UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

21|22

Trabajo fin de máster

El impacto de los sistemas de recomendación en la propagación de la desinformación en redes sociales

Alfonso de Paz García

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

Universidad Autónoma de Madrid

Escuela Politécnica Superior

Proyecto para la obtención del título de
Máster en Máster Universitario en Investigación e Innovación en Inteligencia Computacional y Sistemas Interactivos
por la Universidad Autónoma de Madrid

Tutor del trabajo fin de máster:
Alejandro Bellogín Kouki

El impacto de los sistemas de recomendación en la propagación de la desinformación en redes sociales

Alfonso de Paz García

Alfonso de Paz García
El impacto de los sistemas de recomendación en la propagación de la desinformación en redes sociales

Alfonso de Paz García
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Agradecimientos

En primer lugar me gustaría agradecer a Alejandro Bellogín por todo el buen trato que ha tenido conmigo, tanto como tutor en el TFG como del TFM, por guiarme en ambos trabajos desde el principio y siempre estar dispuesto a ayudarme y responder todas mis dudas.

A toda mi familia por apoyarme desde siempre y haber estado encima ayudando cuando lo necesitaba, desde que comencé mis estudios en la educación primaria hasta la actualidad, gracias por enseñarme la importancia de la constancia y el trabajo si uno quieres lograr sus objetivos.

Y como no, también a todos mis compañeros de la carrera y del máster que me han estado acompañando en mis últimos años de estudiante y me han hecho más agradable el camino.

Resumen

En los últimos años, la expansión de los teléfonos móviles inteligentes ha provocado una explosión en el consumo de servicios en Internet. Con este aumento del consumo ha dado lugar la aparición de una gran cantidad de portales de noticias que ahora realizan el trabajo que antes hacían unos pocos medios tradicionales, a su vez, las redes sociales han adquirido una mayor importancia a la hora de conocer e incluso influenciar en la opinión popular.

A simple vista, este aumento de la competitividad provocada por el hecho de que cualquier persona pueda competir como medio de información, parece un punto positivo. Sin embargo, también crece la cantidad de intentos de generar noticias desinformativas con el objetivo de viralizar un relato alternativo y, junto con las redes sociales como herramientas de difusión, manipular la opinión de los usuarios, ya sea para un beneficio económico o político.

Con lo cual, en este Trabajo Final de Máster, basándose en la idea de que los sistemas de recomendación de estas redes sociales son los encargados de ofrecer las noticias a los usuarios, se propone estudiar si la recomendación es la principal responsable de viralizar la desinformación además de conocer el nivel de impacto que tiene, en concreto, se plantea analizar qué algoritmos de recomendación son más propensos a devolver dicha desinformación.

Para ello, se ha creado una base de datos haciendo uso de noticias provenientes de una agencia verificadora, Politifact, junto a la interacción de usuarios en una red social (Twitter), que comentan estas noticias. Tras esto, se ha usado dicha base de datos, formada por contenido informativo y desinformativo, para recomendar usando un amplio catálogo de algoritmos: no personalizados (aleatorio y más popular), basados en vecinos próximos, en factorización de matrices y en redes neuronales.

Como conclusión, se ha comprobado que hay ciertos algoritmos donde la cantidad de desinformación que recomiendan está muy ligada con la existente en los datos de entrenamiento, este es el caso de la recomendación basada en popularidad. En general, no se ha encontrado una técnica que desinforme menos que el resto en cualquier situación, pero sí se ha realizado un estudio preliminar razonablemente completo como para conocer qué métodos de recomendación son más aconsejables según la desinformación existente en los datos.

Palabras clave

Desinformación, fake news, verificadores, sistemas de recomendación, minería de datos, recuperación de información

Abstract

In recent years, the expansion of smartphones has led to an explosion in the consumption of web services. This increase in consumption has led to the emergence of a large number of news portals that, nowadays, take the space that was previously used by a few traditional media. At the same time, social networks have become more important in knowing and even influencing popular opinion.

At first sight, this increase in competitiveness caused by the fact that anyone can compete as a means of information seems to be a positive point. However, there is also a growing number of attempts to generate misinformative news aiming at viralizing an alternative narrative and, together with social networks as diffusion tools, manipulating the opinion of users, either for economic or political benefits.

Therefore, in this Master's thesis, based on the idea that the recommendation systems of these social networks are responsible for providing news to users, we propose to study whether the recommendation is the main responsible for viralizing misinformation in addition to knowing the level of impact it has. In particular, we propose to analyze which recommendation algorithms are more likely to return such misinformation.

For this purpose, a database was created using news items from a fact-checking agency, Politifact, together with the interaction of users on the Twitter social network. After that, this database, formed by informative and misinformative content, has been used to recommend using a wide range of algorithms: non-personalized (random and most popular), based on nearest neighbors, matrix factorization, and neural networks.

In summary, we have found that there are certain algorithms where the amount of misinformation they recommend is closely linked to the amount of misinformation in the training data, as in the case of recommendation based on popularity. In general, we have not found a technique that misinforms less than the rest in every situation, nonetheless we have performed a preliminary study which is reasonably complete to know which recommendation methods are more appropriate according to the existing misinformation in the data.

Keywords

Misinformation, fake news, fact-checkers, recommender systems, data mining, information retrieval

Índice

Listas

Lista de códigos

Lista de ecuaciones

Lista de figuras

Lista de tablas

1

Introducción

En este capítulo se detallará la motivación que ha dado lugar a este trabajo, los objetivos que se buscan alcanzar y la estructura general de este documento.

1.1. Motivación del proyecto

En la última década, Internet ha ido cogiendo cada vez más fuerza como herramienta informativa y, poco a poco, ha conseguido reemplazar en cuanto a importancia a los medios tradicionales como la prensa escrita. Esto es debido principalmente a su enorme accesibilidad y la gran facilidad para comunicar noticias de última hora en cualquier momento y lugar.

Sin embargo, en los últimos años, esta herramienta ha ido adquiriendo un carácter más negativo, usándose para desinformar con el objetivo de crear un relato alternativo y así engañar y manipular nuestra opinión. Estas noticias se denominan *misinformation* (desinformación, en inglés) o, de forma más vulgar, *fake news*. Uno de los ejemplos más sonados fueron las elecciones de Estados Unidos de 2016, que se vio expuesta a una enorme campaña desinformativa en Facebook [1]. No obstante, no basta solo con crear estas noticias desinformativas, sino que también es necesario que se divulguen al gran público, y es aquí donde entran en juego los sistemas de recomendación.

Hoy en día, la mayoría de contenido que consumimos lo hacemos de forma involuntaria, es decir, no hemos hecho una búsqueda directa sobre ello. Las compañías de estos servicios que usamos, con el objetivo de mantenernos el mayor tiempo en la plataforma y así sacar un beneficio normalmente económico mucho mayor, tratan de mostrar contenido que nos sea interesante y así mantenernos entretenidos. Para tener un conocimiento mayor y conocer algunos inconvenientes, nos interesa estudiar si se cumple, y en qué nivel, que los artículos con contenido claramente no verídico son los que el sistema de recomendación tiende a considerar más interesantes para el usuario, si solo ocurre bajo ciertas condiciones, o si se trata de meramente algo casual.

1.2. Objetivos

Para poder llevar a cabo el estudio comentado anteriormente es necesario desarrollar un conjunto de datos formado por contenido informativo y desinformativo para examinar este problema a fondo. También, como nos interesa conocer su efecto en los sistemas de recomendación, hay que desarrollar e integrar un entorno de recomendación donde realizar pruebas del conjunto de datos generado y conocer qué técnicas de recomendación son más propensas a recomendar desinformación.

El primer objetivo a abordar en este Trabajo Fin de Máster será el desarrollo de esa base de datos de desinformación mencionada para la cual habrá que seguir tres pasos principales:

1.– Proceso de búsqueda de noticias reales y desinformativas etiquetadas correctamente por su veracidad.

2.– Extracción automatizada de las noticias.

3.– Extracción automatizada de información que relacione estas noticias con interacción entre usuarios, para ello se hará uso de la red social de Twitter.

Toda la información extraída será procesada para obtener el conjunto de datos final y, tras esto, se podrá comenzar con el objetivo final que se trata de un estudio del comportamiento de los sistemas de recomendación con el dataset generado. Este proceso se puede dividir en tres pasos:

1.– Proceso de selección de los algoritmos de recomendación más representativos en el área de estudio.

2.– Entrenar dichos algoritmos con distintos subconjuntos del dataset para probar distintas situaciones.

3.– Analizar los resultados devueltos usando diversas métricas orientadas a medir la cantidad de desinformación.

1.3. Estructura del trabajo

El documento actual está dividido en varios capítulos detallados a continuación:

Capítulo 1 - Introducción Primer acercamiento al trabajo, donde se introducen las ideas iniciales, motivación y objetivos de este, y la estructura del documento.

Capítulo 2 - Estado del arte Se realiza un estudio sobre la situación actual de las tecnologías usadas en este trabajo, tales como los procesos de minería web para extracción de información, los sistemas de recomendación, además de realizar una definición sobre la desinformación y sus tipos.

Capítulo 3 - Sistema desarrollado En este capítulo se detalla toda la implementación de los elementos desarrollados, describiendo así el proceso de búsqueda y extracción de los datos, filtrado, almacenamiento y la generación final del dataset para aplicarlos en un sistema de recomendación y así poder obtener resultados.

Capítulo 4 - Experimentos A lo largo de esta sección se realiza un análisis de los datos extraídos, además, se explica el proceso seguido para generar los ficheros que se usarán para recomendar con los distintos algoritmos de recomendación. Finalmente, se detalla el proceso de recomendación y se realiza un análisis de los resultados.

Capítulo 5 - Conclusiones y trabajo futuro Se establecen las conclusiones finales de este trabajo y se realiza un breve estudio sobre las posibles mejoras que se podrían desarrollar en el futuro.

2

Estado del arte

En este capítulo se va a hablar sobre las distintas metodologías y tecnologías usadas a lo largo del proyecto, tales como los procesos de minería web para extracción de información, los sistemas de recomendación, además de realizar una definición sobre la desinformación y sus tipos. También se expondrán ejemplos de las herramientas existentes así como de las usadas finalmente en el desarrollo del proyecto.

2.1. Minería web

La minería web es una de las técnicas más importantes para la recuperación de información y obtención de patrones vía Internet. Se centra en la obtención de información no estructurada de forma automática tanto del contenido, como de la estructura y los usuarios [2].

Es una de las tareas principales que los motores de búsqueda deben llevar a cabo para poder obtener los datos a procesar, tanto para el posicionamiento de contenidos como para su indexación. Además, forma parte de cualquier actividad donde intervengan grandes cantidades de información, como las tiendas online, los servicios de publicidad y cualquier web con contenido que requiera estructurarlo, por ejemplo, Amazon, Youtube, etc.

Podemos diferenciar tres tipos de minería web en función del tipo de información que se desea obtener:

Minería de contenido La web no solo está formada por HTML, sino que también hay una gran cantidad de documentos, como textos, PDF, imágenes, vídeos, audios. Además, todos estos documentos contienen información adicional como son los metadatos, donde podemos encontrar autores, fecha de creación, de modificación, etc. Toda esta información se puede recopilar y estructurar para un fácil acceso o para sacar ciertas conclusiones de su procesamiento.

Minería de estructura A partir del uso de hipervínculos se trata de ir saltando entre páginas e ir conociendo la estructura de cualquier web. Esto puede servir para conocer la distribución

y tener una idea de cómo fluye el tráfico en ella, además de encontrar posibles errores que dificulten el acceso a sub-páginas importantes.

Minería de uso En este caso se trata de obtener la mayor cantidad de información posible sobre las acciones que realizan los usuarios, para ello, se suele hacer uso de los logs, que registran las acciones sobre la web, la fecha de acceso, nombre de usuario o IP que realizó dicha acción, entre otras cosas.

En el proceso de minería web se pueden distinguir cuatro acciones principales [3]:

1.– Obtención de la información

2.– Procesamiento

3.– Descubrimiento de patrones

4.– Análisis de patrones

Una de las mayores ventajas de este proceso es la capacidad para estructurar y detectar patrones en grandes cantidades de información de forma automática. Esto es muy útil para una gran cantidad de negocios online, como el marketing digital personalizado que nos aporta recomendaciones más útiles y permite un mayor conocimiento del cliente, lo que da lugar a una mayor fidelidad que se traduce en mayores beneficios. Sin embargo, no todo está relacionado con el comercio, también se trabaja en agencias de seguridad que permiten descubrir y pelear contra movimientos terroristas, detectar actividades criminales, como fraude fiscal y blanqueo de dinero o incluso aplicaciones en la salud, para realizar diagnósticos más exactos y mejorar la calidad de vida [4–6]. Además, es una herramienta muy útil para la detección de informaciones falsas que se están viralizando en redes sociales, como veremos a lo largo de este trabajo.

No obstante, también tiene su parte negativa. Cuando se hace uso de estas tecnologías para procesar información de ámbito personal sin el debido consentimiento de los usuarios o sin garantizar una correcta seguridad de los datos puede dar lugar a una violación de la privacidad. No solo eso, el simple tratamiento de información privada, aunque sea tratada de forma anónima para crear un perfil digital de los usuarios, está sujeto a ciertas licencias éticas con las que no todo el mundo está de acuerdo y, es por ello, que en los últimos años se ha endurecido la ley para proteger a la población, por ejemplo con la Regulación General de Datos Personales (GDPR, por sus siglas en inglés) que entró en vigor el 25 de mayo de 2018 [7].

2.1.1. Obtención de información: crawling

Uno de los componentes clave de la minería web es el Crawler, también llamado araña o spider-bot, en inglés. Este módulo se encarga de solicitar páginas web, descargar su contenido y, tras esto, acceder a todos sus hipervínculos para descargarlos y así sucesivamente hasta que se cumpla alguna

condición de parada.

El crawler se inicia con un grupo de URLs, llamadas semillas, puestas manualmente. Tras esto, se procede a descargar su contenido y recorrer sus hipervínculos recursivamente siguiendo una serie de políticas establecidas previamente. Estas políticas pueden ir desde normas muy generales, como aquellas marcadas en el propio HTML. Por ejemplo, en las etiquetas de hipervínculos se puede establecer el atributo *follow/nofollow* para decirle a la araña si deberá seguir o no ese enlace. También tenemos la posibilidad de indicar en la web el archivo `robots.txt`, orientado a los bots de los buscadores, donde se establecen una serie de normas como páginas anidadas que deberá rastrear y cuáles no. Además, podemos establecer en el crawler políticas más específicas, como mantenerse dentro de un dominio, detenerse al cumplirse cierta condición o evitar repetir enlaces [8].

A la par, toda esta información que se va descargando se suele filtrar. Mediante lenguajes como XPath [9], los crawlers son capaces de acceder a información concreta de archivos XML, como es el caso del HTML con la que están construidas las páginas. Después de extraer el contenido deseado se puede pasar a su análisis o almacenarlo en una base de datos hasta que se requiera su uso. En resumen, el crawling va a permitir a este proyecto extraer una gran cantidad de datos que sería difícil obtener de otra forma y mucho menos manualmente, además de quedarse únicamente con aquella información concreta que necesitemos y almacenarla de forma organizada. Existen muchos crawlers para usar en distintos lenguajes, sin embargo, se puede desarrollar uno propio usando librerías como JSoup en Java o Pyspider y Scrapy en Python.

JSoup Es una biblioteca de Java cuya función es la de extraer y tratar datos de la web, principalmente documentos HTML [10].

Pyspider Una biblioteca de Python para realizar web crawling, cuenta con una interfaz de usuario web para hacer más accesible el trabajo. Sin embargo, la comunidad que hay detrás no es la más grande y todavía está en desarrollo [11].

Scrapy Otra librería de Python para desarrollar un crawler, pero, a diferencia de la anterior, cuenta con una gran comunidad detrás y una muy buena documentación, además de ser un proyecto sólido [12]. Por preferencia del lenguaje y experiencia previa, esta es la librería escogida para la parte de crawling de este proyecto.

2.1.2. APIs, qué son y para qué sirven

A lo largo de esta década, las aplicaciones o servicios multiplataforma tales como Youtube, Twitter, Netflix, etc, han experimentado un gran crecimiento y se han tenido que adaptar a los numerosos dispositivos para consumir contenidos que existen en la actualidad: teléfonos, televisores, ordenadores, tablets, etc. La filosofía de crear aplicaciones centradas en un solo sistema queda obsoleta y es necesario flexibilizar el desarrollo. De esta forma surgen las APIs: se desarrolla un servidor único y

universal, separado del cliente, y será cada tipo de cliente (la app de Android, Windows o iOS, por ejemplo) quien se encargará de comunicarse con el servidor a través de un lenguaje universal.

Una API, término cuya abreviatura viene de **Application Programming Interfaces** en inglés, engloba un conjunto de protocolos y definiciones a la hora de desarrollar un software para establecer cómo realizar la comunicación entre dos módulos, por lo general, cliente y servidor. Normalmente se usa la arquitectura API Rest, donde la comunicación se hace por medio del protocolo HTTP y los datos enviados en formato JSON. Esta última se trata de la tecnología usada por todos los servicios web en la actualidad gracias a las facilidades del protocolo usado, donde con simples peticiones HTTP cualquier aplicación escrita en cualquier lenguaje se puede comunicar con un servidor [13].

Otra de las ventajas es la de permitir el uso de un servicio o tener acceso a información sin la necesidad de hacer de intermediario ni de que se conozca con detalle cómo está desarrollada la base de datos o el resto del sistema. Abrir las APIs al usuario general tiene grandes ventajas, aunque también inconvenientes si no se protege bien, a la hora de hacer crecer una comunidad alrededor de un servicio. Por ejemplo, para crear aplicaciones en dispositivos donde no está disponible, Samsung podría desarrollar la aplicación de Youtube para sus televisores y así ambas empresas se verían beneficiadas. También en el caso de las redes sociales, donde se tiene una gran cantidad de información valiosa, ya que se puede permitir su explotación e incluso monetizarla, como es el caso de Twitter pero manteniendo el control (o incluso revocándolo) de la información que se comparte [14].

En el caso de este proyecto, se va a hacer uso de la API de Twitter para acceder a la información de ciertos tweets y usuarios. Existen varias librerías disponibles para una gran variedad de lenguajes, sin embargo, se ha escogido **Tweepy** [15]. Se trata de la librería principal de Python para el acceso a la API RESTful oficial de Twitter y permite acceder a todos los métodos de esta haciendo uso de funciones en Python. El uso de la API oficial permite extraer una gran cantidad de información que de otra forma sería muy complicado obtenerla de forma automatizada o directamente inviable. Cabe mencionar que para el uso de la API de Twitter es necesario iniciar sesión con unos tokens enlazados a tu cuenta, y de esta forma controlar un correcto uso y el tipo de acceso, pues existen dos tipos de cuenta: estándar y premium. En nuestro caso se hará uso del acceso estándar lo cual dará lugar a ciertas limitaciones de las que se hablarán en el apartado 3.3.3.

2.2. Sistemas de recomendación

A partir del siglo XXI, con la masificación de Internet a los hogares, comenzaron a surgir una gran cantidad de servicios que buscaban aprovecharse de todo este tráfico. Las tiendas, la prensa e incluso el ocio poco a poco se fueron digitalizando para tratar de ganar clientes, sin embargo, al mismo tiempo que crecía rápidamente el número de usuarios también crecía una competencia por el clic.

Llegan los teléfonos móviles inteligentes, todo el mundo es objetivo de ser un posible cliente desde

cualquier sitio y en cualquier momento y surge el sentimiento de la inmediatez. Los consumidores, que buscan un producto, cierta información o entretenerse no solo no quieren esperar, sino que tampoco harán mucho esfuerzo en encontrarlo hasta saltar a otra web. Las consultas explícitas, y las recomendaciones basadas en lo más visto o comprado comienzan a quedarse obsoletas pues requieren de la participación de un usuario cada vez más exigente.

Por lo general, este nuevo tráfico será muy efímero y para nada fiel, por lo tanto, hay que capitalizar al máximo sus visitas. Para ello, es necesario que su estancia en la web tenga la mayor duración posible, tanto para monetizar al máximo su estancia como para aumentar la probabilidad de que adquiera un posible producto. Con lo cual, dentro del marco de mantener a los usuarios sin siquiera ellos buscarlo entran los sistemas de recomendación.

A partir de los datos que se obtienen de los usuarios (contenido visto, duración, feedback directo), podemos obtener unas puntuaciones finales de cada usuario para cada ítem, por ejemplo, en el caso de que un usuario no haya interactuado nunca con un item esta puntuación será de cero. Y finalmente, con esto, obtener una matriz que relacione usuarios (filas) con ítems (columnas), como se puede observar en la figura 2.1.

