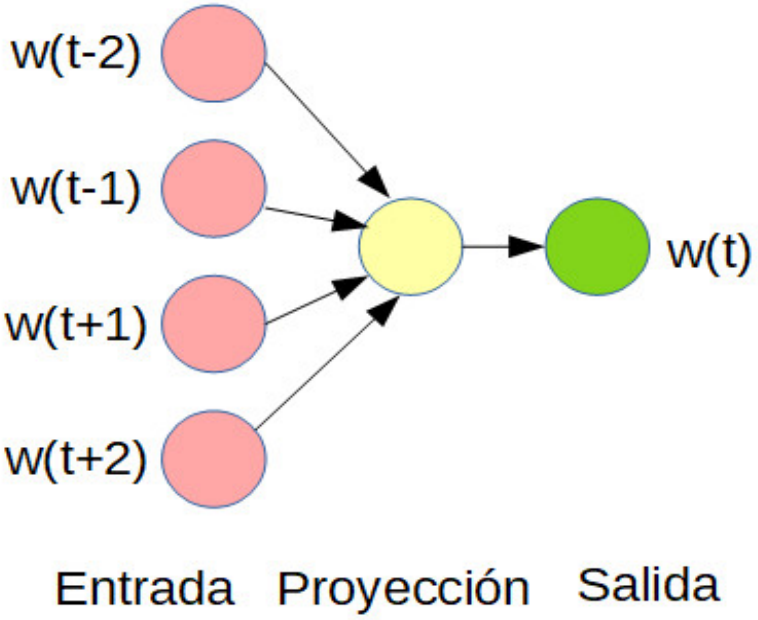

Figura 5.13: Modelo CBOW con n = 2

Por ejemplo, en la misma cadena de texto usada anteriormente, “el hombre ama su hijo”, con “ama” como palabra central y n = 2, el modelo considera la probabilidad condicional de generar la palabra central “ama” con base en las palabras de contexto “el”, “hombre”, “su” e “hijo”.

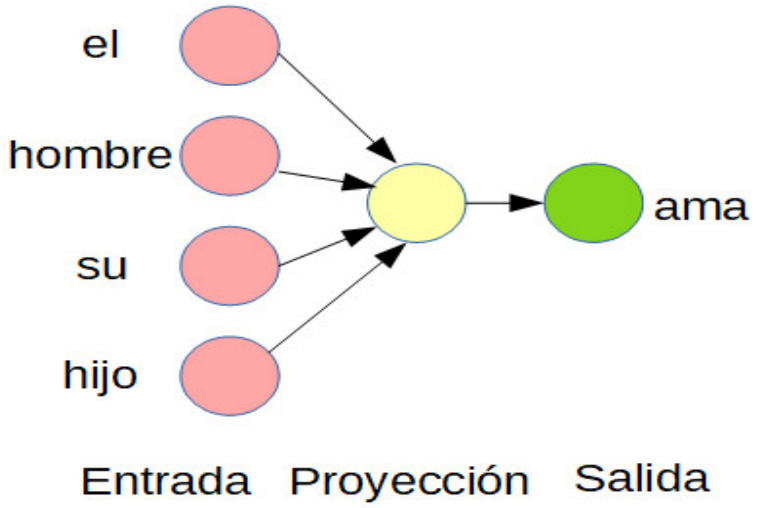

Figura 5.14: Ejemplo CBOW

5.1.7. Embedding

Una vez entrenados los modelos, se crearán vectores que contengan un valor numérico por cada usuario, sobre el que calcularemos la similitud de los mismos, de acuerdo a Tabii et al. (2018).

Average Word2Vec Vectors

Calcula el valor promedio de todos los *embeddings* de palabras de un usuario calculado mediante word2vec. Este vector promedio será, a su vez, un *embedding* de los usuarios de la siguiente manera:

$$V(t) = \frac{1}{N} \sum_{i=0}^{N} V(W_i) \tag{5.6}$$

Donde N es el número de palabras del usuario, y $V(W_i)$ el índice de la palabra (W_i) dentro del vocabulario del modelo.

5.2. Parte práctica

De cara a implementar el sistema de recomendación, se entrenarán dos modelos de *Word2Vec*, uno usando la arquitectura de la bolsa continua de palabras (CBOW), la cual es la que se usa por defecto al crear un modelo y otro empleando la arquitectura *Skip-Gram*, para poder así comparar el desempeño de ambos modelos. Asimismo, a la hora de comparar la similitud entre los usuarios, se emplearán los *Average Word2Vec Vectors*, gracias al método de similitud del coseno.

Gracias a la herramienta online *embedding projector* del sitio web *tensorflow.org*, puede visualizarse una nube de puntos, de los cuales cada uno es una palabra del total de palabras entre todos los usuarios. Dicha herramienta hace un análisis de componentes principales, estableciendo los tres primeros con mayor varianza como ejes del espacio donde se proyecta la nube de puntos.

La proyección visualizada es la correspondiente a la arquitectura CBOW. Como puede verse, se ha seleccionado la palabra *war*, apareciendo resaltadas las 100 palabras más afines a la seleccionada, como *atrocity, nuclear, genocide, troops*, etc.

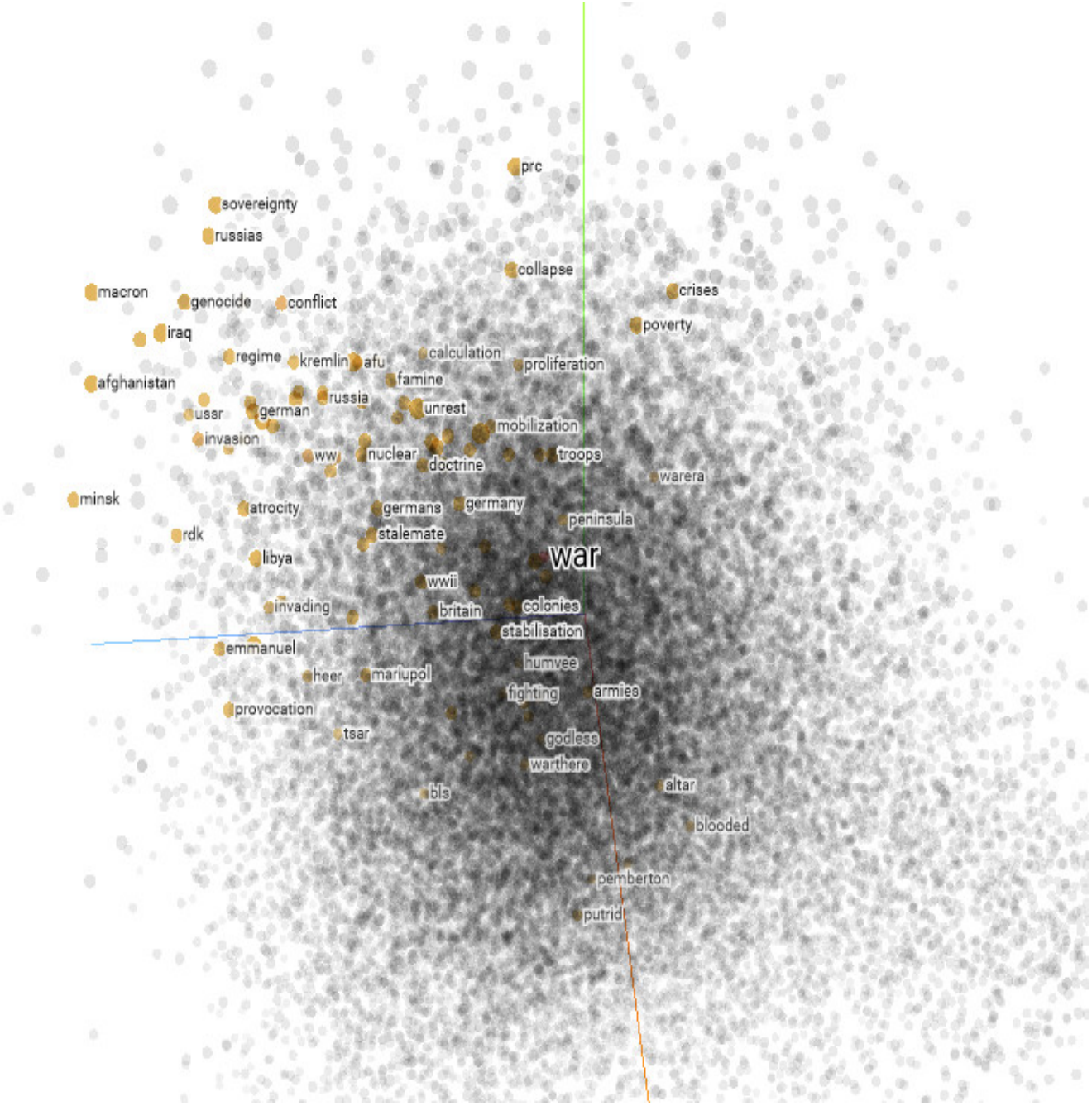

Figura 5.15: Proyección

De la misma manera, puede observarse que las palabras resaltadas ocupan un lugar característico de la nube de puntos pero no otras regiones, que estarían pobladas por otras palabras menos afines a la seleccionada.

5.2.1. Skip-Gram

Tras entrenar *Word2Vec* usando la arquitectura *Skip-Gram* y solicitar las recomendaciones, se obtiene lo mostrado en la figura 5.16.

```
El usuario seleccionado es: trsrpc (xbox)

1          iamsarahlb (xbox)
2             adeandus (war)
3     RealRonaldTrum4 (war)
4       Cannon2082901 (war)
5     JoeShow68347212 (war)
6           JoeyElrod (xbox)
7     Born2beSlicker (xbox)
8      BoulderPreston (war)
9        MarloFranson (war)
10        nyhaze4201 (xbox)
11           ImZalius (war)
12        ShaneSays_ (xbox)
13        jhpsorensen (war)
14      Casper_Raine (xbox)
15           ssjkobe8 (xbox)
16    joshuachuminski (war)
17       GeeboSamuel (xbox)
18       SnoworchidP (xbox)
19             __maiab (war)
20     anchordown902 (xbox)
```

Figura 5.16: Recomendación Skip-Gram

5.2.2. Continous Bag Of Words, CBOW

Con el fin de comprobar si los resultados son diferentes, se vuelve a entrenar el modelo *Word2Vec*, esta vez especificándole que la arquitectura es la de bolsa continua de palabras (CBOW). Los resultados son mostrados en la figura 5.17.

```
El usuario seleccionado es: trsrpc (xbox)

1          iamsarahlb (xbox)
2             adeandus (war)
3     RealRonaldTrum4 (war)
4       Cannon2082901 (war)
5     JoeShow68347212 (war)
6           JoeyElrod (xbox)
7     Born2beSlicker (xbox)
8      BoulderPreston (war)
9        MarloFranson (war)
10        nyhaze4201 (xbox)
11           ImZalius (war)
12        ShaneSays_ (xbox)
13        jhpsorensen (war)
14      Casper_Raine (xbox)
15           ssjkobe8 (xbox)
16    joshuachuminski (war)
17       GeeboSamuel (xbox)
18       SnoworchidP (xbox)
19             __maiab (war)
20     anchordown902 (xbox)
```

Figura 5.17: Recomendación CBOW

Puede verse que no es un sistema recomendador demasiado exitoso, ya que un 50 % de los usuarios es de cada uno de los temas.

Capítulo 6

Análisis de sentimientos

En este capítulo serán realizadas diversas tareas como clasificación, análisis de tendencias de los usuarios, etc, en base a los sentimientos que se pueda inferir de los tuits del usuario en cuestión.

6.1. Sentiment Lexicon

Para hallar los sentimientos se empleará un concepto conocido como *sentiment lexicon*. De acuerdo a Asghar et al. (2019), un lexicon es un repositorio de palabras en el cual determinadas palabras reciben una calificación según la connotacion que tenga. Por ejemplo, la palabra “desgracia” será calificada como negativa, mientras que “felicidad” lo será de forma positiva. Dicha calificación puede tener varios formatos, asignando por ejemplo la palabra “positivo” o “negativo”, o bien asignando un valor numérico a la palabra.

Existen varios *sentiment lexicon* de propósito general que pueden servir a los propósitos de este TFM, habiéndose seleccionado entre ellos el llamado *AFINN*, llamado así por su creador: Nielsen (2011). Este en concreto está conformado por palabras en inglés y otorga un valor numérico que va desde -5 para las palabras más negativas hasta +5, para las positivas. Esto resulta más útil que calificarlas como positivas o negativas, debido a que permite más flexibilidad a la hora de hacer diferentes representaciones gráficas.

A la hora de calcular el sentimiento total de un tuit, se comprobará cuáles son las palabras que tienen puntuación y se hará una suma para determinar el sentimiento. Si ninguna de las palabras presentes en el tuit está a su vez presente en el lexicon, se le asignará un 0, y se considerará como tuit neutro.

6.2. Análisis exploratorio de los datos

Debido a que en el dataset inicial figuran 496 usuarios, que es una cantidad muy grande, para los primeros pasos de este análisis se usarán tan sólo los tuits de dos usuarios: el usuario *neuralogic*, perteneciente al tema “war” y que cuenta con 763 tuits descargados y el usuario *MattsGeekCorner*, del otro tema, con 786 tuits.

Como primer acercamiento se puede visualizar el porcentaje de tuits de las distintas emociones por usuario:

	neuralogic (war)	MattsGeekCorner (xbox)
Tuits positivos	28.18 %	38.55 %
Tuits neutros	31.45 %	23.66 %
Tuits negativos	40.37 %	37.79 %

Como puede verse, el usuario alineado con el tema de la guerra tiene muchos un 10 % menos de tuits positivos que el otro usuario, que tiene un equilibrio entre positivos y negativos.

Seguidamente, se visualizará el número total de tuits publicados por día para cada usuario. Debido a que los datos de publicación de un tuit que proporciona *Twitter* son bastante precisos e incluyen la hora, minuto y segundo de la publicación, se reducirá la dimensionalidad de la fecha a tan sólo el año, mes y día de la publicación. Por esto, varios tuits corresponderán al mismo día, sin hora que los distinga en el eje temporal, por lo que se hará una suma para determinar todos los tuits publicados en un día. De igual forma ocurrirá con los sentimientos, sumando el sentimiento de cada tuit publicado en un día, para obtener el valor sentimental de ese determinado día.