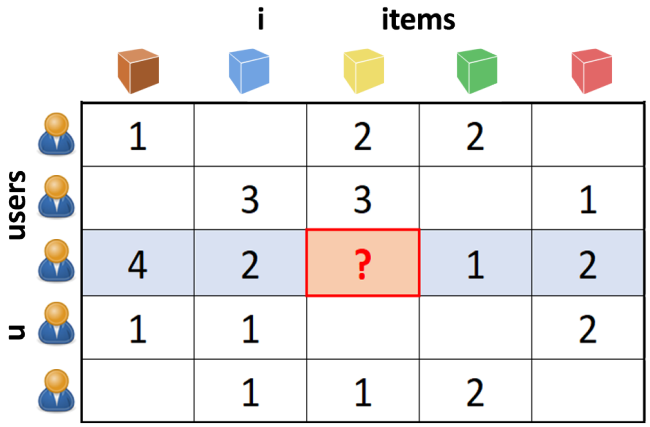

Figura 2.1: Matriz de puntuaciones de usuarios e items.

En cuanto a las tecnologías existentes actualmente, podemos encontrar una gran cantidad de librerías en diversos lenguajes que ya aportan la funcionalidad de muchas técnicas de recomendación, como por ejemplo, Librec [1] en Java, o RecBole [2] y Elliot [3] en Python. Se ha decidido usar la última librería mencionada, Elliot, principalmente gracias a su sencillez de uso que permite realizar recomendaciones sin necesidad de escribir código, pues no es una librería como tal donde hay que importar elementos y escribir código, sino que basta con completar un archivo de configuración y ejecutar una función. Permite hacer uso de una gran cantidad de recomendadores, los cuales cuentan con una gran flexibilidad para ser configurados, como también con una amplia gama de métricas que puede calcular

[1] Librec website: https://guoguibing.github.io/librec/index.html
[2] RecBole website: https://recbole.io/
[3] Elliort github: https://github.com/sisinflab/elliot

y devolver junto con los resultados y, además, al tratarse de un proyecto reciente, está actualizada.

En los siguientes apartados vamos a hablar brevemente de los distintos tipos de recomendación que existen, tanto recomendación no personalizada (random y most popular) como personalizada, así como de las distintas técnicas usadas.

2.2.1. Recomendación no personalizada

En la recomendación no personalizada, algunas de las técnicas más conocidas son las de recomendación aleatoria y de popularidad. La primera se trata, como su nombre indica, de recomendar de forma totalmente aleatoria y en ocasiones se usa para tratar de encontrar en los usuarios contenido interesante, que según sus gustos no se habría recomendado, para usarlo posteriormente en técnicas personalizadas. La segunda, se basa en ordenar los elementos en base a un indicador de popularidad con la filosofía de que "lo que gusta a mucha gente es probable que le guste".

En cuanto a la recomendación no personalizada tradicional, donde no se usa información acerca del usuario por lo que hay que trabajar con el entorno, se basa en calcular la similitud del ítem actual (el que se está observando) con la del resto de items y obtener los N elementos más similares. Como es una similitud, se va a calcular mediante la fórmula de similitud coseno que calcula el coseno de los vectores de dos ítems para obtener su similitud [16].

$$sim(i,j) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\| \cdot \|\vec{j}\|} \tag{2.1}$$

2.2.2. Recomendación personalizada

En cuanto a la recomendación personalizada, es necesario usar cierta información del usuario, para ello, necesitamos registrar algunas acciones de estos, desde información más relevante como puntaciones explícitas, pero que requieren más participación por parte del usuario, hasta información más implícita como acciones de este, así como logs con las páginas que visita o historial de compras, a partir de toda esta información se rellena la matriz de datos mencionada anteriormente y se calculan las recomendaciones haciendo uso de ella.

Esta matriz servirá como base para el cálculo de los ratings con los distintos tipos de algoritmos de recomendación existentes, que podemos dividir en principalmente en tres tipos: basada en filtrado colaborativo, donde se explotan las relaciones entre usuarios e items; basado en contenido, que como su nombre indica se usa el contenido (ya sean metadatos, textos, multimedia) de los items y de esta forma se extraen similitudes; híbrido, que como su nombre índica, se mezclan los dos métodos anteriores usando distintas estrategias para así obtener las mejores recomendaciones. Sin embargo, en este

trabajo nos vamos a centrar en los métodos de filtrado colaborativo, que se encuentran explicados en las secciones 2.2.3 y 2.2.4.

2.2.3. Filtrado colaborativo basado en vecinos

El filtrado colaborativo usa exclusivamente la matriz de usuarios-items, ver figura 2.1, e ignora el contenido de los elementos. Podemos encontrar dos tipos basados en la misma técnica, KNN basado en usuarios y basado en ítems.

Una de las ventajas de este tipo de recomendación es la capacidad de obtener resultados novedosos, que permiten al usuario no cerrarse en un tema, debido a que no se usa el contenido para nada, sino las interacciones de toda la comunidad de usuarios e items [17].

KNN basado en usuarios (UserKNN)

Se obtienen los N usuarios más similares (usando las filas de usuarios de la tabla) y a partir de ellos se calcula la puntuación con el ítem. Es decir, como se observa en la ecuación 2.2, la puntuación que se espera de un usuario (u) para un ítem (i) depende de la puntuación que tienen los usuarios (v) más similares a él y para ello se recorren todos ellos, esta similitud entre usuarios se calcula usando la matriz anterior, en concreto la fila. De hecho, existen varios tipos de similitudes, aunque las más conocidas son la de Pearson o coseno, está última, cuya fórmula se puede ver en la ecuación 2.1, es la elegida tanto para KNN basado en usuarios como en items.

$$r(u,i) = c \sum_{v}^{users} sim(u,v) \cdot r(v,i) \tag{2.2}$$

KNN basado en items (ItemKNN)

Igual que basado en usuarios, pero aplicado a ítems. Es decir, tal como muestra su fórmula en la ecuación 2.3, se recorren el resto de los items (j) y se calcula su similitud coseno con el ítem (i), elemento para el cual queremos obtener su rating con el usuario (u).

$$r(u,i) = c \sum_{j}^{items} sim(i,j) \cdot r(u,j) \tag{2.3}$$

2.2.4. Filtrado colaborativo basado en factorización de matrices

En una gran cantidad de recomendadores, como es el caso de los desarrollados anteriormente, se usa la idea de una matriz única formada por la relación entre usuarios e items donde se almacena su puntuación. Ahora veremos otro tipo de técnicas basadas en la reducción de la dimensionalidad a un subespacio de factores latentes conseguidos a partir de la descomposición de la matriz anterior a un producto de varias matrices, logrando una información más sencilla de operar y mejorando la escalabilidad de los sistemas de recomendación, además de adquirir propiedades implícitas de los usuarios e items.

Aunque se tienen artículos usando estos métodos desde hace 20 años [18], su gran popularidad llegó tras unos buenos resultados conseguidos en el concurso que realizó Netflix en 2006 para mejorar sus recomendaciones.

Funk Singular Value Decomposition (FunkSVD)

El método SVD estándar tiene algunos inconvenientes, como por ejemplo, no está bien definido para matrices dispersas, es decir, no da buenos resultados debido a que hay que rellenar los valores desconocidos y, puesto que suelen ser la mayoría, la matriz ya no se asemeja a la original. Entonces, en el Netflix Prize de 2006 Simon Funk desarrolló un algoritmo que se saltaba este procesamiento y consiguió la masificación de este método de factorización de matrices [19].

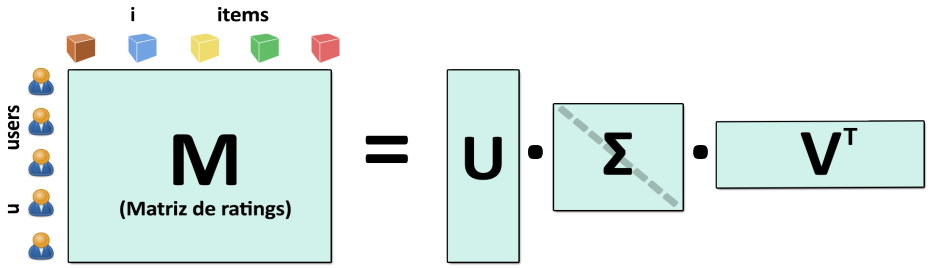

Figura 2.2: Descomposición de matrices en SVD.

En este algoritmo se realiza una factorización de matrices que produce la descomposición de la figura 2.2, donde U y V son vectores propios de características y el símbolo sigma Σ es una matriz diagonal formada por valores singulares que actúan de factores de ponderación para los dos vectores. En concreto, Funk ideó un método para aproximar esta descomposición de matrices por medio de un algoritmo incremental usando descenso por gradiente para, finalmente, ir ajustando los vectores de características minimizando el error cuadrático medio (ECM) de los ratings calculados hasta una precisión objetivo [20].

$$r(u,i) = \mu + b_u + b_i + q_i^t \cdot p_u \tag{2.4}$$

Singular Value Decomposition ++ (SVD++)

SVD++ se trata de una versión con mayor precisión que la versión estándar. Mientras que la versión de Funk tiene la limitación de solo admitir puntuaciones explícitas numéricas, esta versión permite también tener en cuenta las interacciones implícitas, como los me gusta, las compras, elementos guardados, etc.

Para ello, se basa en la inclusión de una serie de factores de items y usuarios, relacionando cada item y usuario con su vector de factores, como se puede ver en la ecuación 2.5, donde se relaciona cada item i con cada vector de factores x_j y cada usuario con cada y_a.

$$r(u,i) = \mu + b_u + b_i + q_i^t \cdot \left(p_u + \frac{1}{\sqrt{|I(u)|}} \sum_{j \in I(u)} x_j + \sum_{a \in A(u)} y_a \right) \tag{2.5}$$

Bayesian Personalized Ranking with Matrix Factorization (BPRMF)

Este algoritmo de recomendación se basa en el uso de la factorización de matrices, como su propio nombre indica, junto con la aproximación de AUC como función de pérdida, es decir, se busca maximizar la probabilidad de que un par de items aleatorios estén correctamente ordenados en el ranking. Por otro lado, el método de aprendizaje usado está basado en SGD, en otras palabras, se busca minimizar el error durante el entrenamiento mediante descenso por gradiente estocástico [21].

2.2.5. Autoencoders

La mayoría de las técnicas de recomendación personalizada tienen la limitación de un correcto funcionamiento cuando existe escasez de datos, por ejemplo, en el arranque en frío, donde no hay suficiente información de los usuarios como para realizar recomendaciones de calidad. Los autoencoders son un tipo de red neuronal de aprendizaje no supervisado que ya han demostrado tener grandes capacidades para detectar características ocultas en los datos. Además, mientras que los recomendadores tradicionales solo están pensados para trabajar con datos ~~t~~extuales, aquí se pueden mover en dominios mucho más variados como audio, vídeo o imágenes y trata mucho mejor el ruido de los datos de entrada de los usuarios [22].

Variational Autoencoders for Collaborative Filtering (MultiVAE)

Los recomendadores MultiVAE hacen uso del **Variational autoencoder**, un tipo de autoencoder que mezcla las redes neuronales con distribuciones de probabilidad. Su funcionamiento se basa principalmente en el aprendizaje de patrones a partir de un entrenamiento inicial para después ser capaz de crear datos sintéticos.

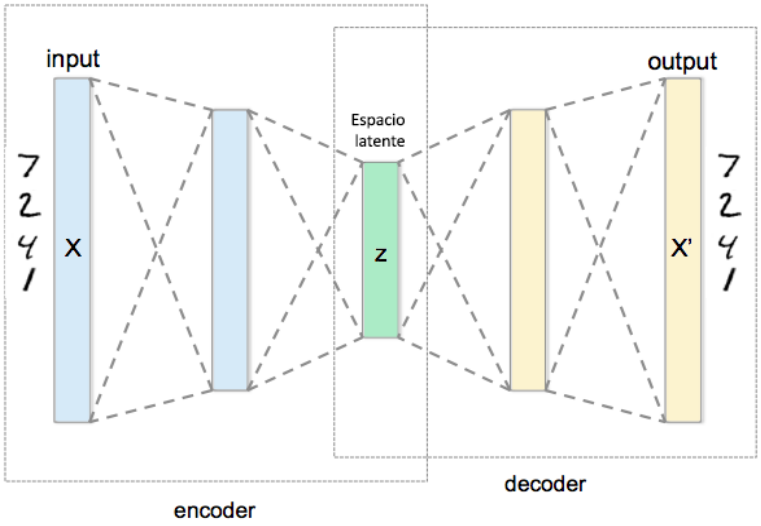

Figura 2.3: Modelo de Variational Autoencoders. Versión adaptada de una imagen sacada de Wikipedia [23]

Como se puede apreciar en la figura 2.3, el MultiVAE se divide en dos partes: una parte de codificación (encoder) donde se produce el aprendizaje de los datos y su compresión en patrones, y posteriormente, la fase de decodificación (decoder), donde, partiendo de los patrones anteriores es capaz de crear nuevas recomendaciones [24].

Durante el aprendizaje se usa todo el modelo, suele estar configurado con varias épocas y se va buscando minimizar el error de los datos al ser decodificados, es decir, la diferencia respecto a la entrada. Una vez se ha entrenado, habremos comprimido los datos en un espacio de representación intermedia, también llamado **espacio latente**, con un número inferior de variables que en los datos originales, formado por un vector de medias y de desviaciones estándar [25].

2.3. Desinformación, principios básicos y técnicas de extracción

A continuación, se va a realizar una revisión que abarque todo lo posible el tema de los elementos desinformativos, así como su definición, tipos, y técnicas de extracción centradas en la minería web que existen en la actualidad. Este último apartado no corresponde directamente con el contenido de este proyecto, aun así, es interesante analizar qué procesos se estudian o existen para su detección.

2.3.1. Desinformación, tipos e impacto

La era actual de la comunicación por internet ha dado lugar al acceso y difusión de una gran cantidad de información no siempre veraz. Además, el auge de las redes sociales como herramienta para conocer e incluso cambiar la opinión popular sobre algún tema ha convertido a la información falsa como una poderosa aliada para la manipulación. Este tipo de noticias, en función de la intención del emisor a la hora de transmitirlo, podría agruparse en dos categorías [26]:

Misinformation Se puede traducir al castellano como la falta de información o desinformación y se define como aquella información falsa que se comunica independientemente de que la intención del emisor sea de engañar al receptor.

Disinformation En este caso la intención del emisor a la hora de comunicar un mensaje será el de engañar al receptor. Así pues, es habitual que en las redes sociales se vean campañas de desinformation con el objetivo de influir en la opinión pública.

Sin embargo, en el lenguaje coloquial, se usa la traducción literal de misinformation, desinformación, como cualquier tipo de información falsa sea o no intencionada, por ello, será el significado que le daremos a esta palabra a lo largo de este trabajo. A continuación, se describirán algunos tipos de desinformación más comunes.

Fake news

A pesar de que las noticias falsas existen desde la antigüedad y han sido difundidas por los diferentes medios de cada época o personas influyentes, este término se popularizó principalmente en las campañas políticas de las elecciones de los Estados Unidos del año 2016 [1] y se ha convertido en el término común para hacer referencia a todos los tipos de desinformación. Sin embargo, esta palabra engloba a todo tipo de información creada con la finalidad de imitar una noticia real que ha sido producida por algún medio de comunicación a pesar de que difiere en su proceso o intención organizacional.

De hecho, el término *fake new* es muy amplio y cuenta con gran variedad de definiciones; por

generalizar, se van a dividir en tres tipos [27]:

Noticias malintencionadas La intencionalidad es la de transmitir o hacer pública una información que es errónea, con la idea de causar un perjuicio. Este tipo es el más frecuente, sobre todo muy común en el ámbito político y en los temas de actualidad como es el caso de la COVID-19.

Noticias humorísticas o satíricas Aquellas que se crean sin ninguna intención de causar un engaño en sus lectores y cuya finalidad es la de generar una situación cómica, pero que cuando se difunden en las redes sociales, a menudo pierden su intención original, causando que usuarios piensen que son noticias veraces [28]. Un claro ejemplo es el caso de **El Mundo Today** [4], una empresa dedicada a la creación de noticias humorísticas con tono informativo y de carácter crítico [29].

Bulos Son alertas engañosas o mentirosas que están diseñadas para influenciar principalmente de forma negativa en las personas acerca de productos, servicios, sucesos, empresas o personas; como puede ser la difusión de la muerte de una celebridad, cuando esta sigue viva. Por lo general este tipo de fake news suelen tener una comunicación en cadena y a gran escala a través de redes sociales [30].

Rumores

A lo largo del tiempo, la definición que se le ha dado en investigaciones y artículos al término rumor ha ido cambiando, por lo que encontrar un significado concreto es complicado. En este caso, haciendo uso del **Oxford English Dictionary (OED)** [5], un rumor se define como una historia o una información que circula en el presente por un medio de comunicación, que es de incierta o dudosa veracidad. Su principal diferencia respecto a los bulos es que en ningún momento son tratados como información veraz, simplemente como una posibilidad. Los rumores pueden acabar siendo verdaderos, pero no se puede saber hasta que eventualmente la información es confirmada.

Además, podemos dividir los rumores en dos categorías [31] atendiendo a sus características temporales:

Rumores efímeros Forman parte aquellos rumores que emergen durante las noticias de última hora. Este tipo de rumores son tratados durante un corto periodo de tiempo con mucha atención, por ejemplo, noticias de fichajes futbolísticos el último día del mercado de transpasos.

Rumores duraderos Son discutidos durante largos periodos de tiempo, cuya veracidad no ha sido posible contrastar desde que apareció hasta la actualidad o han acabado convirtiéndose en cuentos y leyendas populares. Este tipo de rumores provocan un interés constante

[4] Página web de El Mundo Today: https://https://www.elmundotoday.com/

[5] Oxford English Dictionary official website https://www.oed.com/

pese a no poder ser corroborada su veracidad.

Clickbait

Según OED, el término clickbait se define como contenido cuyo objetivo principal es atraer la atención y alentar a los visitantes a hacer clic en una página web en particular. Por lo general, clickbait es considerado un tipo de misinformation porque en la mayor parte de los casos, la forma de atraer a los usuarios es dando información falaz acerca del contenido en cuestión, haciéndose preguntas que no se resuelven o prometiendo contenido inexistente. El ejemplo más claro lo encontramos en algunos vídeos de la página web **Youtube** [6], donde los diversos creadores de contenido intentan llamar la atención para recibir la mayor cantidad de visitas posibles a base de títulos o portadas llamativas que se alejan de la realidad del vídeo [32].

Spam y revisiones falsas

Se considera como spam todo aquel mensaje irrelevante o no solicitado enviado a través de Internet, normalmente a un gran número de usuarios, con fines publicitarios, propagación de malware, phishing, etc. En los dos últimos casos se considera como desinformación pues la finalidad principal es la de engañar al usuario final generalmente para obtener un beneficio económico.

En el caso de las revisiones falsas, tienen una finalidad meramente publicitaria o propagan, se busca llenar un producto con opiniones positivas o negativas totalmente falsas, generalmente bots, para así influenciar en la opinión de un posible consumidor, es el caso más claro de las opiniones sobre productos en Amazon o de lugares y negocios en Google Maps. También, actuando directamente en la opinión pública para un beneficio político, por ejemplo, creando perfiles falsos tanto para aparentar popularidad como para aumentar las interacciones sobre algún tweet de opinión y así dar una sensación hacer creer que existe debate y que se hable del tema sobre una idea Twitter [33].

2.3.2. Técnicas de minería web para la extracción de desinformación

Las tareas para la detección de noticias desinformativas se han abordado desde una gran cantidad de perspectivas, principalmente se han centrado en el uso de técnicas de minería web para obtener comportamientos que las identifiquen o en la utilización de modelos de machine learning, como deep learning, para detectar esos patrones automáticamente, así también, una mezcla de ambas. A continuación, se repasarán todas aquellas técnicas centradas en el análisis de la información a partir de minería web, aunque, como dijimos antes, este no es el foco del trabajo realizado.