En la figura 6.1 puede verse, a modo de ejemplo, la representación gráfica de las publicaciones del usuario *neuralogic*, relativo al tema “war”:

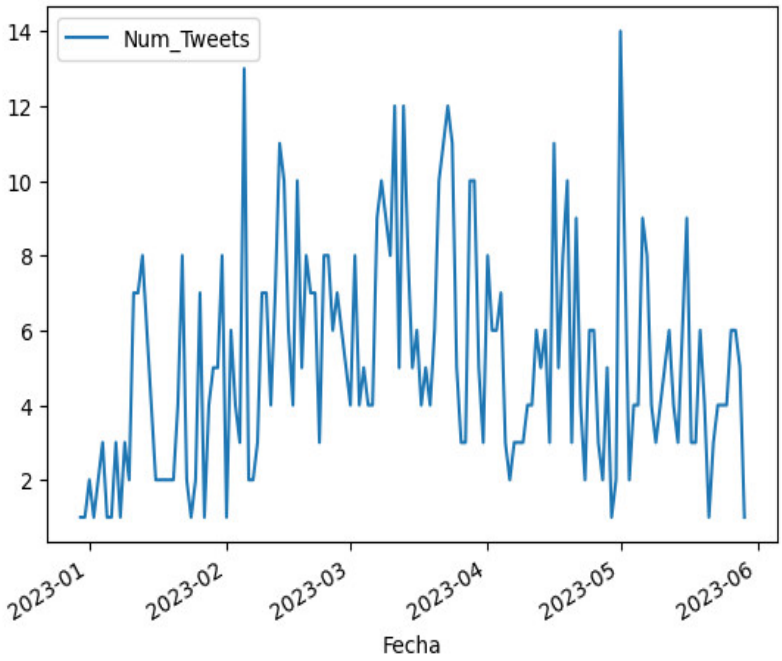

Figura 6.1: Publicaciones neuralogic (war)

Estos datos no ofrecen mucha información útil a primera vista, por lo que se emplea una sencilla herramienta de análisis de series temporales, un filtro de *Hodrick-Prescott*, que permite ver la tendencia que siguen los datos, tal y como se muestra en la figura 6.2

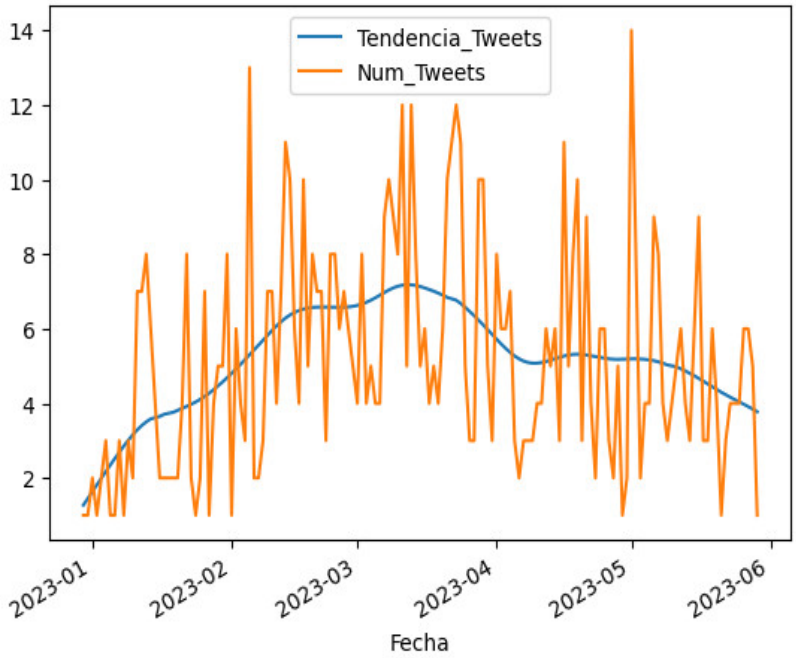

Figura 6.2: Tendencia publicaciones neuralogic (war)

Puede verse claramente la tendencia que sigue la actividad del usuario. A continuación, en la figura 6.3, se compararán tendencias de publicación para ambos usuarios:

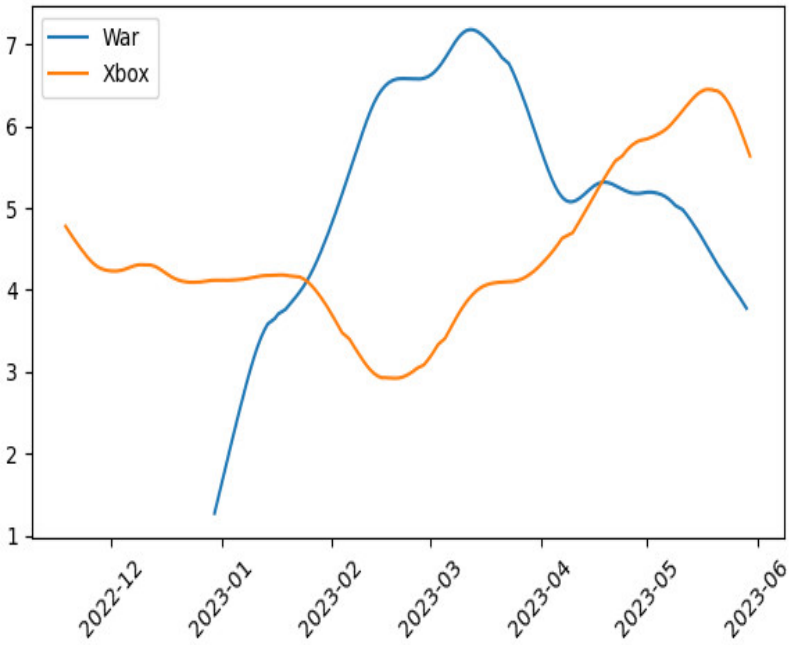

Figura 6.3: Tendencia publicaciones usuarios

Las tendencias de ambos usuarios son contrarias, puesto que en el momento en que el usuario relacionado con el tema “war” se encuentra en su punto de máxima actividad, el otro usuario está en el mínimo. Esto sucede en torno a los meses de febrero y marzo de 2023.