En el caso de la minería web, como se aprecia en la figura 2.4, podemos distinguir dos puntos de vista claramente diferenciados: las técnicas centradas en el uso de **Procesamiento de Lenguaje**

[6] Youtube official website https://www.youtube.com/

Natural (NLP) y las basadas en **Análisis de Redes Sociales (SNA)**. A su vez, podemos dividirlas en dos clases en función de la información que precisan, centradas en contenido y en el contexto:

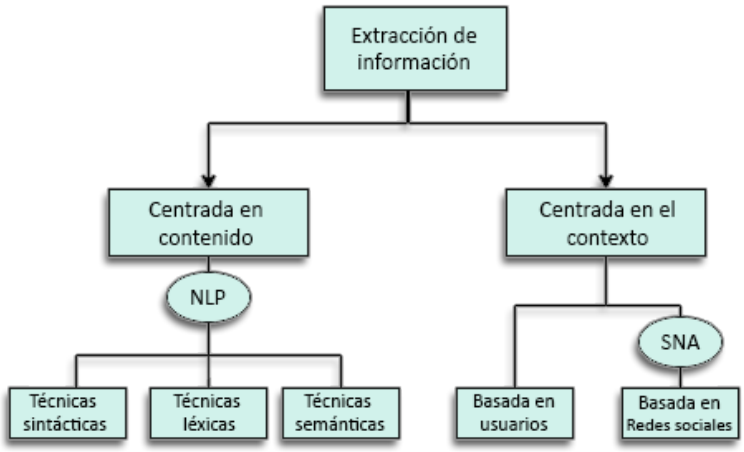

Figura 2.4: Tipos de técnicas de extracción de desinformación basadas en minería web más comunes. Los elementos circulares indican la rama de estudio que se usa.

Centradas en contenido

Se basa en la extracción de información a partir, única y exclusivamente, del uso de la información a evaluar. Es aquí donde participan las técnicas de NLP para extraer toda la información lingüística relevante del texto y existe un amplio catálogo de librerías para ello, tales como Stanford Core NLP [7] en Java, o NLTK [8] y Polyglot [9] en Python.

Recientes estudios relacionados en la comunicación e interacción en redes sociales han sido capaces de encontrar ciertos comportamientos lingüísticos que suceden en textos desinformativos, tales como el uso de palabras negativas o insultos, el engaño y las referencias hacia trabajos propios [34,35].

Centradas en el contexto

Se basa en la extracción de información a partir de todo lo que rodea al mensaje, sin analizar de por sí este, mediante su comportamiento en redes sociales y mediante el estudio de los usuarios y de su conducta. Por lo tanto, aquí entran las técnicas centradas en SNA y Basada en usuarios.

Estas técnicas nos permite estudiar cómo se comportan las informaciones, falsas o verdaderas, en un entorno que simula una situación real, como son las redes sociales, conocer a los individuos que participan en su divulgación y cómo reaccionan a estas [36], por ejemplo, si sabemos que un cierto usuario suele estar predispuesto a compartir desinformación, entonces su círculo de contactos

[7] Stanford Core NLP website: `stanfordnlp.github.io`

[8] Natural Language Toolkit website: `https://www.nltk.org/`

[9] Polyglot website: `https://polyglot.readthedocs.io/en/latest/Installation.html`

estrechos probablemente también lo esté. Sin embargo, tienen algunos problemas, en relación con la creación de perfiles, a menudo existe una dificultad para obtener información de estos usuarios debido a la anonimidad de las redes y la protección de datos.

2.3.3. Verificadores de noticias

Con el objetivo de frenar esta ola de desinformación llegan las llamadas *agencias verificadoras* o *factcheckers* en inglés. Estas agencias se dedican a luchar contra las noticias falsas creando respuestas a bulos donde comprueban su veracidad y la argumentan para que puedan ser compartidas en las redes sociales y se haga eco de que se trata de desinformación [37]. Este tipo de compañias se han hecho muy populares en los últimos años y sobre todo están centradas en el ámbito político, algunos ejemplos destacados: Newtral [10] y Maldito Bulo [11] en España, o Politifact [12] y FactCheck [13] en Estados Unidos.

El auge de este tipo de verificadores ha llegado a tener la suficiente relevancia como para colaborar con alguna de las redes sociales más populares. Por ejemplo, actualmente Facebook colabora con Politifact o Maldito Bulo para indicar si una publicación es falsa, tanto en la plataforma de Instagram como en Facebook [38].

Sin embargo, hay que tener cuidado con el poder que se les da a estas agencias, asegurarse de que sean totalmente neutrales y de que no exista ningún conflicto de intereses, puesto que el simple hecho de verificar solo unos tipos de noticias permitiendo la desinformación, por ejemplo, en solo una parte del espectro político, también se considera una forma de desinformar a la población. Por lo tanto, es fundamental que exista una total transparencia [39].

Dado que en este trabajo no vamos a intentar detectar la desinformación, puesto que es un problema suficientemente complejo de por sí, utilizaremos sitios verificadores de noticias para extraer noticias ya verificadas y construir nuestro conjunto de datos. La página web elegida, para extraer estas noticias, es la de Politifact, debido a sus excelentes características, correcto etiquetado, tener un amplio catálogo de elementos verificados (sobre todo del ámbito político) y ser lo suficientemente popular para que la gente comparta sus publicaciones en las redes sociales.

[10] Newtral website: https://www.newtral.es/

[11] Maldito Bulo website: https://maldita.es/malditobulo/1

[12] Politifact website: https://www.politifact.com/

[13] FactCheck website: https://www.factcheck.org/

3

Sistema desarrollado

A lo largo de esta sección se detallará todo el desarrollo del trabajo, describiendo así el proceso de búsqueda y extracción de los datos, filtrado, almacenamiento y la generación final del dataset para aplicarlos en un sistema de recomendación y así poder obtener resultados.

3.1. Idea general

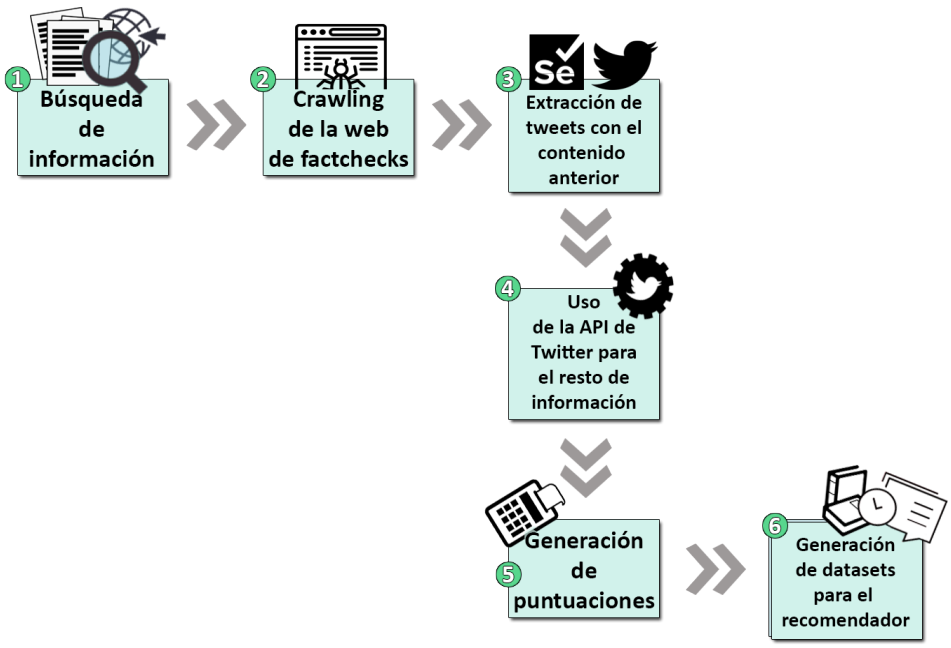

Figura 3.1: Diagrama del proceso de generación de los datos. Los marcadores numéricos indican el orden seguido para la obtención del fichero de datos final con el que realizar las recomendaciones.

En la figura 3.1 superior, gracias a los marcadores numéricos, se puede visualizar el procedimiento seguido para generar el conjunto de datos final, obtenido tras finalizar el quinto paso el cual se usará para el estudio de desinformación en recomendadores. Cabe mencionar que prácticamente en cada etapa del proceso se generará un fichero, sin embargo, se trata de conjuntos de datos intermedios que

tienen como única función aportar toda la información necesaria.

Para empezar, se ha realizado una búsqueda exhaustiva de contenido desinformativo previamente verificado y clasificado, encontrando finalmente la página web de **Politifact**, una web verificadora de bulos y virales. En segundo lugar, una vez tenemos la web, se hace uso de un crawler para extraer de ella una gran cantidad de contenido desinformativo correctamente etiquetado. Posteriormente, se utiliza el conjunto de datos generado anteriormente para encontrar bulos en Twitter, es decir, se buscarán tweets que contengan algún enlace de Politifact para encontrar usuarios verificadores, y así, más adelante, usar la API de Twitter para extraer más información, como el tweet a quien responde, el cual consideraremos como un bulo (debido a que ha sido respondido con una verificación).

Una vez hemos extraído toda la información requerida, formada por tweets desinformativos e informativos, vamos a pasarlo por un proceso de filtrado a partir del cual conseguiremos la base de datos final del trabajo. Esta base de datos está formada por entradas de usuarios con información adicional como la puntuación que se le asocia (bueno, malo o neutral) así como el enlace del tweet por el que se le dio dicha puntuación, entre otras cosas. Finalmente, usaremos toda la información anterior para recomendar y evaluar en base a distintas situaciones simuladas.

3.2. Implementación

3.2.1. Extracción de información

Obtención de material desinformativo

Tal y como se comentó en la sección 2.3.3 existen un amplio catálogo de webs para extraer contenido potencialmente desinformativo ya etiquetado. Como se discutió en esa sección, Politifact es la web elegida por sus características en cuanto a diseño de web y amplio contenido. Por ello, vamos a hacer uso del framework de Python Scrapy para extraer todo el contenido necesario y así crear una base de datos de la que partir para comenzar este proyecto.

Como se puede ver en la figura 3.2, la web de Politifact está formada por entradas donde se exponen en cada una de ellas una noticia o viral, el autor, la procedencia (televisión, Twitter, Facebook) y su clasificación como verdadera o falsa en función de unas puntuaciones, además, se añade información adicional como la explicación de por qué es verdadera o falsa con enlaces de las fuentes. Estas entradas se encuentran correctamente categorizadas y la web dispone de una gran variedad de temas.

El programa desarrollado usando Scrapy se dedicará a recorrer toda la página web y para filtrar correctamente cada página recibida, y solo quedarse con aquellas que siguen la estructura de la figura 3.3, se hará uso de XPATH, que permite acceder a información concreta de archivos XML, y si alguno de estos campos se encuentra vacío se ignora esa página. En concreto, se van a extraer todos los

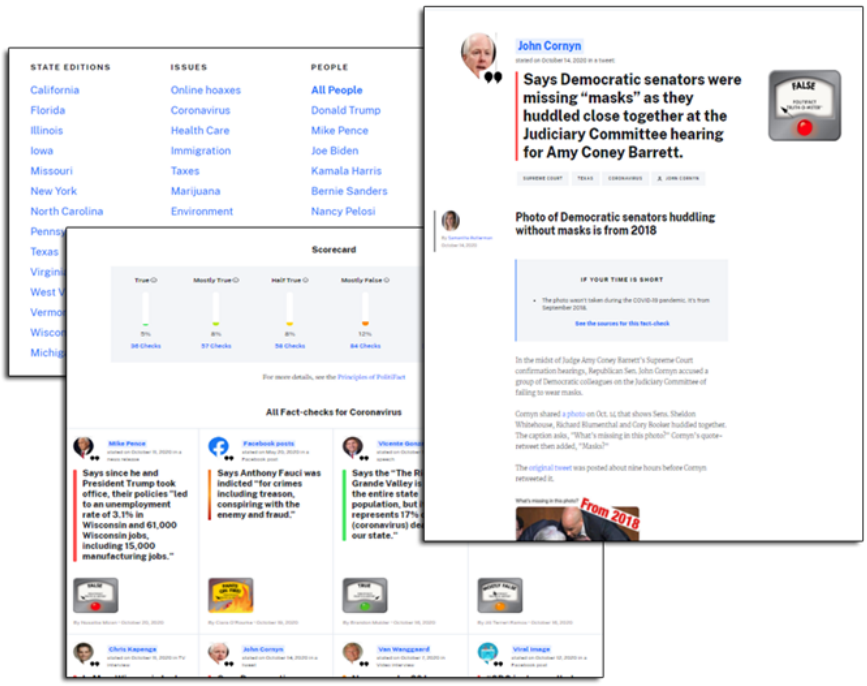

Figura 3.2: Resumen visual del portal web Politifact. Se muestran los tres tipos de páginas que tiene la web: menú de categorías, listado de noticias verificadas y página de la noticia verificada.

Figura 3.3: Elementos extraídos del portal web Politifact. Se muestra los contenidos que se van a extraer al crawler la web de Politifact.

campos que se muestran en la figura 3.3 superior, que constan de (en orden de numeración): autor, categoría, contenido a evaluar, termómetro de veracidad dado por Politifact, título de la explicación, contenido de la explicación y URLs extraídas de la explicación. Además de la información anterior, a cada entrada se le añade su URL en la base de datos.

En cuanto al termómetro de veracidad, cabe mencionar brevemente los tipos de evaluaciones que da este portal de verificación a sus noticias, y que será la misma evaluación que tendrá nuestra base de datos de misinformation. Esta web cuenta con un gran abanico de posibilidades que se dividen en dos tipos, verdaderas: true, mostly-true y half-true, y falsas: mostly-false, false y pants-on-fire.

Obtención de tweets sobre noticias

Después de obtener la base de datos con todo lo comentado anteriormente, se van a buscar tweets que comenten alguna de las entradas anteriores. Para ello, debido a limitaciones de la API oficial de Twitter relacionadas con la extracción de tweets, que ya se entrará mas en detalle en la sección 3.3.3, se hará uso de un programa partiendo de la herramienta de Selenium, usando el Web Driver del navegador Chrome, que buscará cada una de las URLs de las entradas de Politifact obtenidas y extraerá todos los tweets que las hayan usado.

Selenium [40] es una herramienta que permite realizar pruebas automatizadas de nuestras páginas web y, a partir del web driver de un navegador, es capaz de controlarlo como si fuese un usuario normal y así realizar acciones como la depuración. De momento, Selenium está disponible en Java, C#, Ruby y Python, nosotros lo hemos usado en este último lenguaje.

Debido a que esta herramienta simula el comportamiento de un usuario real buscando en la red social, la información obtenida de los tweets es limitada, pues solo se puede extraer de su HTML, en concreto se obtiene la siguiente información: nombre, apodo, fecha del tweet y su URL. Además, se trata de un proceso lento debido a la naturaleza del mismo, principalmente por el hecho de imitar un comportamiento humano y así evitar baneos, lo que provocó una duración total en todo este proceso de alrededor de 4h.

Tras esto, tenemos todos los tweets que han mencionado algún elemento de nuestra base de datos de Politifact y el siguiente paso es recopilar toda la información necesaria de Twitter.

Obtención de discusiones en twitter

Otra vez se volverá a Twitter para extraer de esta red social toda la información faltante, pero esta vez haciendo uso de la API oficial de Twitter. La lógica de realizar este proceso en dos partes se detallará más adelante en el apartado 3.3.3, aunque resumiendo brevemente, la API de Twitter impone muchas limitaciones en su versión gratuita y Selenium complica mucho la extracción de información más específica, como por ejemplo, los identificadores de usuario o los tweet en respuesta a los obtenidos. Es por eso que la combinación de ambas técnicas resulta ideal para realizar el proceso

satisfactoriamente.

Para ello, es necesario hacer uso de la librería de Python Tweepy para acceder a la API de Twitter, se recorrerán todos los IDs de los tweets extraidos con Selenium (se pueden extraer a partir de la URL) y se solicitará a la API el JSON del tweet y así tener toda la información posible que facilita oficialmente Twitter. Cabe mencionar que no nos quedaremos con toda la información, pues hay un montón de campos innecesarios en el contexto de este trabajo, tales como el número de interacciones del tweet o toda la información de la cuenta del autor del tweet, pero sí con todos los IDs de usuarios y tweets, fecha, nombre, contenido y entidades que hay dentro del mismo (menciones, hashtags y URLs), por si fuese necesario, consiguiendo con esto la base de datos de tweets final, paso 4º de la figura 3.1.

Finalmente, y como información adicional para completar nuestro conjunto de datos final más adelante y así generar mejores archivos para recomendar, se van a extraer los últimos 150 tweets (lo que se llamará **timeline**) de cada usuario recogido en el proceso anterior, es decir, independientemente de si ha sido un usuario desinformador o verificador.

3.2.2. Filtrado, extracción de características y puntuación de los tweets

En cuanto al filtrado de información, es importante recalcar que lo primero que se hizo tras obtener la base de datos de Politifact fue dividir sus elementos en dos, los evaluados como falsos por un lado y los verdaderos por otro. De esta forma, todos los pasos siguientes, como la obtención de tweets, se hará de forma aislada con cada una de las versiones y así tendremos los resultados separados y podremos evaluar si existe un comportamiento distinto en cada caso.

Al finalizar el paso 4 de la figura 3.1 obtenemos todos los tweets necesarios, sin embargo, será preciso realizar un proceso de filtrado y extracción adicional para completar la información.

Este último proceso, como se muestra en el algoritmo 3.1, se basa en recorrer cada uno de los tweets y puntuarlos, para ello, con cada tweet recogido se comprobará si contiene una URL de Politifact, en caso afirmativo será puntuado como bueno, con valor 1, además asignaremos dicha URL como **claim**. Nos referimos a claim como el URL de aquella noticia verificada, varios tweets distintos pueden hacer referencia al mismo claim. En otro caso, puntuaremos el tweet como malo, con valor -1, la claim asignada será nula y, en el caso de tener el tweet hijo, la claim del hijo será asignada como **claim_in_replay**. Como excepción, si no hay tweet hijo y tampoco hay URL entonces se puntuará como neutro, con valor 0.

Una vez hemos terminado con un tweet, se verifica si responde a otro tweet (al que llamaremos padre), en dicho caso se analizará de forma recursiva. De esta forma, para cada uno de los tweets obtendremos un elemento con la siguiente información: ID del usuario, ID del tweet, ID del tweet que contenía la claim, ID del tweet que ha respondido (si lo hubiera), claim, claim_in_replay. La lógica de

Código 3.1: Código simplificado en python para explicar la lógica de obtención del fichero de puntuaciones.

```python
def generateUsersRateJSON(dataset, black_list):
    rated_list = []
    for tweet in dataset:
        if tweet["id"] not in black_list:
            rated_list.append(checkTweet(tweet, None, None))

    saveJSON(rated_list)


def checkTweet(tweet, claim_in_replay, tweet_in_replay):
    rate_list = []
    rate,claim = getRate(tweet) # Extraemos la puntuación y el claim

    if rate == -1:
        # Se juzga el tweet por la id del tweet hijo del que provienes
        tweet_id_rated = tweet_in_replay
    else:
        # En otro caso, se le juzga por el mismo
        tweet_id_rated = tweet["id"]

    rate_list.append({
        "user": tweet["user"],
        "rate": rate,
        "tweet_id_rated": tweet_id_rated,
        "tweet_id": tweet["id"],
        "tweet_in_reply": tweet_in_reply,
        "claim": claim,
        "claim_in_reply": claim_in_reply
        })

    # Comprobamos si tiene un tweet padre de otro usuario
    if haveFather(tweet):
        father_tw = getFatherTweet(tweet)
        # Analizamos el padre recurisivamente
        rate_list = add(checkTweet(father_tw, claim, tweet["id"]))

    return rate_list
```

esta idea es que un tweet que conteste con un enlace de Politifact está tratando de verificar o corregir al tweet a quien responde, considerando a la respuesta como un tweet “bueno” y al respondido como “malo”.

Los tweets correspondientes a ambas versiones, la de elementos de Politifact verdaderos y falsos, suelen ser respuestas a otros tweets que contienen un bulo, y por ello le responden con una corrección, pero se diferencian en que aquellos que responden con un elemento evaluado como falso se trata de una respuesta hacia alguien que ha realizado un bulo mintiendo, mientras, cuando la respuesta se trata de un elemento evaluado como verdadero, se dirige hacia alguien que ha mentido también, por tanto, desinforma, pero esta vez por decir que algo es falso cuando no lo es.

Sin embargo, existe alguna excepción, si se trata de un hilo (serie de tweets de un mismo autor), no revisaremos recursivamente el tweet padre. Esto se debe a que el autor es el mismo, escribió un hilo de tweets acabando con uno de Politifact, es decir, todos los elementos del hilo son "buenos", por lo que si aplicamos el mismo proceso de revisión del tweet de arriba le estaríamos dando erróneamente una puntación negativa por el mismo contexto.

Tras ello, obtendremos el conjunto de datos final del trabajo, previo a los que se van a usar para el sistema de recomendación, que serán versiones aleatorizadas de este. En concreto tendremos tres conjuntos de datos finales, los dos provenientes de elementos verdaderos y falsos de Politifact y uno nuevo formado por los tweets de los timelines de todos los usuarios, en este último caso, y salvo algunas excepciones, estos tweets serán puntuados como neutrales, pues no estarán relacionados con Politifact, pero servirán para añadir ruido y tener una recomendación más realista.

3.2.3. Ficheros usados para el almacenamiento

Todos los ficheros de almacenamiento se han realizado bajo el formato de texto JSON.

Fichero de Politifact

Este fichero está formado por entradas del portal web Politifact dedicadas a la verificación de noticias donde indican el contenido a verificar, su procedencia, el autor, y el resultado de la verificación. El objetivo es el de tener material que esté relacionado con la desinformación para posteriormente relacionarlo con Twitter.

En el texto formateado 3.2, se puede ver el formato que tendrá cada item de este fichero, a continuación, se explicarán cada uno de los campos:

path Ruta de la noticia.

host Página web, en este caso solo será la de Politifact.

statement_quote Cita o declaración la cual se quiere verificar su veracidad.

Código 3.2: Muestra de un elemento del fichero de datos de Politifact.

```
{
  "path": "/ruta/a/la/pagina",
  "host": "www.politifact.com",
  "statement_quote": "Cita a verificar",
  "statement_author": "Autor de la cita",
  "statement_date": "Fecha de la cita",
  "statement_category": "Categoria de la cita",
  "title": "Titulo de la argumentación",
  "content": "Contenido argumentando...",
  "content_urls": [
    "URL1",
    "URL2",
    "URL3"
  ],
  "rate": "Puntuación"
}
```

statement_author Autor de la cita anterior.

statement_category Categoría de la cita anterior, nos permite saber si viene de Facebook, Twitter, de la televisión, etc.

title y content Contienen todo el texto donde se argumenta por qué se le dio dicha puntuación a la cita.

content_urls Es una lista con las URLs que hay dentro del contenido, suelen ser enlaces que se usan para referenciar datos.

Fichero de Twitter usando Selenium

Este es el fichero resultante de recorrer Twitter usando Selenium y extraer elementos mediante direcciones XPATH. Como no es viable extraer de aquí toda la información necesaria solo necesitaremos la URL del tweet. Sin embargo, como se puede ver en el texto formateado 3.3, donde se muestra el formato de cada elemento de este fichero, se ha añadido algún otro campo más, tales como el nombre de usuario o fecha de publicación, que no será usado.

El fichero actúa como paso intermedio para recoger toda la información requerida.

Código 3.3: Muestra de un elemento del fichero de datos de Twitter usando.

```
{
  "user": "Donna",
  "handle": "@Donna96725876",
  "postdate": "2021-01-31T20:56:17.000Z",
  "url": "https://twitter.com/Donna96725876/status/1355983263710408706"
}
```

Fichero de Twitter usando la API

Todos los elementos extraídos de la API serán introducidos en este fichero, por lo que sus elementos tendrán el mismo formato que los que aporta Twitter, con algunas diferencias como ciertos campos que se ignoran por no resultar interesantes o algunas variaciones.

Código 3.4: Muestra de un elemento del fichero de datos de Twitter usando su API oficial.

```
{
  "id_str": "1358274594365448193",
  "user_id": "85273142",
  "user_screen_name": "cap10d71",
  "text": "@kimmyann1111 https://t.co/1JJ7tfHGzg",
  "created_at": {...},
  "retweet_count": 0,
  "favorite_count": 0,
  "in_reply_to_status_id_str": "1358271156126552064",
  "in_reply_to_user_id_str": "2655912266",
  "entities": {
    "hashtags": [],
    "user_mentions": [...],
    "urls": [...]
  }
}
```

Este fichero, que se puede visualizar en el texto formateado 3.4, se usa para almacenar toda la información necesaria para posteriormente generar el conjunto de datos final.

Fichero de puntuaciones

El fichero de puntuaciones se trata de la base de datos final del proyecto, contiene toda la información necesaria para comenzar a generar datasets para la recomendación. Usa todos los datos de Twitter para generar elementos como el que muestra el código 3.5.

Código 3.5: Muestra de un elemento del fichero final de puntuaciones, previo al proceso de recomendación y fuente de donde generar los datos de prueba para ella.

```
{
    "user": "8953122",
    "rate": 1,
    "tweet_id_rated": "1357741528022618112",
    "tweet_id": "1357741528022618112",
    "tweet_in_reply": null,
    "claim": "https://www.politifact.com/factchecks/2021/feb/04/
       facebook-posts/joe-biden-has-not-canceled-student-loan-debt/",
    "claim_in_reply": null
}
```

Cada uno de los campos de cada elemento del fichero son los siguientes:

user Id del usuario que escribió el tweet.

rate Puntuación que se le da al tweet, puede ser bueno (0), malo (-1) o neutro (0). La explicación a esta lógica de puntuación se puede encontrar en la sección 3.2.2.

tweed_id_rated Id del tweet por el que se le dio dicha puntuación. No tiene porque ser el tweet del usuario, sino que su puntuación puede venir dada por otro tweet. De hecho, es lo que ocurre en tweets malos, pues su puntuación viene de una respuesta que le verifica.

tweet_id Id del tweet puntuado.

tweet_in_replay Id del tweet a quien responde, si lo hubiera.

claim Enlace de Politifact que contiene dicho tweet.

claim_in_replay Si ha sido puntuado por otro tweet, es el enlace de Politifact que contiene ese tweet.

3.3. Limitaciones y retos abordados

3.3.1. Limitaciones del PC

Durante la extracción de los datos, como los ficheros de puntuaciones o los resultantes de extraer tweets de los usuarios, debido a su tamaño el ordenador utilizado no es capaz de guardar todo su contenido en RAM, por lo tanto, es inviable cargarlo de golpe como se hace de forma tradicional. También existe la opción de usar un f.read convencional, sin embargo, solo permite cargar archivos sin parsear su contenido, y al tratarse de archivos JSON, obstaculizaría demasiado su manejo (al no poder controlar, por ejemplo, que se lee una entidad de JSON completa cada vez). Para ello, se han buscado algunas librerías que fuesen capaces de parsear poco a poco un JSON. Podemos destacar algunas como **json-stream-parser** [1] o **ijson** [2], aunque finalmente se ha usado esta última por ser una librería más sólida. Simplemente haciendo uso de ijson.items podemos recorrer item por item el archivo y así evitando el problema de RAM. También hay que realizar la creación de los ficheros poco a poco, según se van obteniendo los datos, para liberar espacio en RAM.

3.3.2. Pérdida de información

Si analizamos de los archivos de puntuaciones, fichero 3.5, los obtenidos de tweets con enlaces de Politifact, es decir, ignorando los neutrales (que corresponde con los timelines de los usuarios), nos encontramos que hay ciertos elementos donde no es capaz de rellenar la etiqueta **tweet_id_rated**, dicho de otro modo, el tweet por el que fue puntuado como bueno o malo, dando lugar a tuplas con dicha etiqueta adquiriendo valor nulo. Esto no debería ser posible en los dos archivos mencionados anteriormente, pues se recorren tweets que sabemos que están relacionados con Politifact. No obstante, se trata de un problema que sucede en ocasiones y se debe a cómo Twitter almacena la información. Más concretamente, durante el proceso se solicita a la API tweets que previamente sabemos que contienen una URL de Politifact, sin embargo, la información devuelta en el apartado de URLs no siempre es correcta, Twitter acorta la URL a conveniencia y a veces se pierde el rastro hacia la web de original, incluso si se intenta deshacer el proceso.

Como ejemplo, en la figura 3.4 se puede observar uno de los tweets con el problema y un fragmento de su JSON devuelto por la API que contiene el apartado de URLs, donde se encuentra un enlace de Politifact, pero que Twitter modifica por un enlace propio que hace imposible recuperarlo, con lo cual, como el programa no es capaz de detectar la URL de Politifact, su parejo en el archivo de puntuaciones tendrá puntuación negativa, dejando las etiquetas **tweet_id_rated** y **claim** nulas. Esto es algo que le ocurre a 345 elementos de 1131, si observamos los tweets con enlaces de Politifact evaluados como verdaderos y a 2522 de 12289, de los evaluados como falsos. Es decir, al 30.5 % y 20.5 % de los tweets respectivamente. Después de analizar el problema, existen tres soluciones posibles:

[1] Json-stream-parser proyect website: https://pypi.org/project/json-stream-parser/
[2] Ijson proyect website: https://pypi.org/project/json-stream-parser/

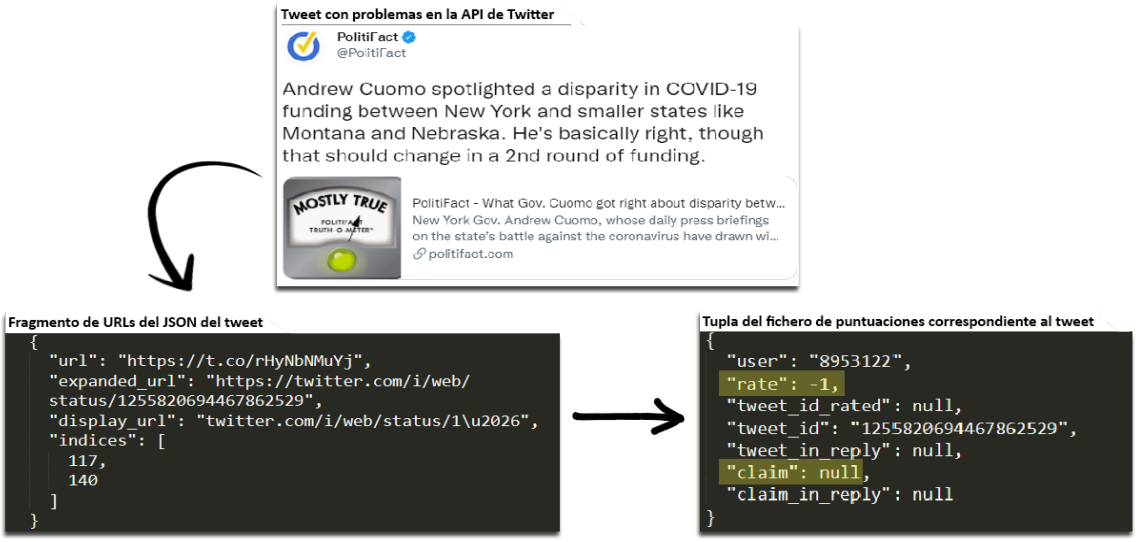

Figura 3.4: Ejemplo de tweet con problemas en la API de Twitter. Se muestra el Twitter (arriba), sus URLs en el JSON devuelto por la API (izquierda) y su tupla en el fichero de puntuaciones (derecha).

1.– Debido a que no existe forma de extraer el enlace original de Politifact, lo que se puede hacer en el programa que genera este fichero de usuarios es puntuarles como neutro.

2.– Otra opción para conseguir rellenar estos elementos de forma correcta es que una vez tengamos los tweets incorrectos, arreglarlos manualmente, sin embargo, es un proceso tedioso pues son más de 2500 elementos.

3.– Finalmente, la opción elegida es la de eliminar esas tuplas, por considerarlas como ruido. Es decir, son tweets que deberían tener puntuación positiva, pues tienen una URL de Politifact, pero como Twitter no las aporta correctamente no puede saberse con seguridad.

3.3.3. Limitaciones con la API de Twitter

En primer lugar, para poder extraer información de una red social, lo más lógico es hacer uso de la API oficial de esa red social y ceñirte a sus limitaciones. Sin embargo, como Twitter se trata de una plataforma bastante abierta se puede intentar saltar estas limitaciones usando otros métodos. Las limitaciones de la API de Twitter en su versión gratuita permiten la extracción de tweets de hasta 7 días de antigüedad, limitado a 100 tweets por petición y un máximo de 180 peticiones cada 15 minutos. Por ello, cuando se ha querido extraer todos los tweets que mencionan aquellos enlaces de Politifact recogidos con el crawler es necesario saltarse la limitación de antigüedad, es en este momento donde se usa la herramienta de Selenium y así poder cargar más tweets. De todas formas, sigue siendo necesario usar Tweepy para extraer toda la información necesaria de los tweets anteriores y es imposible saltarse la limitación de las peticiones. Esto da lugar a tiempos de ejecución elevados a la hora de extraer tweets, lo que genera un problema sobre todo al obtener el fichero de los timelines

de todos los usuarios, pues, ignorando el tiempo de espera de 15 minutos que pone twitter tras 180 peticiones, si queremos extraer todo el timeline de cada usuario, sabiendo que el máximo permitido son los últimos 2446 tweets, daría lugar a los siguientes números (dependiendo del fichero escogido):

- Para el fichero de puntuaciones de tweets con enlaces de Politifact verdaderos hay 618 usuarios, de los que se extraerían 2446 tweets por usuario, tardando unos 1-2 minutos. Esto hace un tiempo total de 618 usuarios x 2 min = **2 horas** y obteniendo finalmente **1.511.628 tweets**.
- Para el fichero puntuaciones de tweets con enlaces de Politifact falsos hay 6180 usuarios. Del mismo modo, tenemos un tiempo total de 6180 usuarios x 2 min = **9.45 días**, obteniendo finalmente **16.657.260 tweets**.

Es decir, recogiendo todo el timeline tendríamos una gran cantidad de datos, pero el tiempo de generación es inviable, es por eso por lo que la solución encontrada más acorde es la de extraer solo un pequeño conjunto de elementos por usuarios, 150 tweets. Repitiendo el cálculo anterior tenemos un tiempo total (entre los dos ficheros) de 618+6180 usuarios de **12,38 horas** y **1.019.700 tweets**, que es una cantidad más que suficiente para realizar pruebas.

4

Experimentos

4.1. Descripción de los datos

Se va a realizar un análisis de los datos originales extraídos, esto es, el conjunto de datos obtenido del portal web de Politifact y el de los tweets relacionados con este. Toda esta información data entre enero de 2020 hasta febrero de 2021 y se trata de contenido relacionado con la actualidad de Estados Unidos.

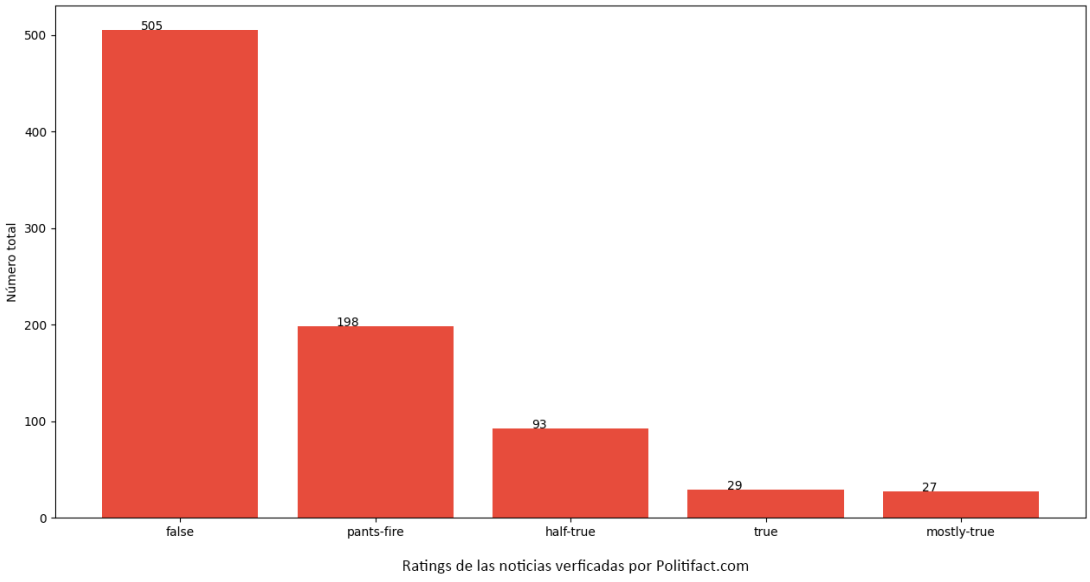

Figura 4.1: Cantidad de noticias de Politifact según su puntuación. Pants-fire y False conforman el grupo de falsas y True, Moslty-true y Half-true el grupo de verdaderas.

En concreto, se han extraído 852 elementos de Politifact, repartidos según la puntuación de veracidad que esta web le da en 703 elementos falsos y 149 verdaderos, tal como muestra la figura 4.1. A partir de estas noticias, se han extraído un total de 6207 tweets, siendo 5792 tweets relacionados con noticias puntuadas como falsas y 415 como verdaderas.

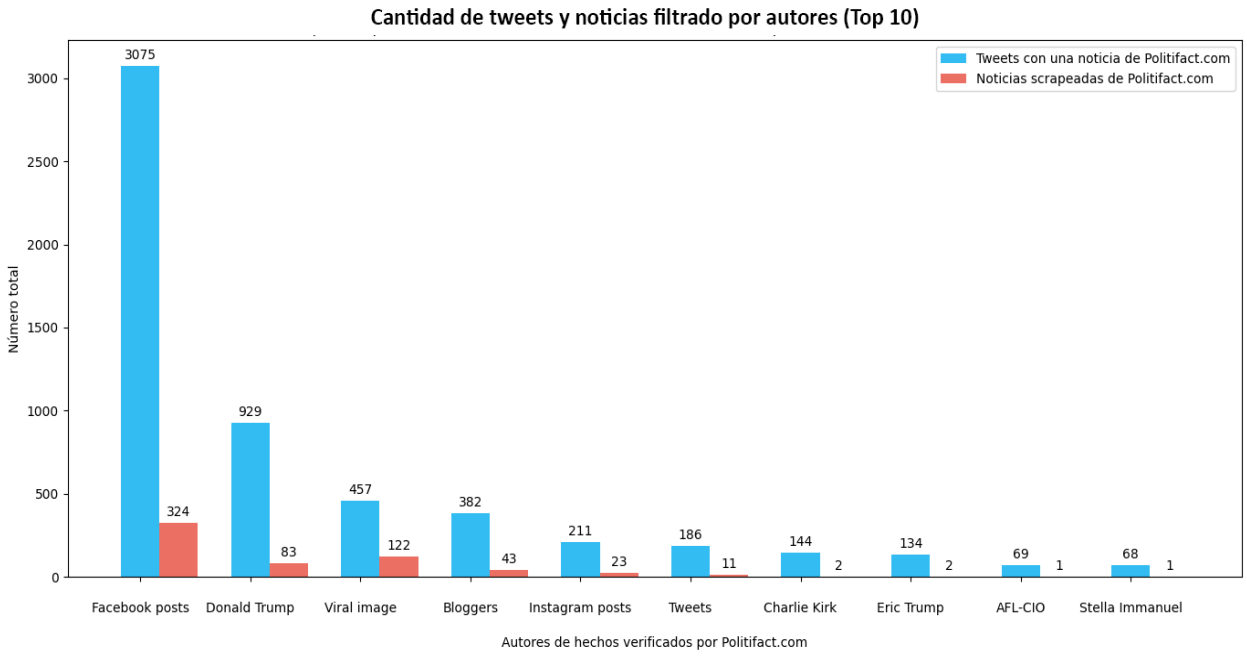

Figura 4.2: La gráfica relaciona la cantidad de tweets que comparten un enlace de Politifact con un cierto autor respecto a cuantos elementos de Politifact se tienen con ese autor.

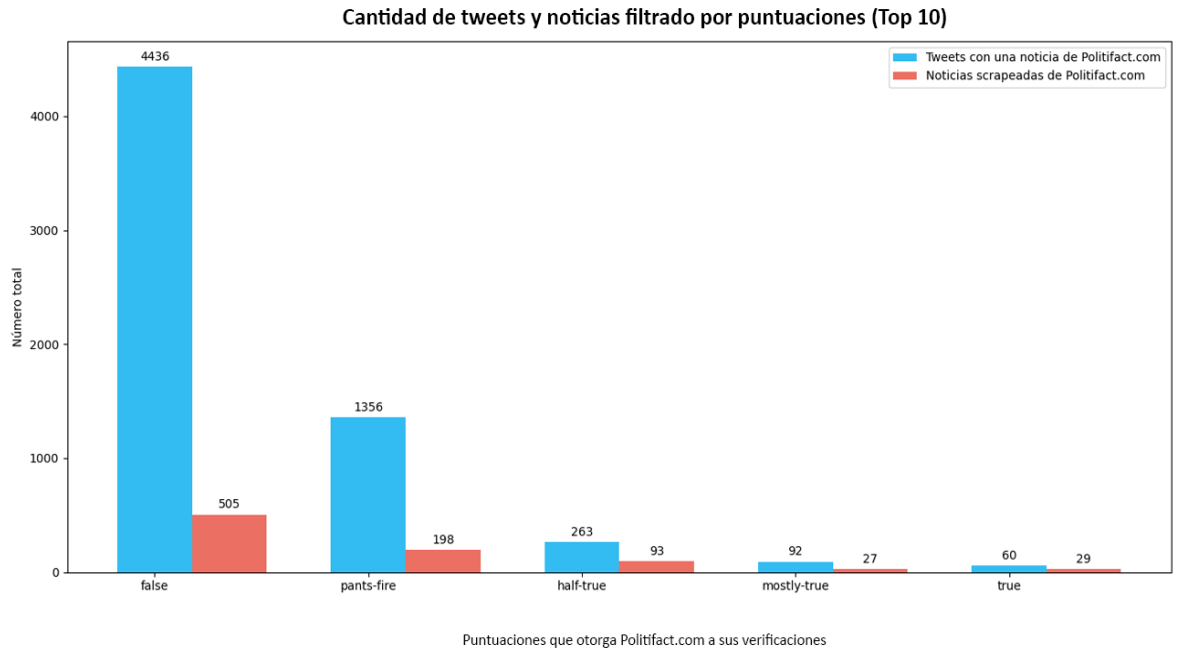

Figura 4.3: La gráfica relaciona la cantidad de tweets que comparten un enlace de Politifact con un cierta puntuación respecto a cuantos elementos de Politifact se tienen con esa puntuación.