En la figura 6.4 se comparará la tendencia de los sentimientos de los tuits publicados por ambos usuarios:

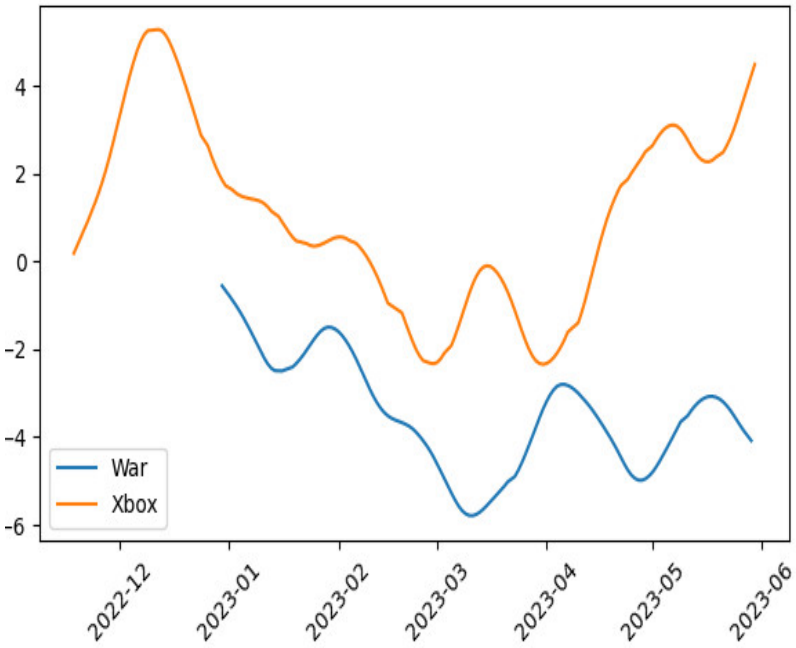

Figura 6.4: Tendencia sentimientos usuarios

Se observa que la tendencia de sentimientos del usuario *neuralogic (war)*, representada en color azul, está por debajo del 0, mientras que la del otro usuario permanece casi siempre positiva. Esto encaja con lo mostrado en la tabla anterior, en la que podían verse los porcentajes de tuits de las emociones por usuario.

6.3. Modelos de clasificación

Empleando los algoritmos y técnicas vistos en el capítulo 4 se pueden crear modelos que permitan clasificar los tuits según la emoción que en ellos subyace. Para ello se transformará el valor numérico asociado a la emoción en una variable de clase de tipo binaria en la cual el 0 representará a los tuits con una emoción menor que 0, y que tomará el valor 1 si la emoción es mayor o igual a 0.

A pesar de que el total de tuits recopilados es de cerca de 400.000, debido a las limitaciones del equipo en el que se ha realizado el TFM, el dataset sobre el que se conformarán los modelos contará con tan sólo 10.000 tuits, perteneciendo 5.000 de ellos a tuits positivos y 5.000 a tuits negativos. De esta forma se asegura el perfecto balanceo de las clases.

Partiendo de ese conjunto de tuits se hallará la matriz TF-IDF, para contar así con una representación numérica de las palabras de cada tuit. Esto hace innecesaria la normalización de los datos, ya que todos se encuentran en el intervalo $[0, 1]$.

La matriz obtenida de esta manera cuenta con una alta dimensionalidad: 10.000 instancias x 21738 variables. Se impone, pues, aplicar algún método de reducción de dimensionalidad como el PCA. Se podrían mostrar de igual manera los resultados sin realizar el PCA a fin de poder hacer una comparativa, pero en el caso del SVC el tiempo requerido para hallar el mejor modelo ascendería a varios días, por lo que se descarta dicha comparativa.

Al realizar el análisis de componentes se obtiene que el número de los mismos que contiene el 90 % de la varianza asciende a 5087, tal y como puede verse en la figura 6.5:

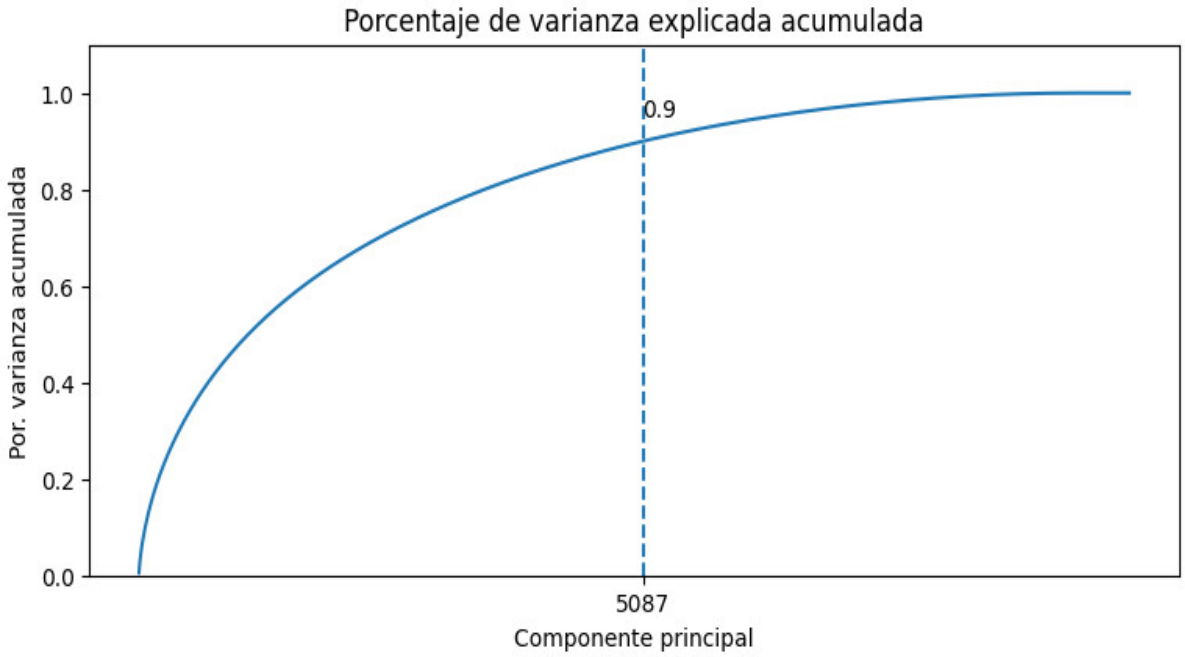

Figura 6.5: Gráfica varianza

Sabiendo el número de componentes adecuado se vuelve a calcular el PCA, esta vez indicándole el número de componentes deseado que se busca a la salida.

Acto seguido se elaboran los modelos tal y como se ha detallado en el capítulo 4. Los resultados obtenidos para cada uno de los algoritmos se muestran en la siguiente tabla:

	Arbol	**Naive-Bayes**	**SVC**	**Random forest**
Accuracy	58,70 %	55,87 %	81,47 %	59,30 %

Los resultados son bastante pobres, excepto en el caso del SVC.

Capítulo 7

Conclusiones y líneas futuras

Las conclusiones y líneas futuras de este trabajo estarán estructuradas de igual forma que el trabajo mismo, por capítulos.