Como se puede observar en las figuras 4.2 y 4.3, se van a relacionar la cantidad de tweets y noticias de Politifact por autores y puntuaciones, se recuerda al lector que en la sección 3.2.3 se encuentran dichas etiquetas del fichero de Politifact, donde se explican estos conceptos. Una diferencia significativa entre la cantidad de tweets que hay, respecto a la cantidad de noticias de Politifact con esas mismas etiquetas, indica un cierto grado de interés por parte de la gente. Es decir, si la cantidad de tweets es muy grande comparado con los elementos de Politifact recogidos, significa que esos elementos se están compartiendo varias veces y, por lo tanto, existe una alta respuesta hacia bulos de ese autor o tipo de puntuación.

Por ejemplo, el autor asociado a una persona pública más compartido en Twitter es Donald Trump, para el cual, se han recogido 929 tweets y 83 publicaciones de Politifact, dejando una proporción 11.2 tweets por cada noticia recogida. De esta forma, aunque se encuentre en la posición dos del ranking, supera en proporción los posts de Facebook (con 9.5 tweets/noticia) e incluso a imágenes virales (con 3.74 tweets/noticia). Recordemos que estos tweets son de usuarios que están tratando de desmentir una desinformación, lo que indica un mayor interés por desmentir bulos relacionados con un político que un post de Facebook o una imagen viral. Además, se puede observar que Facebook aparece un gran número de veces, esto se resume en que Politifact tiene una gran cantidad de noticias suyas puesto que colabora con ellos en su programa de verificadores, tal como se comentó en la sección 2.3.3 del estado del arte.

Además, es interesante mencionar que, dentro de los diez primeros autores más compartidos en Twitter, los pertenecientes a individuos están relacionados con la política, dándonos a entender el contexto temporal de cuando se publicaron siendo el año de las elecciones de Estados Unidos. Además, la única persona en el ranking que no está relacionada con la política, Stella Immanuel, corresponde con la protagonista del famoso bulo sobre la hidroxicloroquina para la cura de la COVID-19. Esto quiere decir que se pueden conocer las tendencias en un marco temporal si sabemos cuáles son los bulos más desmentidos en ese momento.

Por otra parte, existe una gran diferencia sobre cómo se comportan las noticias de Politifact en cuanto a la difusión que tienen en la red social en función de si son verdaderas o falsas. Por ejemplo, tenemos una proporción de 8.24 tweets/noticia para las falsas y 2.78 para las verdaderas, casi cuatro niveles menos de interacción. Además, tal como muestra la figura 4.4, dentro del conjunto de tweets que comparten elementos puntuados como falsos, tenemos un 62 % de los tweets que son en respuesta a otro, mientras que en los tweets con elementos verdaderos tenemos un 53 % de tweets en respuesta a otro. Esto quiere decir principalmente dos cosas:

- Cuando se desinforma, en su mayoría se hace creando una información falsa en vez de tratar como falso un hecho real. Esto se puede ver fácilmente, pues hay muchos más elementos de Politifact falsos que verdaderos.
- Normalmente, los usuarios reaccionan más ante una información inventada que hacia una

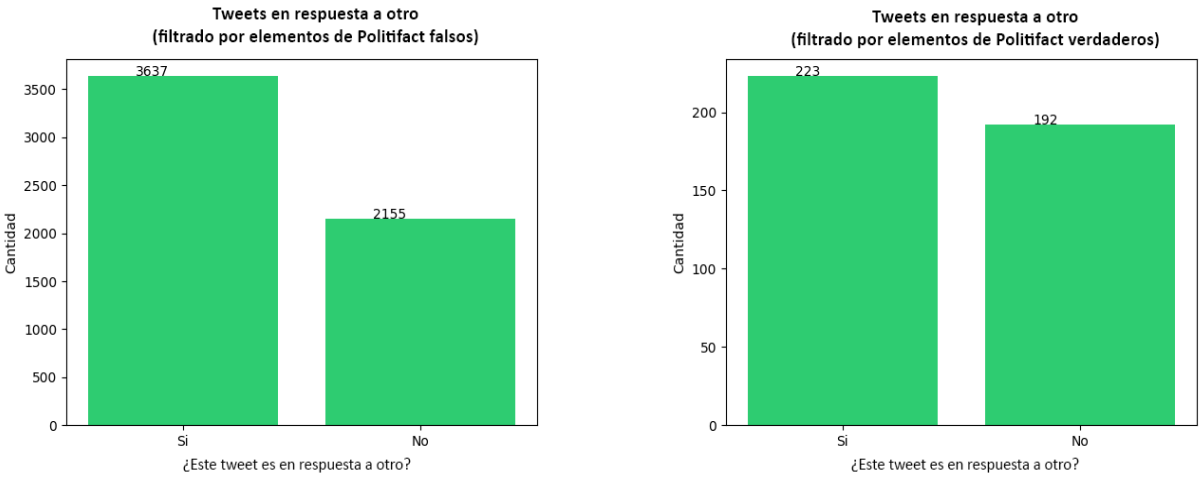

Figura 4.4: La gráfica de la izquierda hace referencia a la cantidad de tweets en respuesta a otro para tweets con elementos de Politifact puntados como falsos. La gráfica de la derecha hace referencia a la cantidad de tweets en respuesta a otro para tweets con elementos de Politifact puntados como verdaderos.

información real que ha sido tratada como falsa. Tal como muestran los promedios de tweets calculados anteriormente.

Por lo tanto, una conclusión que se podría extraer de esto es que es más fácil desinformar desmintiendo una noticia verdadera que inventando algo, puesto que existe una menor reacción de la gente a contrastar esa información. Esto se ve claramente en las gráficas 4.4, se responde mucho más cuando se trata de algo falso (gráfica izquierda), que cuando se trata de desmentir algo verdadero (gráfica derecha).

Finalmente, vamos a describir el conjunto de datos final de puntuaciones que, como ya se ha mencionado en la sección anterior, está formado por dos archivos resultantes de filtrar los datos extraídos anteriores y un archivo adicional formado por los timelines de todos los usuarios. En los primeros dos archivos se tienen un total de 8722 tweets puntuados, como se observa en la figura 4.5, el 74 % han sido puntuados como *buenos* (con 1) y el 16 % restante como *malos* (con -1).

Además, es interesante comentar la proporción de usuarios y tuplas ya que nos da una idea de como están distribuidos los elementos, que luego vamos a recomendar, con cada usuarios. Como podemos apreciar en la figura 4.6, nos encontramos con una cantidad de 8721 tuplas repartidas en 6205 usuarios, dando una proporción de 1.4 tuplas/usuarios y una mediana de 1. Es decir, prácticamente solo hay una tupla por usuarios, sin embargo, existe un reducido número que acumulan grandes cantidades, como se ve la gráfica de la distribución de la figura anterior. De hecho, los dos primeros usuarios corresponden con cuentas de Twitter que se dedican a compartir noticias de Politifact y el tercero se

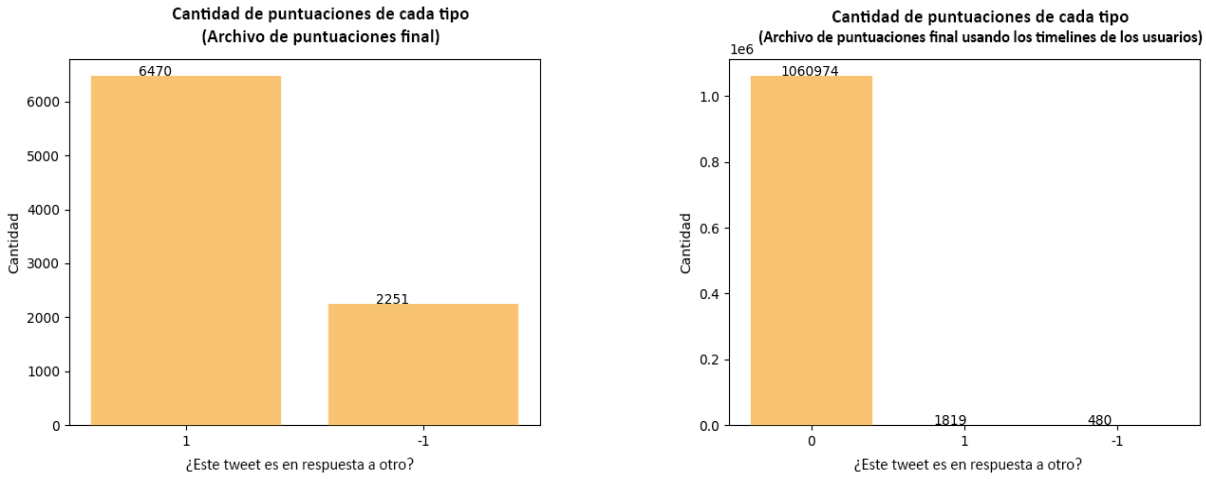

Figura 4.5: La gráfica de la izquierda hace referencia a la cantidad de puntuaciones de cada tipo que hay entre los dos primeros archivos de puntuaciones. La gráfica de la derecha hace referencia a la cantidad de puntuaciones de cada tipo que hay en los archivos de puntuaciones que usan los timelines de los usuarios.

trata de un usuario que puso un tweet falso, y como mencionamos anteriormente, dichos tweet tienen una alta cantidad de interacciones, muchas de ellas tratando de verificarle con Politifact.

Por otro lado, en el tercer fichero habrá principalmente puntuaciones neutras (con 0), pues son elementos que no tienen nada que ver con Politifact, con lo cual, no se podrá establecer una puntuación negativa o positiva. En la figura 4.5 se muestra un total de 1.063.274 tweets puntuados en un 99.7 % como neutrales, 0.17 % como *buenos* y un restante de 0.03 % negativos. Cabe mencionar que, en su mayoría, los tweets puntuados como *buenos* y *malos* están duplicados en los dos primeros ficheros, pues se están cogiendo tweets de los mismos usuarios, por lo tanto, habrá muy pocas novedades entre ellos.

Como información adicional, si consideramos como usuarios *buenos* aquellos sin un solo tweet puntuado como negativo y como *malos* viceversa, de los 6205 usuarios totales tenemos un 70 % de *buenos* y 30 % *malos*, tal como se ve en la figura 4.7.

En cuanto a las claims (recordemos que con esto nos referimos al enlace de Politifact usado en un tweet) se han analizado para conocer cuáles son las diez más compartidas (populares) en Twitter, ver figura 4.8. En dicha figura se muestran mapeadas con un identificador, para poder conocer a qué elemento exacto de Politifact hacen referencia, como también su puntuación asociada o la temática, puede ir a la tabla 4.1. Resumiendo el contenido de esta tabla, se puede observar que tenemos un pleno de claims falsas, dato que concuerda con la información que se mencionaba anteriormente relacionada con una mayor reacción de los usuarios hacia noticias inventadas que hacia noticias reales

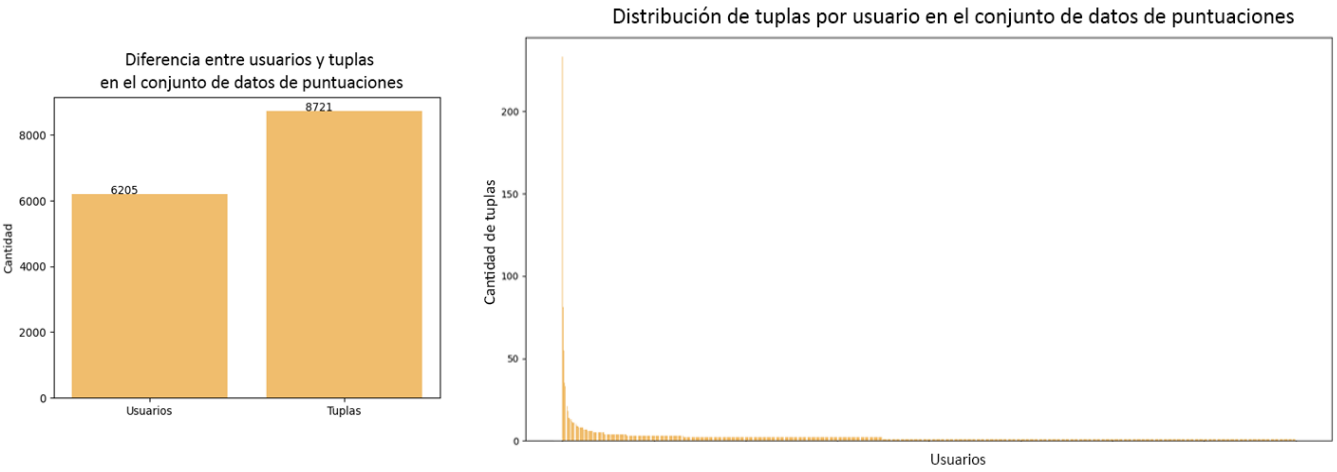

Figura 4.6: Distribución de usuarios y tuplas en el conjunto de datos de puntuaciones. La gráfica de la izquierda muestra la cantidad de usuarios con respecto a las tuplas y la gráfica de la derecha muestra (sin entrar en detalle) la distribución que sigue el reparto de tuplas por usuarios.

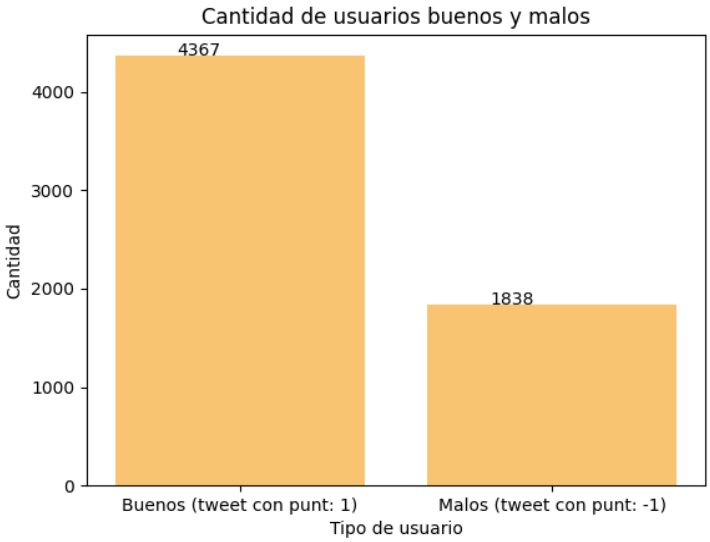

Figura 4.7: La gráfica muestra la cantidad de usuarios *buenos* y *malos* hay entre todos los archivos de puntuaciones.

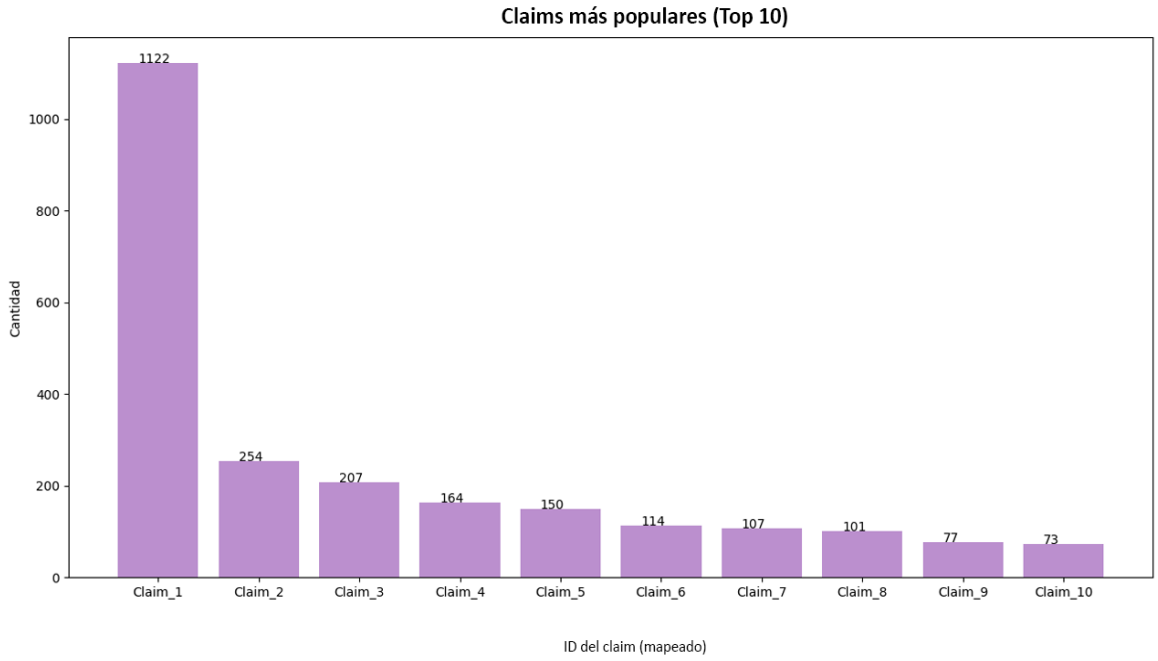

Figura 4.8: La gráfica muestra las diez claims más populares, es decir, las que han sido más compartidas entre los tweets extraídos.

tratadas como falsas. Además, los bulos más populares están directamente relacionados con los hechos más relevantes del año (fraude inventado en las elecciones de EE. UU., culpas entre políticos por COVID-19, etc.).

4.2. Metodología de evaluación

4.2.1. Generación de ficheros de prueba

Para poder realizar los experimentos con los distintos sistemas de recomendación se necesita generar archivos adicionales que cumplan ciertas características, como la de controlar la proporción de usuarios y tweets buenos o malos, además de establecer los datos en el formato adecuado para que puedan ser correctamente procesados por los recomendadores. Con ese fin, se ha hecho un programa que permita automatizar todo este proceso. A partir de la base de datos final con las puntuaciones, se generan archivos TSV de forma aleatoria configurables con ciertos parámetros, y así, aunque la generación es aleatoria, poder comprobar si se cumplen ciertos comportamientos bajo los mismos criterios.

Estos archivos TSV estarán formados por tripletas con el ID del usuario, item y puntuación. En concreto, el item será un identificador del claim previamente mapeado, además, como estos se repiten,

ID	Claim	Puntuación	Tema
Claim_1	How President Barack Obama handled the outbreak of H1N1	False	Trump y COVID-19
Claim_2	Mask box label is legitimate, but people are misinterpreting it	False	COVID-19
Claim_3	The Obama administration "didn't do anything about" swine flu.	False	Trump y COVID-19
Claim_4	Says Nancy Pelosi was "caught trying to..."	False	COVID-19
Claim_5	"Wisconsin has more votes than people who..."	Pants in fire	Fraude electoral
Claim_6	"Nancy Pelosi may have just committed a violation of 18 U.S.C. § 2071..."	Pants in fire	Política
Claim_7	"We inherited a broken test" for COVID-19.	Pants in fire	Trump y COVID-19
Claim_8	Early morning election results from Michigan and Wisconsin show voter fraud.	False	Fraude electoral
Claim_9	Says photo of beaten woman is Aracely Henriquez...	False	George Floydd
Claim_10	Voters in Maricopa County, Ariz., were forced to...	False	Fraude electoral

Tabla 4.1: Esta es una tabla que muestra las claims mapeadas de la figura además de información adicional.

se reduce la dispersión a la diferencia de si se utilizara directamente el ID del tweet, ya que el número de estos es mucho mayor. Sin embargo, también podemos añadir neutralidad usando los tweets de los timelines con puntuación neutra, donde el item tomará el ID de dicho tweet (ya que en estos casos no hay claim).

En la herramienta, cuyo algoritmo se puede ver en el código 4.1, se podrán configurar principalmente dos parámetros, el primero de ellos y el más fundamental es el de la **proporción de la desinformación**, se podrá establecer la proporción deseada de elementos negativos (desinformación) y positivos, de hecho, esa proporción podrá estar calculada según todas las tuplas de los datos (usando como parámetro el código "ALL") o conforme a los usuarios (usando como parámetro el código "USER"), es decir, considerando un usuario como malo si contiene algún tweet negativo.

El otro parámetro es la **proporción de neutralidad**, para elegir qué porcentaje de tweets neutros usar de los timelines. Es decir, como se ha mencionado anteriormente, se han extraído alrededor de 150 tweets neutrales de cada usuario, por lo tanto, un 25 % de neutralidad implica utilizar unos 37 tweets por usuario dando un total de 229.585 elementos para el fichero generado.

Además, para poder diferenciar los claims en función de cómo intervinieron en el tweet, se podrán etiquetar dependiendo de si participaron para verificar un tweet desinformativo, etiqueta negativa con __b, o si formaban parte de este, etiqueta positiva con __g, y así aportar información al sistema de qué relación tienen con el tweet, como se puede ver en la figura 4.9. Es decir, actúan dos usuarios: el primero quien desinforma y el segundo que responde con el claim, ambos tendrían el mismo item asociado pero uno con puntuación negativa y el otro con positiva; al añadir estas etiquetas diferenciamos los claims en función de si el usuario propagaba desinformación (negativo) o la verificaba (positivo).

A la hora de devolver las recomendaciones, tener etiquetados los claims nos permite saber qué cantidad de desinformación ha sido recomendada a un usuario, pues se considerará los items con

Código 4.1: Código simplificado en python para explicar la lógica de obtención de los ficheros de prueba para la recomendación.

```
def generateRecDataset(dataset_rates_list, prop_rate, type_prop_rate, prop_user,
      distin_items_by_rate):
  # Obtenemos una lista con IDs de elementos elegidos con la proporción de puntuación deseada
  if type_prop_rate == "USER":
    selected_items = getPropRateByUser(dataset_rates_list, prop_target)
  elif type_prop_rate == "ALL":
    selected_items = getPropRateByTweet(dataset_rates_list, prop_target)

  for dataset_rates in dataset_rates_list[:2]:
    addItem(TSV_list, rate_elem, distin_items_by_rate)

  # Obtemos los neutrales en función del prop_user. Si es 0 = no neutralidad.
  neutral_items = getTimelineItemsByUser(dataset_rates_list[-1], prop_user)
  for rate_elem in neutral_items:
    addItem(TSV_list, rate_elem, distin_items_by_rate)

  saveTSV(TSV_list)


def addItem(TSV_list, rate_elem, distin_items_by_rate):
  for rate_elem in dataset_rates:
    # Solo cogemos los seleccionados tras calcular la proproción
    if getID(rate_elem) in selected_items:

      user = getUser(rate_elem)
      # Obtenemos el ID único de la claim, con una etiqueta si lo deseamos
      item = getClaimID(rate_elem, distin_items_by_rate)
      rate = getRate(rate_elem)

      if distin_items_by_rate is True: # Añadir las etiquetas de desinformación
        if rate = 1:
          item = item + "__g" # El claim viene de un elemento que verifica
        else:
          item = item + "__b" # El claim viene de un elemento que desinforma

      # Suavizamos el rate, tal como se explica al final de la sección 4.2.1,
      # e introducimos la tupla en la lista
      TSV_list.append(user, item, smoothedRate(rate))
```

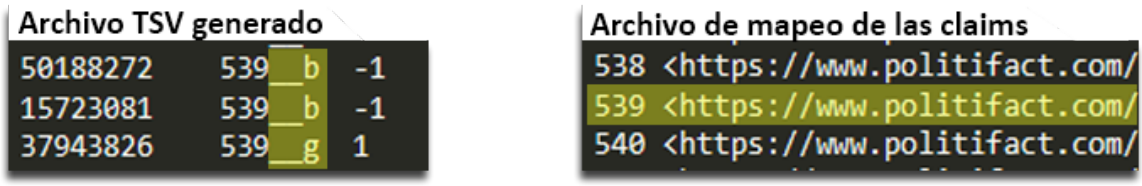

Figura 4.9: Izquierda, ejemplo del fichero TSV generado casi listo para usarlo en el sistema de recomendación (falta suavizarlo) con las etiquetas de positivo, negativo habilitadas. Derecha, el fichero de mapeo de las claims a identificadores.

etiqueta negativa (__b) como desinformación.

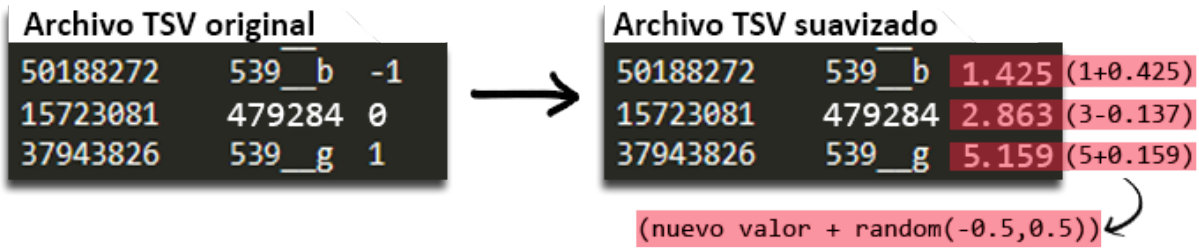

Figura 4.10: Izquierda, ejemplo de un fichero TSV original. Derecha, el fichero TSV tras el proceso de suavizado.

Para finalizar, tal como se observa en la figura 4.10 se ha realizado un suavizado de los datos, es decir, se han cambiado las puntuaciones siguiendo un sencillo algoritmo. Las puntuaciones asociadas a desinformación (-1) son sustituidas por 1, a su vez, aquellas que eran 1 por 5, y por último, las neutrales (0) se cambian por 3. Además, a esto se le añade un pequeño ruido aleatorio que modificará entre los -0.5 y 0.5 los valores anteriores y permite darle variedad a los datos. Este proceso se realiza principalmente para evitar tener puntuaciones negativas puesto que no todas las técnicas de recomendación admiten este tipo de valores, además de conseguir mejorar las recomendaciones, por ejemplo, reduciendo la cantidad de resultados nulos (igual a 0).

4.2.2. Descripción de los datos generados

Una vez conocemos el proceso de generación de los datos, se han creado varios ficheros, listos para ser usados en la recomendación, los cuales podemos dividir en dos clases: calculando la proporción de desinformación por usuarios y por tuplas. Ambas clases tendrán la misma configuración, que irán desde un 10 % de desinformación hasta un 90 % con saltos de 10 en 10. Para cada porcentaje de desinformación tendremos 3 conjuntos de datos, con 0 % de neutralidad, con 25 % de neutralidad y

con 50 %, es decir, para cada tipo de proporción tendremos un total de 27 ficheros, sumando ambos tipos obtenemos un global de 52 ficheros sobre los que generar recomendaciones.

Un problema asociado a los ficheros con neutralidad (25 % o 50 %) es su enorme cantidad de items diferentes, pues dicha neutralidad aporta items únicos entre ellos (tweets sin claims). Esta cantidad enorme de items diferentes provoca que a la hora de recomendar y desarrollar la matriz de usuarios-items algunas técnicas de recomendación requieran de mucha memoria RAM, por lo que ha sido necesario hacer uso de un ordenador externo que supla este requisito y así poder generar las recomendaciones. No obstante, incluso con estas condiciones (el ordenador externo tiene 128GB de RAM y 16 procesadores) ha resultado imposible obtener algunas recomendaciones debido a los requisitos de la librería utilizada.

4.2.3. Descripción del proceso de recomendación

Tal como se comentó en la sección 4.2.1 del estado de arte, se ha hecho uso de la librería Elliot para realizar varias pruebas con algunos sistemas de recomendación sobre los datos explicados en el apartado anterior. Para ello, es necesario crear un fichero de configuración donde se indican los datos de entrada, la ruta de los resultados, y los sistemas de recomendación a emplear con sus parámetros.

```
experiment:
  path_output_rec_result: ../results/{0}/recs/
  path_output_rec_weight: ../results/{0}/weights/
  path_output_rec_performance: ../results/{0}/performance/
  dataset: rates
  data_config:
    strategy: fixed
    train_path: ./path/to/file.tsv
    test_path: ./path/to/file.tsv
  models:
    Random:
      meta:
        verbose: True
        save_recs: True
      seed: 42

    MostPop:
      meta:
        save_recs: True

    ItemKNN:
      meta:
```

Figura 4.11: Ejemplo del archivo de configuración de Elliot. Las zonas remarcadas indican cada uno de las secciones configurables del fichero.

Para facilitar la recomendación se ha creado un programa que automatiza todo el proceso e incluso permite la paralelización del mismo. Simplemente recibe la ruta de una carpeta con todos los conjuntos

de datos a recomendar, que se irán leyendo uno a uno, y la ruta del fichero YAML de configuración de Elliot, que será modificado con cada fichero de información leído para adaptarlo a sus necesidades. Por ejemplo, como se muestra en la figura 4.11, subrayado en amarillo se encuentra la ruta deseada donde generar los distintos resultados para el recomendador, como los pesos calculados, el rendimiento o las recomendaciones resultantes, campos que no se verán modificados. En azul, donde se indica el nombre del dataset, corresponde con el nombre que Elliot usa para crear la carpeta donde se almacenarán los resultados anteriores, este nombre variará para tener una carpeta única con cada ejecución, es decir, con cada fichero de datos.

Después, subrayado en verde, se encuentran las etiquetas correspondientes a la ruta de los ficheros de entrenamiento y test deseados para usar en el proceso. En el caso de este trabajo no nos interesa la parte de testeo, pues solo vamos a estudiar las recomendaciones generadas en entrenamiento, sin embargo, tal como está programado Elliot exige de un fichero de test, el cual debe contener los mismos usuarios del entrenamiento pero con items o puntuaciones diferentes. Para ello, el programa de automatizado se encarga de generar un fichero *espejo* aleatorio al del entrenamiento simplemente para saltarnos esta restricción de la librería. Las rutas de entrenamiento (train_path) y test (test_path) se verán modificadas con cada fichero a recomendar.

Por último, en rojo se encuentra toda la configuración referida a los sistemas de recomendación que se van a usar y se mantendrá constante para todos los ficheros. Se trata de una lista donde se pueden añadir todos los modelos deseados y, para cada uno de estos modelos, se generará un fichero de recomendaciones resultante, que se analizará más adelante en el apartado 4.3. En concreto, los sistemas de recomendación utilizados y sus configuraciones son los siguientes (si se desea conocer su funcionamiento, se puede encontrar su explicación en la sección 2.2 del estado del arte):

Random Es un método de recomendación no personalizada basado en la aleatorización.

MostPop Como su nombre indica hace referencia a “lo más popular”, es una técnica de recomendación no personalizada que sirve para analizar cuáles son los items más usados y, si a medida que se añada desinformación, ésta acaba convirtiéndose en lo más popular.

ItemKNN KNN basado en items con similitud coseno, esta técnica generará tres tipos de recomendaciones: para 20, 50 y 100 vecinos.

UserKNN KNN basado en usuarios con similitud coseno, esta técnica generará tres tipos de recomendaciones: para 20, 50 y 100 vecinos.

BPRMF Bayesian Personalized Ranking with Matrix Factorization dedicando 10 épocas para el entrenamiento, 10 factores latentes y un ratio de aprendizaje de 0.001. Los coeficientes de regularización se pondrán a 0.0025 excepto el del bias que será nulo.

FunkSVD Singular Value Decomposition siguiendo el algoritmo de Simon Funk, se utilizan 10 épocas para el entrenamiento, 10 factores latentes y un ratio de aprendizaje de 0.001. En cuanto a los coeficientes de regularización, el de los factores latentes se pone en 0.1 y el

del bias en 0.001

SVDpp Para esta versión de SVD se sigue la misma configuración que el modelo anterior, la única diferencia es que se establecen 50 factores latentes (en vez de 10).

MultiVAE El modelo Variational Autoencoders for Collaborative Filtering es un modelo basado en redes neuronales, se configurará con una dimensión de 600 elementos en el espacio intermedio y de 200 en el espacio latente, un ratio de aprendizaje de 0.001 y un coeficiente de regularización de 0.01. También, se pondrá la probabilidad de abandono en 1.

4.2.4. Métricas seleccionadas

Como ya se ha mencionado, la finalidad de este proyecto es la de conocer si los sistemas de recomendación, que se encuentran en todos los servicios que usamos, participan en la difusión de elementos desinformativos. Por lo tanto, para cuantificar el impacto que tiene cada una de las técnicas de recomendación se van a hacer uso de algunas métricas novedosas, algunas de ellas propuestas recientemente [41].

Recuento de desinformación (RD) Para cada usuario, mide la cantidad de elementos desinformativos que es recomendada. Es necesario establecer un punto de corte para que todos los elementos estén normalizados bajo la misma cantidad de recomendaciones. Si no establecemos un corte, se calculará el RD como la desinformación entre la cantidad de elementos, devolviendo un resultado en el rango de [0,1].

Diferencia de ratio de desinformación (DRD) Se encarga de calcular cuánto ha variado la cantidad de recomendación con desinformación con respecto a los datos iniciales. Para ello, calculamos la proporción de desinformación al inicio (llamado **mprop i**) y obtendremos su diferencia respecto de la proporción de desinformación recomendada (llamado **mprop r**), obteniendo **mprop i - mprop r**. Un valor negativo implica que ha empeorado, es decir, le han recomendado al usuario más desinformación de la que tenía en un inicio. Por otro lado, si el valor es positivo, implica una mejora. Si recogemos el resultado en valor absoluto indica cuánto ha variado la cantidad de desinformación.

Contador de recomendaciones (CR) Métrica que calcula cuántos usuarios han recibido recomendaciones no nulas, es decir, el sistema de recomendación ha sido capaz de devolverles items con puntuaciones distintas de 0. Junto con el número total de usuarios, permite saber el porcentaje de población a la que se es capaz de recomendar (% CR). Esto es importante ya que en algunos momentos del desarrollo (sobre todo, cuando no se había incluido el suavizado) algunos algoritmos eran capaces de recomendar a muy pocos usuarios.

Contador de recomendaciones promedio (CRP) Métrica que calcula el número de reco-

mendaciones no nulas recibido por cada usuario para luego hacer el promedio total. Nos permite conocer cuántos items promedio es capaz de devolver cada sistema de recomendación a un usuario.

4.3. Resultados

A continuación, se encuentran los resultados de las métricas de los distintos algoritmos de recomendación expuestos en secciones anteriores. Donde, además, se podrán encontrar gráficas que exponen de una forma más visual el comportamiento de los mismos y un análisis donde se explica, junto con evidencias, qué técnicas de recomendación son más propensas a devolver desinformación.

Primero, se comenzará con un análisis de los recomendadores para todas las métricas a partir de ficheros sin neutralidad. Tras esto, un estudio más exhaustivo de cada recomendador, donde se estudia el efecto de los parámetros en la recomendación de desinformación. Por último, repetimos el análisis de los recomendadores pero añadiendo neutralidad a los datos.

4.3.1. Análisis por métricas de desinformación

Las tablas 4.2 y 4.3 presentan los resultados de las métricas para el conjunto de datos sin elementos neutrales. Como se puede apreciar, los resultados entre ambas tablas son muy similares, ya sea calculando la proporción de desinformación con todas las tuplas, como muestra la primera de ellas, o con los usuarios, como muestra la segunda. El principal motivo de esto se encuentra en la condición que tienen los datos recogidos de Twitter, es decir, en el conjunto de datos de puntuaciones prácticamente se tiene una tupla por usuario (proporción de 1.4 tuplas/usuario), como ya se mencionó en la sección 4.1, por lo tanto, al haber casi el mismo número de tuplas que de usuarios los datos van a adquirir un comportamiento similar.

A pesar de que son tablas similares, podemos encontrar algunos detalles diferenciadores, como el hecho de que algunos métodos funcionan un poco diferente con unos datos que con otros. Por ejemplo, en la tabla 4.3 los recomendadores ItemKNN y UserKNN son capaces de devolver mucha más desinformación con solo un ratio del 10 % que en la otra, así como, mientras que para la proporción basada en tuplas se obtiene un RD@20 de 1.62 y 1.52, con la otra proporción estos datos ascienden a 4.73 y 4.90, es decir, se devuelve casi cuatro veces más desinformación.

En estas tablas tenemos cuatro grupos de técnicas de recomendación: primero nos encontramos con los algoritmos no personalizados, Random y Most Popular (MostPop); después, los algoritmos de filtrado colaborativo basado en vecinos, como ItemKNN y UserKNN; técnicas de recomendación de filtrado colaborativo basadas en factorización de matrices, tales como, FunkSVD, SVDpp y BPRMF; y por último, filtrado colaborativo basado en redes neuronales, en concreto, auntoencoders como MultiVAE.

Ratio	Recm	RD@5	RD@10	RD@20	DRD	% CR	CRP@5	CRP@10	CRP@20
0.1	Random	1.44	2.10	3.67	-0.05	100 %	5	10	20
0.1	MostPop	**1.00**	**1.00**	**1.00**	**0.09**	100 %	5	10	20
0.1	ItemKNN	1.10	1.59	1.62	0.04	68.2 %	3.92	6.77	10.55
0.1	UserKNN	1.07	1.47	1.52	0.05	54.1 %	4.09	7.19	12.85
0.1	FunkSVD	1.50	2.73	5.73	-0.16	100 %	5	10	20
0.1	SVDpp	1.39	2.18	4.12	-0.08	100 %	5	10	20
0.1	BPRMF	**1.00**	1.06	1.78	0.03	100 %	5	10	20
0.1	MultiVAE	1.02	1.15	1.68	0.05	100 %	5	10	20
0.3	Random	1.78	**3.04**	5.97	0.05	100 %	5	10	20
0.3	MostPop	1.92	3.9	7.86	-0.002	100 %	5	10	20
0.3	ItemKNN	1.62	3.95	**5.64**	**0.08**	52.8 %	3.95	6.78	11.31
0.3	UserKNN	1.58	4.28	6.35	0.07	39.3 %	3.95	6.94	11.72
0.3	FunkSVD	1.80	3.28	7.59	-0.052	100 %	5	10	20
0.3	SVDpp	2.22	4.14	8.30	-0.06	100 %	5	10	20
0.3	BPRMF	**1.43**	3.82	7.84	0.03	100 %	5	10	20
0.3	MultiVAE	2.10	3.67	6.58	0.03	100 %	5	10	20
0.6	Random	2.43	4.61	9.16	0.18	100 %	5	10	20
0.6	MostPop	4.05	6.77	15.69	-0.04	100 %	5	10	20
0.6	ItemKNN	2.44	4.93	10.14	0.14	45.9 %	3.81	6.04	9.66
0.6	UserKNN	**2.02**	**4.36**	**9.12**	**0.19**	30.4 %	3.55	5.30	7.47
0.6	FunkSVD	3.40	7.05	12.57	0.03	100 %	5	10	20
0.6	SVDpp	3.26	6.28	12.23	0.03	100 %	5	10	20
0.6	BPRMF	3.84	7.32	14.49	-0.06	100 %	5	10	20
0.6	MultiVAE	3.40	6.36	11.88	0.06	100 %	5	10	20
0.9	Random	**3.58**	**7.10**	**14.18**	**0.20**	100 %	5	10	20
0.9	MostPop	5.00	9.99	19.01	-0.05	100 %	5	10	20
0.9	ItemKNN	4.54	8.33	16.14	0.10	56.1 %	3.61	6.08	9.28
0.9	UserKNN	4.41	7.92	15.38	0.14	30.8 %	3.35	5.10	5.53
0.9	FunkSVD	4.33	8.41	16.93	0.05	100 %	5	10	20
0.9	SVDpp	3.87	7.52	14.58	0.19	100 %	5	10	20
0.9	BPRMF	4.98	9.69	19.05	-0.04	100 %	5	10	20
0.9	MultiVAE	4.90	9.73	19.16	-0.04	100 %	5	10	20

Tabla 4.2: Métricas de desinformación para la configuración estándar de los algoritmos de recomendación (por ejemplo, UserKNN e ItemKNN con 50 vecinos), usando el conjunto de datos sin elementos neutrales y **la proporción de desinformación calculada según todas las tuplas**. Todas las métricas, RD, DRD CR o CRP están explicadas en la sección 4.2 y el ratio indica el porcentaje de desinformación. Los valores remarcados en negrita indican el mejor valor para esa métrica, es decir, la que indica una menor desinformación (CR y CRP no se marcan pues no son métricas ligadas a la desinformación).