7.1. Sistema recomendador basado en contenido

Tras la implementación del sistema recomendador, los resultados obtenidos para dos usuarios elegidos al azar son los siguientes:

```
El usuario seleccionado es: MyaAlex92810555 (war)

Los 10 usuarios con mayor similitud al usuario MyaAlex92810555 (war) son:

La similitud con el usuario barlas_o (war) es: 0.7829087689663654.
La similitud con el usuario yoseefbasha (war) es: 0.7814178348522803.
La similitud con el usuario Jessica17390323 (war) es: 0.7811834809743862.
La similitud con el usuario 0214Tiff (war) es: 0.7801779378146478.
La similitud con el usuario RauseoJaimar (war) es: 0.7796183113745293.
La similitud con el usuario Brianna41683904 (war) es: 0.7788564544614788.
La similitud con el usuario micamartinez345 (war) es: 0.777845706709635.
La similitud con el usuario DiannaBrow98834 (war) es: 0.77679906398532.
La similitud con el usuario jonathanshunge3 (war) es: 0.7748503159332217.
La similitud con el usuario Cannon2082901 (war) es: 0.7726124177254051.
```

Figura 7.1: Recomendaciones con alta similitud

```
El usuario seleccionado es: artofmanliness (war)

Los 10 usuarios con mayor similitud al usuario artofmanliness (war) son:

La similitud con el usuario FonzGaming (xbox) es: 0.2788456977268521.
La similitud con el usuario GeneralJ501st (xbox) es: 0.2612399904903153.
La similitud con el usuario coach4kindness (war) es: 0.2135440855035479.
La similitud con el usuario DoktaVODKA_ (xbox) es: 0.1943632711875174.
La similitud con el usuario PureDeadGaming (xbox) es: 0.1822546544646473.
La similitud con el usuario GoodReading (war) es: 0.17322088367332583.
La similitud con el usuario SpookyTBG (xbox) es: 0.1699307113891736.
La similitud con el usuario xZer0ex (xbox) es: 0.12096571141758382.
La similitud con el usuario DiannaBrow98834 (war) es: 0.12093414050160431.
La similitud con el usuario JaneEyrePilled (war) es: 0.1201341921897047.
```

Figura 7.2: Recomendaciones con baja similitud

En el caso de las recomendaciones mostradas en la figura 7.1 puede verse que las recomendaciones de usuario son realmente buenas, siendo todos los usuarios recomendados del mismo tema que el usuario seleccionado. Esto es así porque el grado de similitud entre el usuario y los usuarios recomendados es bastante alto.

Sin embargo, no se da la misma situación en las recomendaciones de que pueden verse en la figura 7.2.

Es evidente que la similitud es un factor clave a la hora de proporcionar buenas recomendaciones, y para mejorar este sistema se podría establecer un umbral de similitud, de tal manera que tan sólo los usuarios que superen ese umbral sean recomendados al usuario en cuestión. Otro factor que podría variarse en el sentido de la similitud sería el de investigar acerca de otros métodos de calcular la similitud, como por ejemplo el cálculo de la distancia euclidiana.

7.2. Algoritmos de clasificación

Tras preprocesar los datos, asegurando que el dataset estuviese balanceado, normalizado, mezclado y habiendo realizado una reducción de dimensionalidad mediante un análisis de componentes principales (PCA), se clasifican usuarios en base al tema por el que fueron seleccionados, “war” o “xbox”. Para ello se prueban los siguientes algoritmos:

- Árboles de decisión
- Clasificador Naive-Bayes
- Support Vector Classifier

- Random Forest

Debido a que la dimensionalidad inicial no era demasiado elevada, se pueden comparar los resultados con y sin realizar el PCA, como se muestra en la siguiente tabla:

Dimensionalidad	**Arbol**	**Naive-Bayes**	**SVC**	**Random forest**
Accuracy sin PCA	86.58 %	84.56 %	87.25 %	85.23 %
Accuracy con PCA	81.21 %	70.47 %	87.25 %	80.54 %

Como ha podido observarse, el desempeño ha bajado de forma general, exceptuando para el *Support Vector Classifier*, que curiosamente ha arrojado los mismos resultados tanto con reducción de dimensionalidad como sin ella. Sin embargo, en el caso de no realizar primero el PCA, la búsqueda de hiperparámetros dura 1 hora, mientras que es cuestión de segundos en caso contrario.

Con respecto al árbol de decisión y al clasificador *Naive-Bayes* se ha tardado segundos en ambos casos, mientras que el *random forest* ha tardado 8 minutos sin reducción y segundos con ella.

En general el mejor algoritmo es el SVC, pero en el caso de que el tiempo sea un factor determinante para la creación del modelo ha de optarse por otros algoritmos como el árbol de decisión o el *Naive-Bayes*.

Como posibles líneas futuras se propone el experimentar con más algoritmos, tanto de *machine learning* como de *deep learning*, así como recabar usuarios de otros temas.

7.3. Sistema de recomendación usando Word2Vec

Los resultados obtenidos en el caso de este sistema recomendador no han sido muy buenos. Para el modelo *Word2Vec* se entrenaron ambas arquitecturas, *Skip-Gram* y bolsa de palabras continua (CBOW) para comprobar las diferencias entre ambas.

```
El usuario seleccionado es: trsrpc (xbox)

1          iamsarahlb (xbox)
2             adeandus (war)
3     RealRonaldTrum4 (war)
4       Cannon2082901 (war)
5     JoeShow68347212 (war)
6           JoeyElrod (xbox)
7     Born2beSlicker (xbox)
8      BoulderPreston (war)
9        MarloFranson (war)
10        nyhaze4201 (xbox)
11           ImZalius (war)
12        ShaneSays_ (xbox)
13        jhpsorensen (war)
14     Casper_Raine (xbox)
15          ssjkobe8 (xbox)
16    joshuachuminski (war)
17       GeeboSamuel (xbox)
18       SnoworchidP (xbox)
19            __maiab (war)
20     anchordown902 (xbox)
```

Figura 7.3: Recomendación CBOW

En la figura 8.3 pueden verse que las recomendaciones ofrecidas tras entrenar el modelo con la arquitectura de bolsa de palabras continua o CBOW. Dichas recomendaciones no son buenas, ya que la mitad de los usuarios recomendados pertenecen a un tema y la otra mitad al otro. Por otro lado se da el caso de que las recomendaciones han resultado ser exactamente las mismas para ambas arquitecturas, por lo que se deduce que el tipo de arquitectura usada no es un factor relevante al usar *Word2Vec*, al menos para este tipo de uso.

Con el fin de mejorar este sistema se debería investigar con otras formas de realizar el *embedding* de los usuarios, como por ejemplo los *TF-IDF Average Word2Vec Vectors*.

7.4. Análisis de sentimientos

Esta sección consta de dos subsecciones diferentes, aunque estén encaminadas al mismo fin.

7.4.1. Análisis exploratorio de datos

En la primera sección se muestra el porcentaje de tuits positivos, negativos y neutros de dos usuarios seleccionados al azar de cada uno de los temas elegidos, “war” y “xbox”.

	neuralogic (war)	**MattsGeekCorner (xbox)**
Tuits positivos	28.18 %	38.55 %
Tuits neutros	31.45 %	23.66 %
Tuits negativos	40.37 %	37.79 %

Esta tabla se muestra claramente que el usuario asociado al tema más negativo tiene menor porcentaje de tuits positivos, en concreto un 10 %.

Se muestran también las tendencias de publicaciones por día y sentimientos por día para cada uno de los dos usuarios, a fin de poder establecer una comparativa:

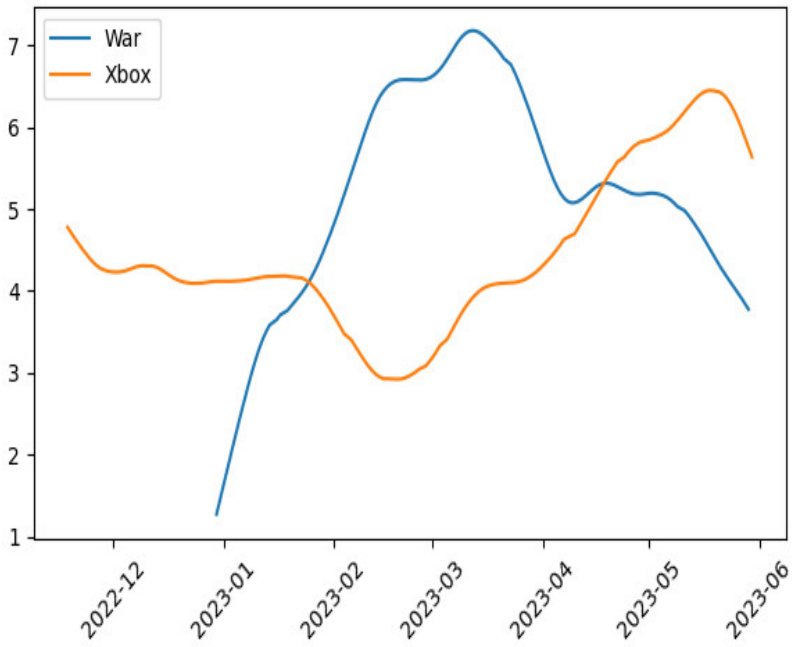

Figura 7.4: Tendencia publicaciones usuarios

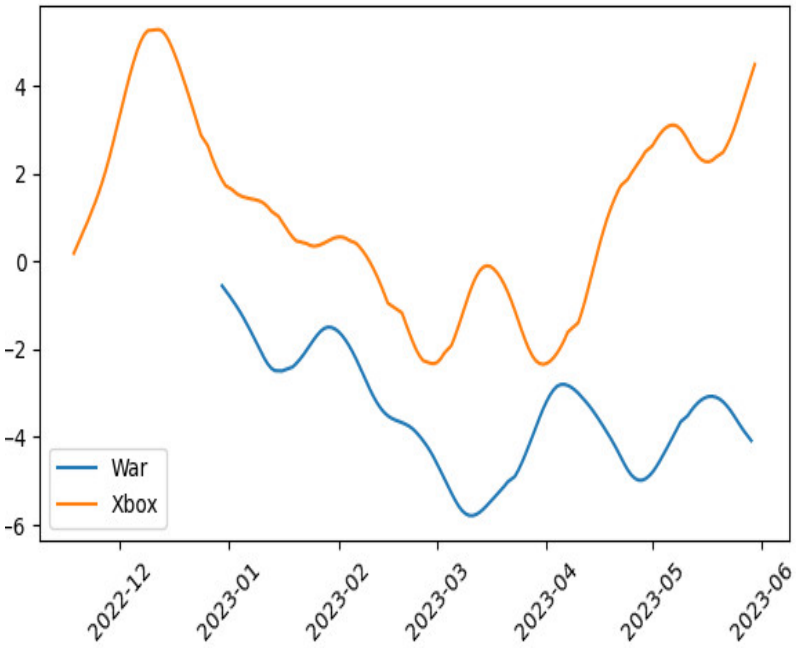

Figura 7.5: Tendencia sentimientos usuarios

En torno a los meses de febrero y marzo la positividad alcanza niveles mínimos para ambos usuarios, por lo que se deduce que algún evento negativo tuvo lugar en esos meses. Sin embargo, el nivel de actividad del usuario más negativo aumenta en la época más negativa. Debido a que la guerra desgraciadamente sigue en curso, podría inferirse, ya que el usuario más negativo está relacionado con este tema, que dicho usuario tiene un gran interés por este tema y está involucrado en él, al menos a nivel de redes sociales, a pesar de ser un tema que genera negatividad.

Por otra parte, el usuario del tema “xbox” alcanza su nivel mínimo tanto de actividad como de positividad en este mismo momento, por lo que podría decirse que no es un tema que le interese activamente pero que sí le afecta a nivel emocional, lo cual es factible ya que la guerra ha generado mucha repercusión negativa a nivel internacional.

Sin embargo, esto es sólo una aproximación en base a los datos visualizados, ya que existe multitud de factores exógenos que podrían combinarse para dar lugar al comportamiento visto en las gráficas.

La posible ampliación de esta parte del capítulo pasaría por la obtención de datos más antiguos de los usuarios, para poder aplicar herramientas de análisis de series temporales y relacionar los datos de forma más efectiva con la realidad, incluso pudiendo elaborar modelos predictivos de la actividad futura de los usuarios.

También se debería investigar sobre otros métodos para obtener los sentimientos de un texto, puesto que un mensaje negativo dicho de forma irónica podría pasar por positivo, cuestión que no podría ser detectada empleando el método usado en el presente trabajo.

7.4.2. Modelos de clasificación

En la sección dedicada a la clasificación se han entrenado varios modelos de igual forma que en el capítulo 4, para clasificar los tuits obtenidos de los usuarios en positivos o negativos. Los resultados han sido los siguientes:

	Arbol	**Naive-Bayes**	**SVC**	**Random forest**
Accuracy	58,70 %	55,87 %	81,47 %	59,30 %

En general los resultados han sido muy malos, exceptuando los del SVC. En el caso de este algoritmo sus resultados fueron los últimos en ser obtenidos debido a que la búsqueda de los mejores hiperparámetros por parte de *GridSearchCV* tardó cinco horas. La tendencia, vistos los otros resultados, apuntaba a que el SVC iba a tener un accuracy igual de malo, no siendo así al final. Queda demostrado que es un algoritmo altamente efectivo, cabe suponer que por la flexibilidad que le aportan las funciones kernel.

Como caminos futuros de investigación sería interesante comprobar la eficacia de *RandomizedSearchCV*, una alternativa a *GridSearchCV* para la búsqueda de hiperparámetros que, en lugar de buscar la mejor combinación por fuerza bruta, realiza un número de pruebas menor y aleatorio, por lo que el tiempo de búsqueda se reduce pero tampoco garantiza encontrar los mejores parámetros.

También se podrían dividir los tuits en positivos, negativos y neutros, para así investigar sobre la implementación de modelos de clasificación multiclase.

Capítulo 8

Summary and Conclusions

The conclusions and future lines of this work will be structured in the same way as the work itself, by chapters.