Ratio	Recm	RD@5	RD@10	RD@20	DRD	% CR	CRP@5	CRP@10	CRP@20
0.1	Random	1.33	1.80	2.97	-0.04	100 %	5	10	20
0.1	MostPop	**1.00**	**1.00**	**1.00**	**0.06**	100 %	5	10	20
0.1	ItemKNN	2.34	3.84	4.73	-0.01	76.7 %	4.32	8.00	14.45
0.1	UserKNN	2.73	3.59	4.90	-0.02	60.8 %	4.23	7.965	14.81
0.1	FunkSVD	1.01	1.32	2.97	-0.05	100 %	5	10	20
0.1	SVDpp	1.47	2.15	3.95	-0.09	100 %	5	10	20
0.1	BPRMF	**1.00**	1.03	1.44	0.03	100 %	5	10	20
0.1	MultiVAE	1.03	1.07	1.39	0.04	100 %	5	10	20
0.3	Random	1.65	2.64	5.01	0.05	100 %	5	10	20
0.3	MostPop	**1.00**	2.93	5.88	0.06	100 %	5	10	20
0.3	ItemKNN	3.27	3.484	5.84	**0.08**	71.1 %	4.3	7.83	13.58
0.3	UserKNN	2.93	3.45	6.15	0.07	51.4 %	3.82	6.43	10.10
0.3	FunkSVD	1.48	3.04	6.69	-0.06	100 %	5	10	20
0.3	SVDpp	1.49	2.54	**4.76**	0.06	100 %	5	10	20
0.3	BPRMF	1.14	2.60	5.72	0.01	100 %	5	10	20
0.3	MultiVAE	1.36	**2.33**	4.94	0.05	100 %	5	10	20
0.6	Random	**1.89**	**3.33**	**6.56**	0.26	100 %	5	10	20
0.6	MostPop	3.83	6.78	11.73	-0.03	100 %	5	10	20
0.6	ItemKNN	2.59	3.78	8.24	0.21	61.7 %	3.40	5.53	8.14
0.6	UserKNN	2.40	4.81	7.52	**0.27**	44.9 %	3.17	4.96	7.76
0.6	FunkSVD	3.08	6.27	11.80	0.03	100 %	5	10	20
0.6	SVDpp	2.48	4.83	9.39	0.13	100 %	5	10	20
0.6	BPRMF	3.62	7.03	13.30	-0.05	100 %	5	10	20
0.6	MultiVAE	3.22	6.08	11.31	0.04	100 %	5	10	20
0.9	Random	**2.62**	**5.10**	**10.26**	**0.38**	100 %	5	10	20
0.9	MostPop	4.99	9.02	19.02	-0.03	100 %	5	10	20
0.9	ItemKNN	3.52	6.59	13.53	0.23	67.8 %	3.70	6.22	9.29
0.9	UserKNN	3.51	6.08	12.45	0.30	39.66 %	3.32	5.38	7.83
0.9	FunkSVD	4.35	8.76	17.56	0.02	100 %	5	10	20
0.9	SVDpp	3.36	6.57	12.38	0.28	100 %	5	10	20
0.9	BPRMF	4.87	9.07	18.64	-0.03	100 %	5	10	20
0.9	MultiVAE	4.67	9.28	18.55	-0.03	100 %	5	10	20

Tabla 4.3: Métricas de desinformación para la configuración estándar de los algoritmos de recomendación (por ejemplo, UserKNN e ItemKNN con 50 vecinos), usando el conjunto de datos sin elementos neutrales y **la proporción de desinformación calculada según los usuarios**. Todas las métricas, RD, DRD CR o CRP están explicadas en la sección 4.2 y el ratio indica el porcentaje de desinformación. Los valores remarcados en negrita indican el mejor valor para esa métrica, es decir, la que indica una menor desinformación (CR y CRP no se marcan pues no son métricas ligadas a la desinformación).

A medida que va aumentando la proporción de desinformación que metemos al sistema los datos se van *contaminando* hasta el punto de que los algoritmos de recomendación devuelven más desinformación por estar rodeado de ella y no por ser más propensos, es lo que ocurre por ejemplo a partir del ratio 0.5 que culmina con el 0.9 donde se puede observar que en la métrica RD@5 hay muy poca diferencia entre recomendadores. Sin embargo, sí que hay ciertos algoritmos cuya lógica de recomendación se ve directamente beneficiada al aumentar la desinformación, es el caso de los dos métodos no personalizados: random, a mayor cantidad de desinformación hay más probabilidad de escogerla aleatoriamente, sin embargo, se sigue manteniendo por debajo del resto de técnicas; most popular se aprovecha directamente de cómo están formados los datos, donde las claims desinformativas se suelen repetir más que las informativas, por lo tanto, a medida que aumentamos la desinformación van a aumentar mucho la aparición de dichas claims convirtiéndose en las más populares, por ello, con un ratio de 0.9 este recomendador prácticamente devuelve únicamente desinformación. Realizando un breve apunte a lo anterior, como se comentó al describir los datos en la sección 4.1, los tweets con desinformación tienen mucha más interacción que el resto, con lo cual, las claims asociadas a dichos tweets desinformativos se van a ver repetidas pues aparecerán en el fichero de puntuaciones por cada usuario que trató de verificar dicho tweet, cosa que no ocurre al revés.

En cuanto a las técnicas de filtrado colaborativo basadas en vecinos, destacamos que tienen en la mayoría de ratios la métrica DRD positiva, de hecho, si hiciésemos la media con todos los ratios serían los algoritmos con mayor valor en dicha métrica. Por lo tanto, esto indica una reducción de la desinformación respecto al conjunto de datos de entrenamiento, pudiéndose comprobar también que se tratan, por detrás del random, de los recomendadores con menor RD, es decir, que devuelven una menor cantidad de desinformación que el resto. No obstante, existen excepciones como es el caso de la tabla 4.3 para un ratio de 0.1, donde UserKNN e ItemKNN se convierten en los algoritmos con mayor desinformación.

Sin embargo, estas ventajas mencionadas solo afectan a un porcentaje de los usuarios, pues como se aprecia en la métrica CR, estos algoritmos, debido a la dispersión de la matriz de recomendación, no son capaces de encontrar vecinos y por lo tanto encontrar recomendaciones para todos ellos. De hecho, se recomienda a más individuos con el conjunto de datos con proporción de desinformación según los usuarios, tabla 4.3, donde se alcanzan diferencias incluso del 20 % respecto a la otra tabla, por ejemplo, para ratio 0.3. Además, otro inconveniente que tienen estos algoritmos es la incapacidad de devolver un número elevado de elementos a todo ese porcentaje de usuarios que sí es capaz de recomendar, como se puede ver en la métrica CRP.

En siguiente lugar, tenemos las técnicas basadas en factorización de matrices (SVDs y BPRMF), donde, mientras que los SVDs (FunkSVD y SVDpp) destacan por devolver altos niveles de desinformación cuando se trabaja con bajos ratios (<0.5), como se observa en los ratios 0.1 y 0.3, por otro lado, con altos ratios (>0.5) entran dentro del grupo de algoritmos que menos desinformación recomiendan (junto con ItemKNN e UserKNN). Por su parte, BPRMF funciona justo a la inversa, aportando

poca desinformación con ratios bajos y siendo de los algoritmos que más desinforman con ratios altos, seguramente porque tiende a ser parecido a popularidad [42].

Por último, tenemos la técnica basada en redes neuronales (MultiVAE) la cual funciona igual que BPRMF o most popular, es decir, es un buen algoritmo con bajos porcentajes de desinformación, pues es capaz de devolver menos contenido desinformativo que otros métodos, pero cuando se usa en un entorno con porcentajes mucho más altos empeora significativamente.

4.3.2. Efecto de los parámetros en los algoritmos

Para mostrar el efecto que producen distintos parámetros y configuraciones de los recomendadores en la generación de resultados con desinformación, se han creado una serie de gráficas, que se pueden ver en 4.12, 4.13, 4.14 y 4.15, donde se muestra de una forma más visual cómo evoluciona la cantidad de desinformación con distintos ratios (0.1, 0.3, 0.6 y 0.9 respectivamente). Dichas gráficas usan los mismos datos de las tablas comentadas anteriormente con neutralidad 0, en concreto, muestran la métrica RD@10, a la que se ha añadido la información con varios parámetros. Para empezar, el primer comportamiento que nos encontramos, algo bastante lógico por una parte, es el aumento de desinformación en cada gráfica a medida que se usa un mayor ratio.

Si nos centramos en las técnicas basadas en vecinos próximos, podemos destacar algunas cosas. Con ratios bajos (<0.5): UserKNN presenta un mejor comportamiento con 50 vecinos, muy seguido su valor con 20 vecinos y empeorando ($\sim$0.4 puntos) al usar 100 vecinos; ItemKNN se comporta de una forma más libre, con 0.1 de ratio va mejorando a medida que aumentan los vecinos, con una mejora importante ($\sim$0.8 puntos) en los 100, mientras, con un ratio de 0.3 se comporta totalmente a la inversa, empeorando a medida que aumentan los vecinos con una diferencia entre extremos en torno a los 0.8 puntos. Además, podemos observar en ambas gráficas (4.12, 4.13) que ItemKNN siempre devuelve mejores resultados que UserKNN cuando trabajan con el máximo de vecinos, mientras que con 20 y 50 la recomendación basada en usuarios es mejor usando un ratio de 0.1 y la basada en items usando un ratio de 0.3.

Por otra parte, estudiando el comportamiento con ratios altos (>0.5), encontramos que ambas técnicas basadas en vecinos próximos devuelven un máximo de desinformación con una configuración intermedia, 50 vecinos. Primero, ambas empeoran al pasar de 20 a 50, ItemKNN lo hace de forma más moderada que UserKNN (que llega a incrementar su desinformación en casi 3 puntos con el ratio de 0.9) y luego mejoran notablemente con 100 vecinos, alcanzando su mínimo. Otro dato interesante que muestran ambas gráficas (4.14, 4.15), es el hecho de que UserKNN tiene un comportamiento en todas las configuraciones mejor que ItemKNN, siendo dicha diferencia mayor con la configuración de 20 vecinos y una proporción de desinformación de 0.9, donde la diferencia alcanza los 2 puntos.

Una hipótesis a este comportamiento es el hecho de que cuando tenemos desinformación entre los

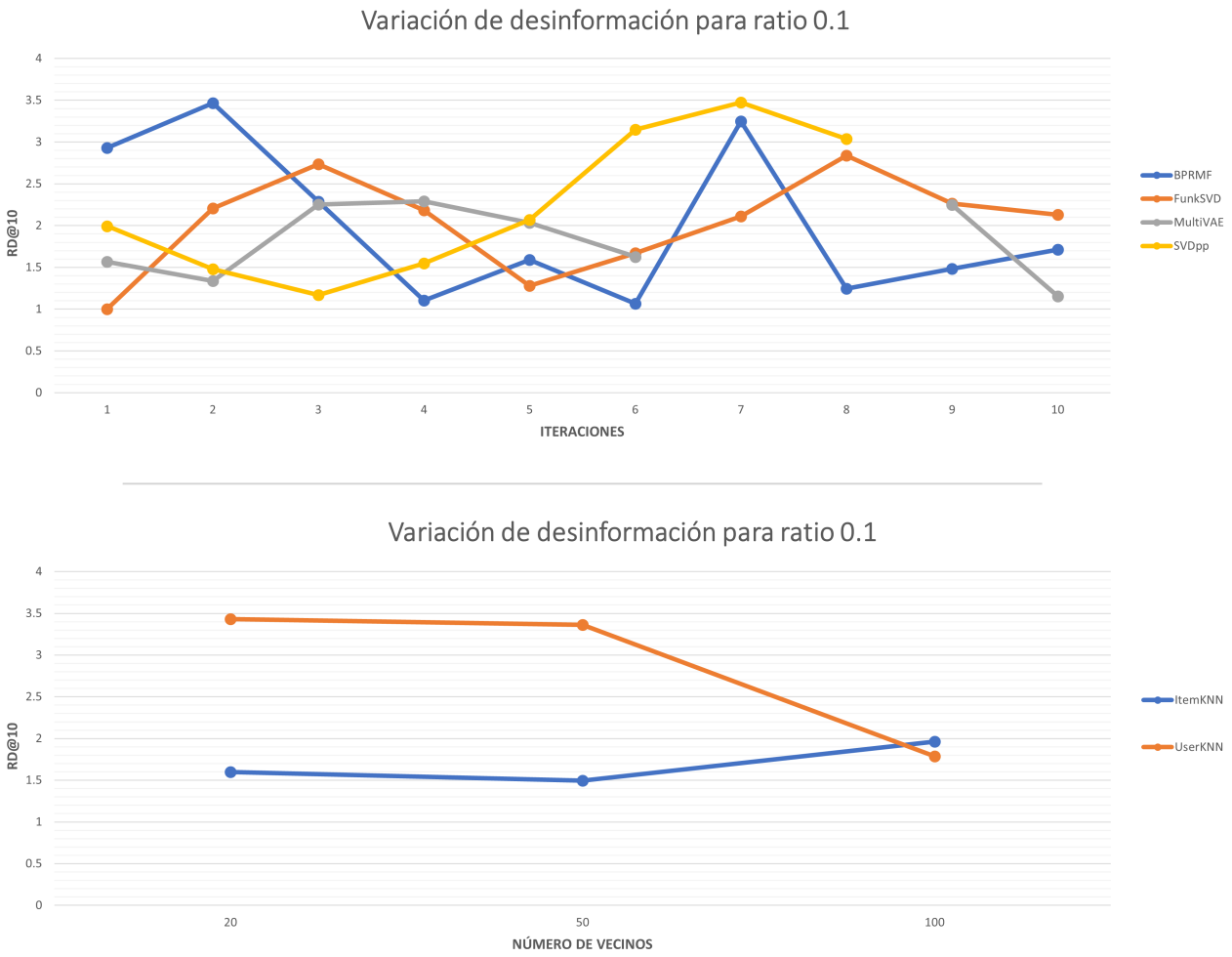

Figura 4.12: Representación visual de como varía RD@10 con los distintos parámetros de los algoritmos de recomendación, en la gráfica superior se muestran los algoritmos que usan iteraciones y en la inferior los que usan vecinos próximos, ambas gráficas para una proporción de desinformación de 0.1 y sin elementos neutrales. Recordemos que un mayor valor implica más cantidad de desinformación devuelta por el recomendador, por lo que tener un valor pequeño indica un mejor comportamiento.