8.1. Content-based recommender system

After the implementation of the recommender system, the results obtained for two randomly chosen users are the following:

```
El usuario seleccionado es: MyaAlex92810555 (war)

Los 10 usuarios con mayor similitud al usuario MyaAlex92810555 (war) son:

La similitud con el usuario barlas_o (war) es: 0.7829087689663654.
La similitud con el usuario yoseefbasha (war) es: 0.7814178348522803.
La similitud con el usuario Jessica17390323 (war) es: 0.7811834809743862.
La similitud con el usuario 0214Tiff (war) es: 0.7801779378146478.
La similitud con el usuario RauseoJaimar (war) es: 0.7796183113745293.
La similitud con el usuario Brianna41683904 (war) es: 0.7788564544614788.
La similitud con el usuario micamartinez345 (war) es: 0.777845706709635.
La similitud con el usuario DiannaBrow98834 (war) es: 0.77679906398532.
La similitud con el usuario jonathanshunge3 (war) es: 0.7748503159332217.
La similitud con el usuario Cannon2082901 (war) es: 0.7726124177254051.
```

Figura 8.1: Recommendations with high similarity

```
El usuario seleccionado es: artofmanliness (war)

Los 10 usuarios con mayor similitud al usuario artofmanliness (war) son:

La similitud con el usuario FonzGaming (xbox) es: 0.2788456977268521.
La similitud con el usuario GeneralJ501st (xbox) es: 0.2612399904903153.
La similitud con el usuario coach4kindness (war) es: 0.2135440855035479.
La similitud con el usuario DoktaVODKA_ (xbox) es: 0.1943632711875174.
La similitud con el usuario PureDeadGaming (xbox) es: 0.1822546544646473.
La similitud con el usuario GoodReading (war) es: 0.17322088367332583.
La similitud con el usuario SpookyTBG (xbox) es: 0.1699307113891736.
La similitud con el usuario xZer0ex (xbox) es: 0.12096571141758382.
La similitud con el usuario DiannaBrow98834 (war) es: 0.12093414050160431.
La similitud con el usuario JaneEyrePilled (war) es: 0.1201341921897047.
```

Figura 8.2: Recommendations with low similarity

In the case of the recommendations shown in figure 8.1 it can be seen that the user recommendations are really good, with all the recommended users being of the same topic as the selected user. This is so because the degree of similarity between the user and the recommended users is quite high.

However, the same situation does not occur in the recommendations that can be seen in the figure 8.2.

It is evident that similarity is a key factor when it comes to providing good recommendations, and to improve this system a similarity threshold could be established, in such a way that only users that exceed that threshold are recommended to the user. Another factor that could be varied in the sense of similarity would be to investigate other methods of calculating similarity, such as the calculation of the euclidean distance.

8.2. Classification algorithms

After preprocessing the data, ensuring that the dataset was balanced, normalized, mixed, and having performed dimensionality reduction using Principal Component Analysis (PCA), users are classified based on the theme for which they were selected, “war” or “xbox”. To do this, the following algorithms are tested:

- Decision trees
- Naive-Bayes classificator
- Support Vector Classifier
- Random Forest

Since the initial dimensionality was not too high, the results can be compared with and without performing the PCA, as shown in the following table:

Dimensionality	Tree	Naive-Bayes	SVC	Random forest
Accuracy without PCA	86.58 %	84.56 %	87.25 %	85.23 %
Accuracy with PCA	81.21 %	70.47 %	87.25 %	80.54 %

As it has been observed, the performance has decreased in general, except for the *Support Vector Classifier*, which has produced the same results both with and without dimensionality reduction. However, in the case of not performing the PCA first, the hyperparameter search lasts 1 hour, whereas it is a matter of seconds otherwise.

Regarding the decision tree and the *Naive-Bayes* classifier, it took seconds in both cases, while the *random forest* took 8 minutes without reduction and seconds with it.

In general, the best algorithm is the SVC, but if time would be a determining factor for the creation of the model, other algorithms such as the decision tree or *Naive-Bayes* must be chosen.

As possible future lines, it is proposed to experiment with more algorithms, both from *machine learning* and *deep learning*, as well as gathering users of other topics.

8.3. Recommender system using Word2Vec

The results obtained in the case of this recommender system have not been very good. For the *Word2Vec* model, both architectures, *Skip-Gram* and continuous bag of words (CBOW) were trained to check the differences between them.

```
El usuario seleccionado es: trsrpc (xbox)

1          iamsarahlb (xbox)
2             adeandus (war)
3     RealRonaldTrum4 (war)
4       Cannon2082901 (war)
5     JoeShow68347212 (war)
6           JoeyElrod (xbox)
7     Born2beSlicker (xbox)
8      BoulderPreston (war)
9        MarloFranson (war)
10        nyhaze4201 (xbox)
11           ImZalius (war)
12        ShaneSays_ (xbox)
13        jhpsorensen (war)
14      Casper_Raine (xbox)
15          ssjkobe8 (xbox)
16    joshuachuminski (war)
17       GeeboSamuel (xbox)
18       SnoworchidP (xbox)
19            __maiab (war)
20     anchordown902 (xbox)
```

Figura 8.3: CBOW Recommendation

In figure 8.3 recommendations offered after training the model with the continuous bag of words architecture or CBOW can be seen. Such recommendations are not good, since

half of the recommended users belong to one topic and half to the other. On the other hand, recommendations have turned out to be exactly the same for both architectures, so it follows that the type of architecture used is not a relevant factor when using *Word2Vec*, at least for this kind of use.

In order to improve this system, other ways of *embedding* users should be investigated, such as *TF-IDF Average Word2Vec Vectors*.

8.4. Sentiment analysis

This chapter consists of two different sections, although they are aimed at the same end.

8.4.1. Exploratory analysis of the data

This chapter consists of two different sections, although they are aimed at the same end. The first section shows the percentage of positive, negative and neutral tweets from two randomly selected users for each of the chosen topics, the user *neuralogic* for the topic “war” and the user *MattsGeekCorner* for “xbox”.

	neuralogic (war)	**MattsGeekCorner (xbox)**
Positive tweets	28.18 %	38.55 %
Neutral tweets	31.45 %	23.66 %
Negative tweets	40.37 %	37.79 %

This table clearly shows that the user associated with the most negative topic has the lowest percentage of positive tweets, specifically 10 %.

The trends of publications per day and sentiments per day for each of the two users are also shown, in order to be able to establish a comparison:

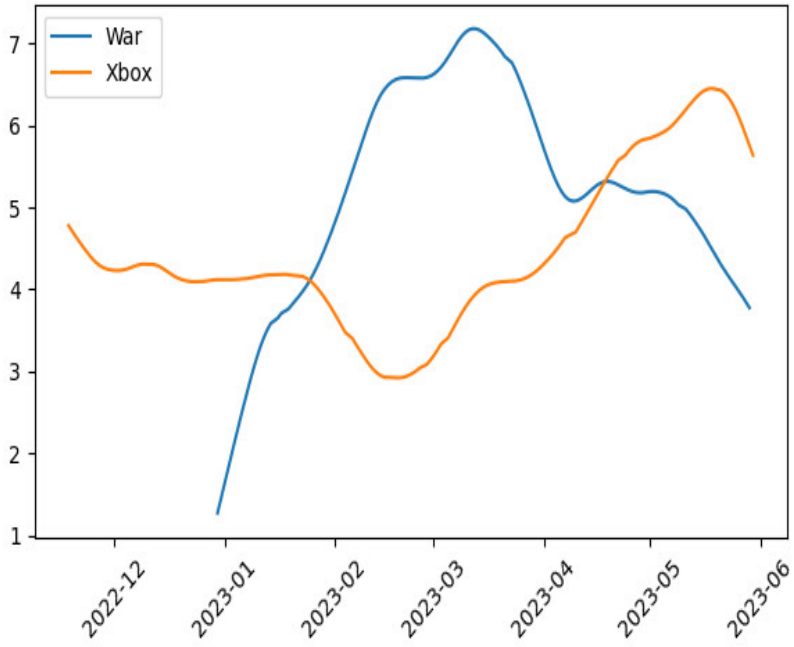

Figura 8.4: Users posting trend

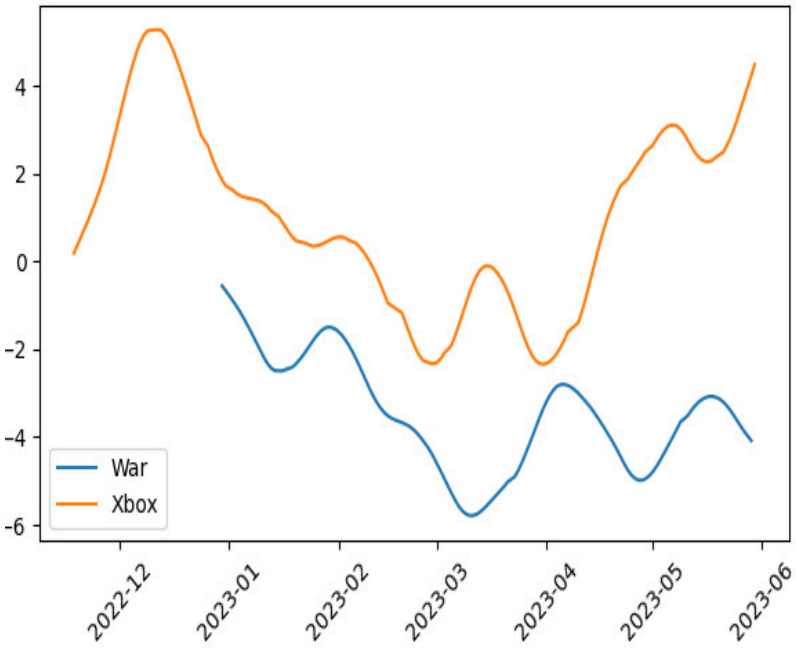

Figura 8.5: Users sentiments trend

Around the months of February and March, positivity reaches minimum levels for both users, so it can be deduced that some negative event took place in those months. However, the activity level of the most negative user increases in the most negative season. Due to the fact that the war is unfortunately still ongoing, it could be inferred, since the most negative user is related to this topic, that said user has a great interest in this topic and is involved in it, at least at the level of social networks, to despite being a subject that generates negativity.

On the other hand, the user of the topic “xbox” reaches its minimum level of both activity and positivity at this very moment, so it could be said that it is not a topic that interests him actively but that it does affect him on an emotional level, which is feasible since the war has generated a lot of negative repercussions internationally.

However, this is only an approximation based on the displayed data, since there are many exogenous factors that could combine to give rise to the behavior seen in the graphs.

The possible extension of this part of the chapter would involve obtaining older data from users, in order to apply time series analysis tools and relate the data more effectively to reality, even being able to develop predictive models of future activity of the users.

Other methods to obtain the feelings of a text should also be investigated, since a negative message said in an ironic way could pass for positive, an issue that could not be detected using the method used in the present work.

8.4.2. Classification models

In the section dedicated to classification, several models have been trained in the same way as in chapter 4, to classify the tweets obtained from users as positive or negative. The results are shown in the next table:

	Tree	**Naive-Bayes**	**SVC**	**Random forest**
Accuracy	58,70 %	55,87 %	81,47 %	59,30 %

In general, the results have been very bad, except for those of the SVC. In the case of this algorithm, its results were the last to be obtained because the search for the best hyperparameters by *GridSearchCV* took five hours. The trend, seen the other results, was that the SVC was going to have an equally bad accuracy, but that was not the case in the end. It has been shown that it is a highly effective algorithm, presumably due to the flexibility provided by the kernel functions.

As future research paths, it would be interesting to check the effectiveness of *RandomizedSearchCV*, an alternative to *GridSearchCV* for hyperparameter search that, instead of searching for the best combination by brute force, performs a random and smaller number of tests, so the search time is reduced but it does not guarantee to find the best parameters either.

Tweets could also be divided into positive, negative and neutral, in order to investigate the implementation of multiclass classification models.

Capítulo 9

Presupuesto

El presupuesto de este trabajo consta principalmente del coste de personal, teniendo en cuenta el número de horas requeridas para el desarrollo del mismo. El cálculo del tiempo se realiza en base a la duración del proyecto; 150 horas, acorde con lo establecido en la Resolución de 29 de octubre de 2020, de la Universidad de La Laguna, por la que se publica el plan de estudios del Máster en Ciberseguridad e Inteligencia de Datos, donde se establece una duración de 6 créditos ECTS para el Trabajo de Fin de Máster. Un crédito ECTS equivale a 25 horas de trabajo; 25 * 6 = 150.

El coste total de personal equivale a 2400€, asumiendo un coste por hora de 16€.

Bibliografía

Asghar, M.Z., Sattar, A., Khan, A., Ali, A., Masud Kundi, F., Ahmad, S., 2019. Creating sentiment lexicon for sentiment analysis in urdu: The case of a resource-poor language. Expert systems 36, e12397–n/a.

Błajda, J., Kucab, A., Miazga, A., Masłowski, M., Kopańska, M., Nowak, A., Barnaś, E., 2023. Google trends analysis reflecting internet users' interest in selected terms of sexual and reproductive health in ukraine. Healthcare (Basel) 11, 1541.

Hsu, C.w., Chang, C.c., Lin, C.J., 2003. A practical guide to support vector classification chih-wei hsu, chih-chung chang, and chih-jen lin .

James, Gareth; Witten, D.H.T., 2013. An Introduction to Statistical Learning. Springer New York.

Matus, R., 2010. Estadística. Instituto Politécnico Nacional.

Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013. Efficient estimation of word representations in vector space .

Müller, A.C., Guido, S., 2016. Introduction to Machine Learning with Python: A Guide for Data Scientists. O'Reilly Media, Incorporated, Sebastopol.

Nielsen, F., 2011. A new anew: Evaluation of a word list for sentiment analysis in microblogs, in: CEUR Workshop Proceedings, pp. 93–98.

Tabii, Y., Lazaar, M., Al Achhab, M., Enneya, N., 2018. Hashtag recommendation using word sequences' embeddings, in: Big Data, Cloud and Applications. Springer International Publishing AG, Switzerland. volume 872 of *Communications in Computer and Information Science*, pp. 131–143.

Tan, P.N., 2014. Introduction to data mining. Pearson new international edition.. ed., Pearson Education, Boston, MA.

VanderPlas, J., 2022. Python Data Science Handbook, 2nd Edition. O'Reilly Media, Inc.

Weng, Y., Yu, W., Lin, R., Tang, Y., He, C., 2023. Scholarrec: A user recommendation system for academic social network, in: Communications in Computer and Information Science. volume 1681, pp. 28–41.

Wu, Y.C., Yin, F., Liu, C.L., 2017. Improving handwritten chinese text recognition using neural network language models and convolutional neural network shape models. Pattern recognition 65, 251–264.

Zhang, A., Lipton, Z.C., Li, M., Smola, A.J., 2021. Dive into deep learning. arXiv preprint arXiv:2106.11342 .