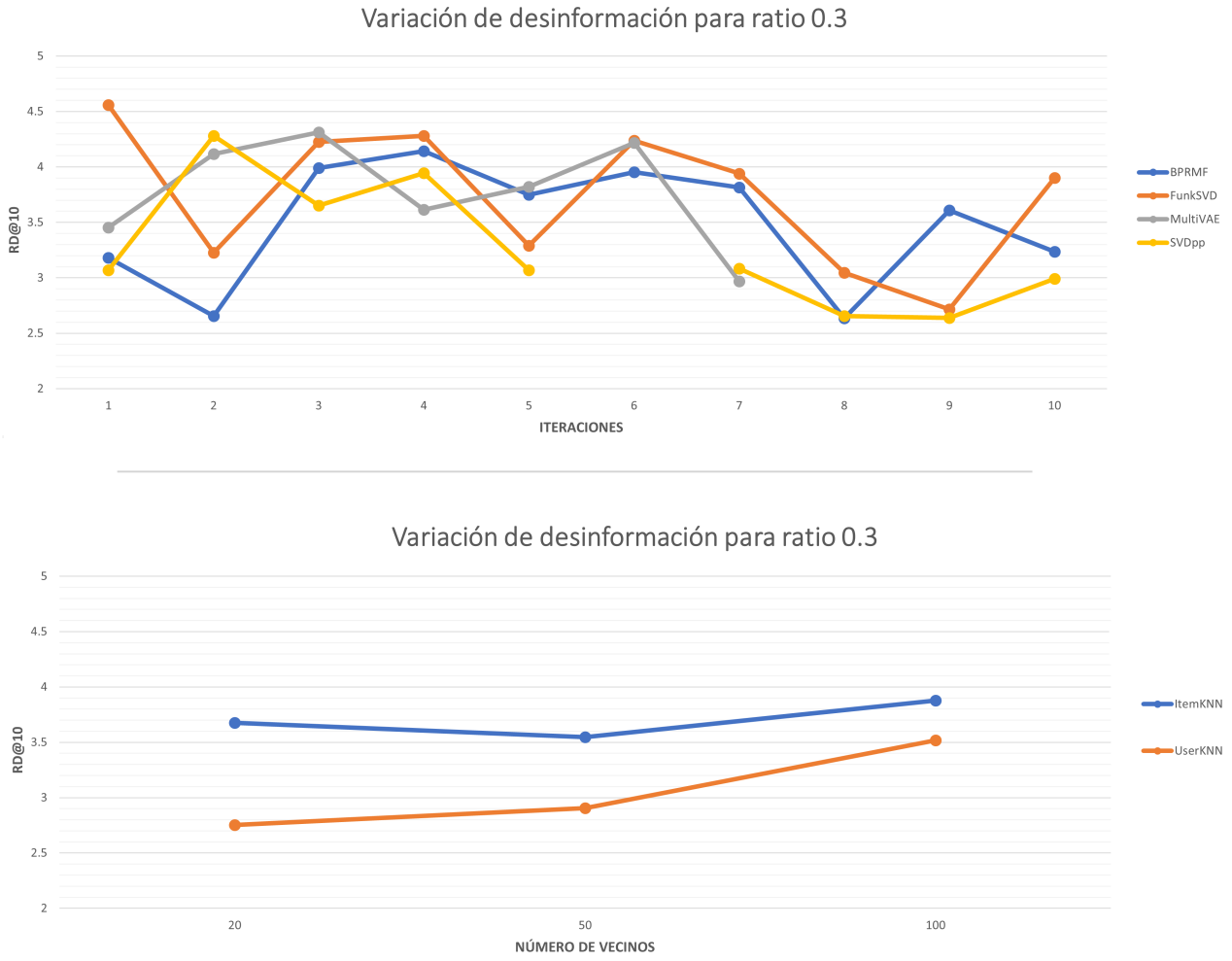

Figura 4.13: Misma información que la figura 4.12 pero con ratio 0.3

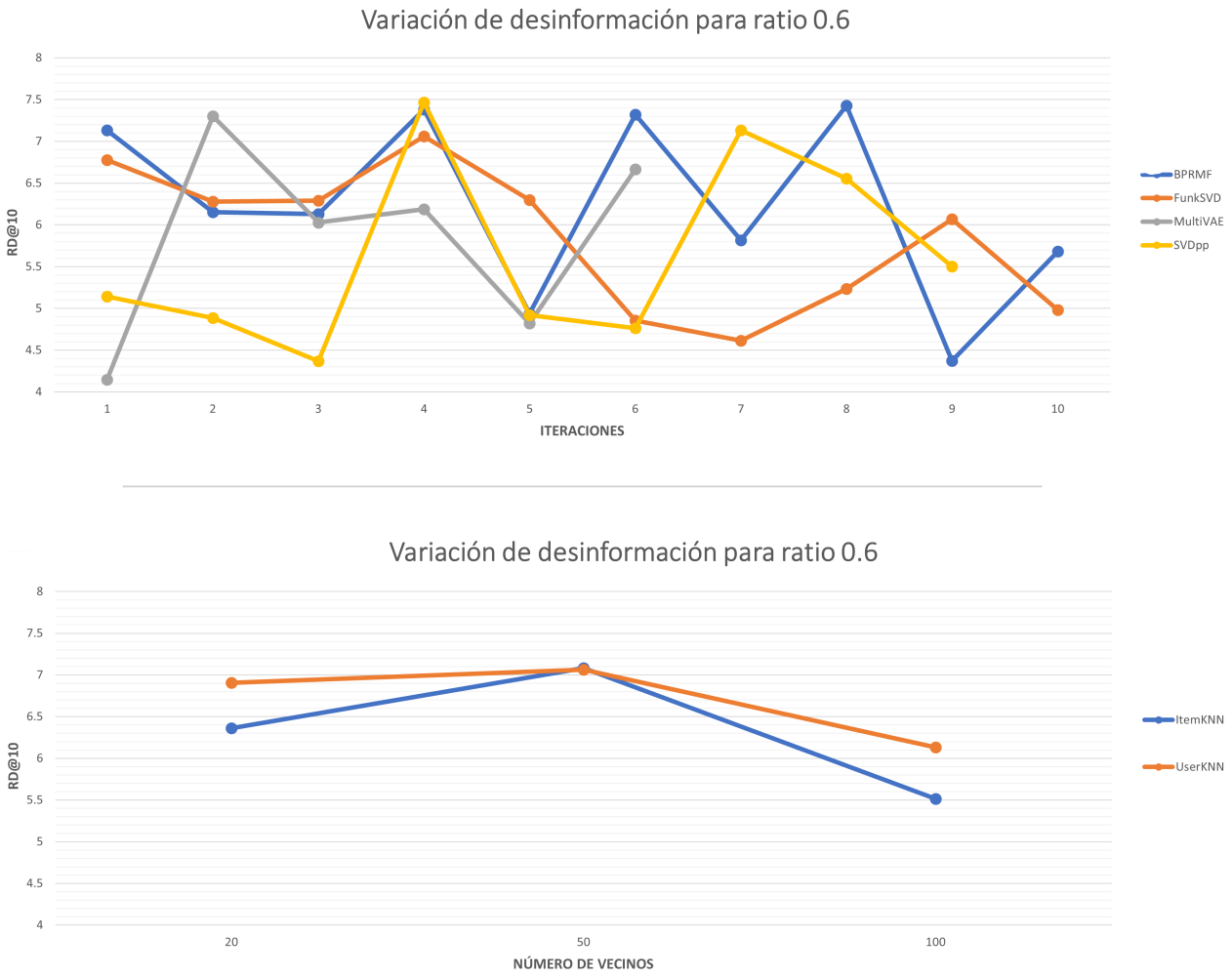

Figura 4.14: Misma información que la figura 4.12 pero con ratio 0.6

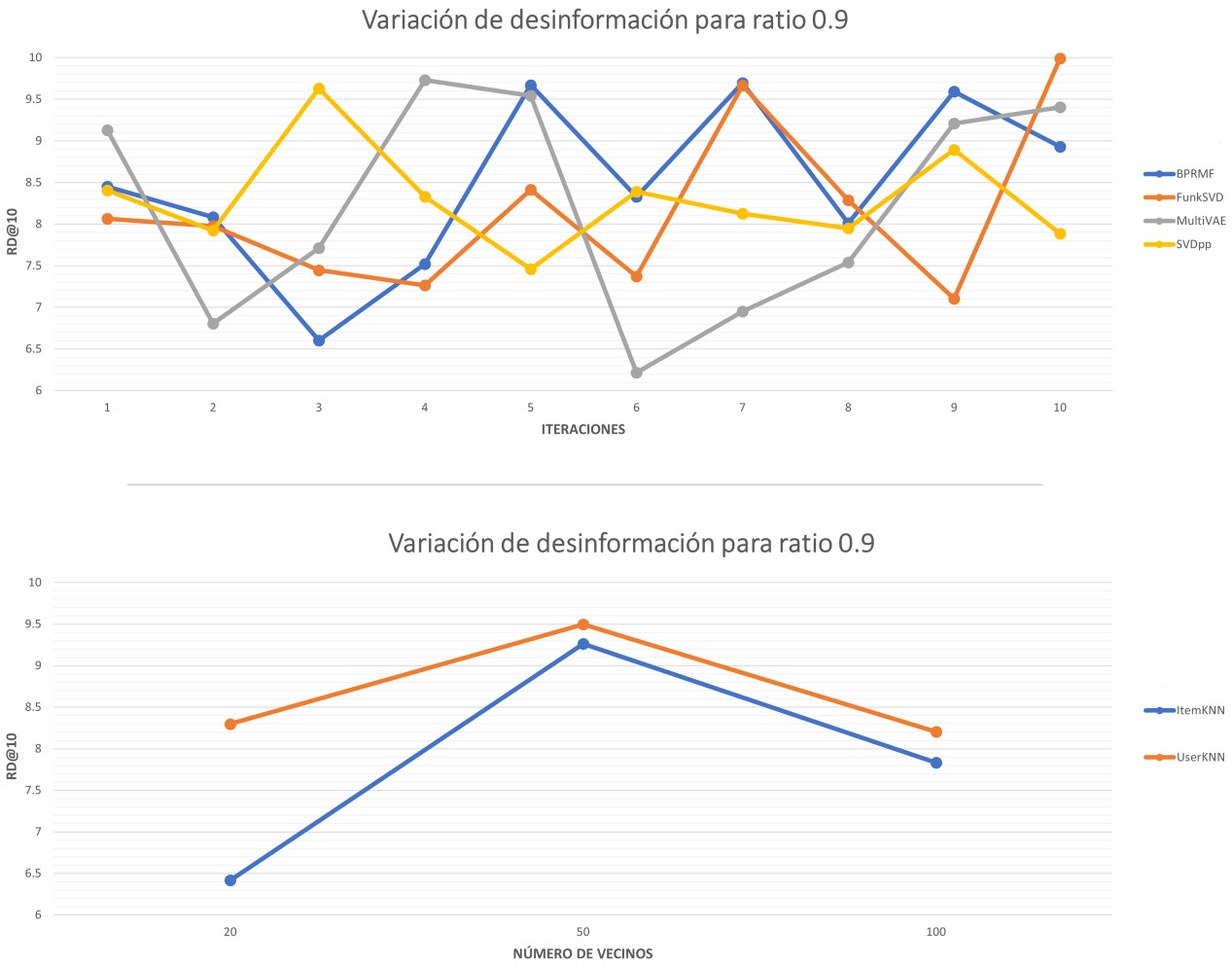

Figura 4.15: Misma información que la figura 4.12 pero con ratio 0.9

item, si se recomienda con ItemKNN, al buscar vecinos similares en un vecindario lleno de elementos desinformativos aumenta la probabilidad de que dichos vecinos sean negativos. Sin embargo, estos métodos basados en vecinos próximos siguen siendo muy similares entre ellos, por lo tanto, aunque basado en items devuelva peores resultados, por lo general ambos ofrecerán una desinformación muy similar.

En cuanto a los niveles de desinformación devueltos por el resto de técnicas de recomendación, el parámetro variable es el número de iteraciones que realiza cada algoritmo. En este caso, ni todos los recomendadores realizan el mismo número de iteraciones ni la librería Elliot muestra exactamente todas (en ocasiones se ven vacíos en alguna). Sin embargo, no se ha podido encontrar un comportamiento claro que relacione el número de iteraciones y el ratio para encontrar una tendencia con respecto a la cantidad de desinformación devuelta.

Sacando una conclusión general para los algoritmos estudiados, podemos resumir que ItemKNN se comporta mejor que UserKNN cuando se configura con 100 vecinos y unas proporciones de desinformación bajas, esto es, 0.1 y 0.3; de hecho, con este último ratio se obtiene su mejor versión, donde es capaz de superarle en todas las franjas. En cambio, con ratios altos sucede todo lo contrario y UserKNN se vuelve más efectivo en todas las configuraciones de vecinos (20, 50 y 100). Por otra parte, como ya se mencionó en la sección anterior, las técnicas basadas en factorización de matrices y redes neuronales suelen recomendar mayor desinformación y en estas gráficas se puede observar de una forma más visual. Por lo tanto, podemos considerar que la recomendación basada en usuarios es el mejor método para prevenir la desinformación, pues si la comparamos tanto con ItemKNN como con el resto de técnicas, ofrece los mejores resultados en la mayoría de casos.

4.3.3. Impacto de la neutralidad en la recomendación

En este apartado se van a mostrar los resultados cuando introducimos neutralidad en los datos. Como ya se mencionó en la sección 4.2, la neutralidad es un parámetro configurable que nos permite adaptar la cantidad de elementos con puntuación neutra (ni positiva ni negativa) y así tratar de simular una situación más realista. Cabe mencionar que hay ciertos valores para algunos sistemas de recomendación que no han sido posible su obtención, esto es provocado por la neutralidad añadida que aumenta en gran medida la cantidad de items, pues cada elemento neutral al no tener claim es un identificador único, esto hace que la cantidad de RAM y procesador requerida para el cálculo de dichas recomendaciones no sea viable con los ordenadores disponibles.

La tabla 4.4 muestra la evolución de la métrica RD@10 para los distintos recomendadores usando el conjunto de datos con proporción de desinformación calculada según todas las tuplas y variando la neutralidad para una proporción de 0, 0.25 y 0.5. Lo que se puede observar de primera mano, es que hay ciertos recomendadores que se ven muy beneficiados (es decir, reducen su desinformación), por la introducción de neutralidad, como es el caso de MultiVAE, random y los (pocos) valores que

tenemos de basados en vecinos, esto es debido a que hay muchos más elementos recomendables en el sistema y, por tanto, menos probabilidad a seleccionar uno desinformativo.

Ratio	Recm	0	0.25	0.5
0.3	Random	**3.04**	**1.00**	1.00
0.3	MostPop	3.90	3.05	3.05
0.3	ItemKNN	3.95	ND	ND
0.3	UserKNN	4.28	2.00	**0.00**
0.3	FunkSVD	3.28	4.92	6.74
0.3	SVDpp	4.14	**1.00**	ND
0.3	BPRMF	3.82	6.07	8.95
0.3	MultiVAE	3.67	1.24	2.21
0.9	Random	**7.10**	**1.00**	**1.00**
0.9	MostPop	9.99	9.99	9.00
0.9	ItemKNN	8.33	ND	ND
0.9	UserKNN	7.92	2.19	4.18
0.9	FunkSVD	8.41	8.83	9.37
0.9	SVDpp	7.52	**1.00**	ND
0.9	BPRMF	9.69	10.0	10.0
0.9	MultiVAE	9.73	2.20	1.83

Tabla 4.4: Tabla que muestra cómo varía la métrica RD@10 en cada algoritmo de recomendación para varias proporciones de neutralidad (0, 0.25 y 0.5) con los ratios de desinformación de 0.3 y 0.9. Se usa el conjunto de datos cuya proporción de desinformación ha sido calculada según todas las tuplas. ND indica que esos datos no han podido ser obtenidos (no están disponibles) ya que la librería falla por falta de memoria.

Por otro lado, tenemos al algoritmo de Most Popular que no se ve prácticamente afectado por la neutralidad, principalmente con una proporción de 0.25, aunque disminuye un poco al aumentar la neutralidad al 0.5, este último resultado probablemente sea debido a algún tipo de ruido de la librería al calcular los resultados. Es lógico pensar que este algoritmo no debería reducir la cantidad de desinformación que recomienda principalmente porque, tal como se dijo anteriormente, los items neutrales son únicos y por tanto no destacarán en los datos por su popularidad, haciendo que los elementos que eran los más populares sin neutralidad, es decir, la desinformación, se mantenga en la cima.

Finalmente, nos encontramos con los recomendadores que sí ven aumentada la cantidad de desinformación que devuelven a medida que se añade neutralidad. Este es el caso de BPRMF que con ratio de 0.3 aumenta su desinformación desde 3.82 sin neutralidad hasta los 8.95 con 0.5, también ocurre con un ratio de 0.9 donde alcanza el máximo de desinformación posible. También es el caso de FunkSVD, que con un ratio de desinformación de 0.3, la métrica RD@10 pasa de 3.28 sin neutralidad a 6.74, y con ratio de 0.9, se pasa de 8.41 a 9.37, aunque se trata de un aumento más reducido que

en el algoritmo anterior.

Sin embargo, aunque a priori parece que en los métodos basados factorización de matrices, como los dos anteriores, la neutralidad provoca un aumento de la desinformación devuelta, nos encontramos que el SVDpp, al usar una proporción de neutralidad del 0.25 y ratio de 0.9, ve reducida su desinformación desde los 7.52 elementos hasta 1, aunque es la única recomendación de este algoritmo con neutralidad que se ha podido generar, por lo tanto, faltaría ser verificado en todos los casos.

Como conclusión, y obviando que hay valores que no se han podido obtener que podrían cambiar esta premisa, vemos que los métodos que se benefician de la popularidad para desinformar, como Most Popular o BPRMF, no se ven afectados por la neutralidad, e incluso, este último se llega a aprovechar de ella. En el lado opuesto, nos encontramos con los métodos basados en vecinos próximos, el random, SVDpp y MultiVAE, siendo los más recomendados en entornos con mucha neutralidad, como es el caso de las redes sociales. Si profundizamos más, junto con la información del apartado 4.3.1, en entornos donde existe neutralidad y baja desinformación, los datos nos aconsejan las técnicas basadas en vecinos próximos, por otro lado, con la misma neutralidad pero alta desinformación, a falta de datos, sería más aconsejable el algoritmo SVDpp o, incluso, MultiVAE.

Por otro lado, el random nos ha demostrado ser un método que no interviene prácticamente en la viralización de desinformación en entornos reales, como Twitter, pues al tratarse de situaciones con cierta neutralidad en un método que se basa en la probabilidad se reduce casi al mínimo la recomendación de desinformación. Sin embargo, el riesgo que tendría usar este tipo de algoritmo es la calidad de las recomendaciones recibidas, por lo que habría que tomar una decisión entre ofrecer sugerencias *personalizadas* al usuario y que tengan en cuenta sus gustos o que estén completamente libres de desinformación. En este equilibrio, por todo lo que hemos visto y analizado, UserKNN o MultiVAE (si hay mucha neutralidad) pueden ser buenas alternativas a utilizar.

5

Conclusiones y trabajo futuro

A continuación, se va a realizar un análisis final del trabajo desarrollado, estableciendo los puntos fuertes y débiles y definiendo el sentido del mismo en el momento actual. Además, se van a definir algunas pautas que se pueden seguir para mejorar esta investigación y llegar a obtener resultados más interesantes y representativos.

5.1. Conclusiones

Con la expansión de Internet, y principalmente provocado por la llegada de los smartphones y la posibilidad de acceder a cualquier portal web en todo momento, las redes sociales fueron adquiriendo cada vez más relevancia hasta el punto de tener un papel importante para conocer e incluso influenciar en la opinión pública acerca de un tema. A su vez, los medios y organizaciones tradicionales han quedado en cierta medida opacados con la llegada de una gran cantidad de portales de noticias independientes a partir de los cuales la población puede sacar información de temas muy variados.

A priori, esta descentralización de la opinión es una buena noticia, pues permite una mayor libertad para conocer sucesos bajo una gran variedad de puntos de vista sin estar regidos por un pequeño grupo de líneas editoriales, que tendían a contar los hechos siempre buscando su propio interés. Sin embargo, esta posibilidad de llegar a un gran público, e incluso, llegar a tener la fuerza de influir en su opinión, da lugar a malas prácticas donde se busca desinformar con la intención de obtener un beneficio económico (cambiar la opinión de un producto propio o de la competencia) o político (influir en el voto).

De este modo, las redes sociales actúan de intermediarios entre la desinformación y el público, siendo fundamentales en su difusión, de hecho, ya es conocido que este tipo de noticias no verídicas, probablemente por su carácter más polémico, suelen ser las que más se propagan. Por ello, es interesante conocer qué papel toman las redes sociales, más en concreto, sus sistemas de recomendación, y así detectar si existen algoritmos más propensos a recomendar desinformación que otros con el objetivo de encontrar una solución al problema.

Con lo cual, este trabajo de investigación se centra en la creación de una base de datos de des-

información, creada desde cero, que relacione noticias falsas y verdaderas con usuarios en las redes sociales. Tras esto, usar dicha base de datos para explotarla con diversos recomendadores, obteniendo métricas con los resultados devueltos que sean capaces de ofrecer información acerca de la desinformación que generan y su rendimiento, para así establecer con argumentos qué técnicas deberían evitarse si se quiere reducir la cantidad de desinformación propagada.

Estas noticias serán extraídas del portal web de Politifact por ser uno de los servicios de verificación más grandes, dato importante pues nos interesa encontrar usuarios en las redes sociales que usen dichas noticias, por lo tanto, deben provenir de un medio reconocido. Además, la calificación que provee este fact-checker a sus noticias, calificándolas de verdaderas y falsas en un amplio abanico, facilita mucho la labor a la hora de puntuar las acciones de los usuarios. Tras esto, la red social elegida es la de Twitter, pues es el medio por excelencia a la hora de encontrar interacción entre usuarios de una forma pública y, además, es justo esta característica la que nos permite saltarnos las limitaciones que tienen las versiones gratuitas de las APIs de estos servicios (mediante el uso de crawlers, como Scrapy, y herramientas de automatización, como Selenium).

Por lo tanto, los datos recogidos de esta red social han sido bastante naturales, buscando siempre la simpleza de los mismos para facilitar la tarea de procesamiento y la futura recomendación. Es decir, la búsqueda de todos los tweets donde se menciona alguna de las noticias de Politifact ya nos permite puntuar esos elementos como positivos, por otra parte, es lógico pensar que un tweet que ha sido respondido con un enlace de una agencia verificadora es porque ha desinformado de alguna manera, por lo tanto, de una forma sencilla obtenemos también elementos calificados como negativos.

El procesamiento que se le ha dado a los datos también consta de bastante lógica: se usan como items del sistema de recomendación los enlaces de Politifact, que hemos llamado claims, pues son el elemento clave que vincula usuarios buenos y malos. De esta forma, un usuario al que le han respondido con un claim tendrá una puntuación para dicho item negativa, y un usuario que lo usa en su tweet, tendrá una puntuación para el mismo item positiva. Así, se ha conseguido traducir unos datos extensos formados por listas de tweets en JSON a simples ficheros de datos formados por tripletas (usuarios, item y puntuación) listos para ser usados en la recomendación y manteniendo, de una forma más abstracta, el comportamiento de interacción entre usuarios que tienen las redes sociales.

En cuanto al proceso de recomendación, se han seleccionado una serie de algoritmos con el objetivo de abarcar la mayor cantidad de tipos disponibles. Esto es, desde recomendación no personalizada (random y most popular) hasta modelos personalizados, como filtrado colaborativo basado en vecinos, basado en factorización de matrices y basado en redes neuronales. La lógica seguida para estudiar el impacto de estas recomendaciones ha sido la de obtener ficheros de datos donde la proporción de desinformación va ascendiendo (desde 10 % hasta 90 %) y calcular una serie de métricas para cada uno de ellos, analizando cómo cambian a medida que aumenta la recomendación.

Las métricas seleccionadas son simples pero con un claro objetivo: tenemos recuento de desin-

formación (RD), como principal, diferencia de ratio de desinformación (DRD), para comprobar si la cantidad de desinformación devuelta empeora o mejora, y luego métricas para tener un contexto sobre ciertos métodos, como son contador de recomendaciones (CR) y el promedio (CRP), que permiten conocer la cantidad de recomendaciones que es capaz de generar cada técnica, es decir, no es lo mismo devolver 2 elementos desinformativos para 5 resultados que para 20, esté último será mejor dato.

Los resultados obtenidos dejan algunas observaciones interesantes. Por un lado, hemos visto cómo los tweets desinformativos son polémicos y generan muchas respuestas, tal y como se registra en nuestra base de datos, la cual se centra en la interacción entre usuarios, y donde hemos observado que las claims relacionadas con desinformación se repitan mucho más que las asociadas a información real. Esto produce que exista un algoritmo en concreto que se vea muy afectado por este comportamiento y tienda siempre a desinformar, sobre todo a medida que aumenta la proporción de desinformación, este es el caso de Most Popular.

Por otra parte, tenemos los métodos basados en vecinos próximos (ItemKNN y UserKNN) los cuales, aunque con un bajo ratio de desinformación son los que peores resultados sacan, según se va aumentando dicho ratio se convierten en los mejores métodos. Sin embargo, no todo es bueno, debido a que la matriz es muy dispersa y estos recomendadores tienen dificultad para encontrar muchos vecinos, por lo que las recomendaciones no están completas, ni son capaces de recomendar a todos los usuarios (se quedan en alrededor del 55 %).

Como resumen, no se ha encontrado un algoritmo por excelencia que sea superior a otro en todo tipo de situaciones, es decir, los SVDs son buenos con altos ratios de desinformación, mientras BPRMF y MultiVAE actúan totalmente a la inversa. Lo que sí encontramos como solución principal es la de alejarse de popularidad a la hora de recomendar, pues es el principal factor de difusión de noticias desinformativas; luego, dependiendo del entorno donde se trabaje y gracias a los resultados obtenidos en este trabajo, se tendrán los suficientes indicios como para saber qué métodos se comportarán mejor.

Por último, el autor de este Trabajo Fin de Máster ha sido beneficiario de la ayuda para el fomento de la Investigación en Estudios de Máster-UAM 2020/2021 de la Universidad Autónoma de Madrid.

5.2. Trabajo futuro

Una de las principales mejoras que se podrían realizar en un trabajo futuro serían las de reducir las limitaciones existentes a lo largo del mismo. Es decir, si nos referimos a la obtención de la información, podríamos obtener una base de datos más completa y representativa, para ello, habría que solventar el problema del tiempo de espera que pone Twitter a la hora de extraer tweets. Si recordamos lo comentado en la sección 3.3.3, existe un crecimiento exponencial del tiempo, por ejemplo, pasar de

1.5M de tweets a 16.6M implica una duración adicional de 2h a 9,45 días. Las soluciones pueden ir desde paralelizar este proceso (habría que crear varias cuentas para saltarse los límites de la versión gratuita de la API de Twitter); otra opción sería la de adquirir una versión premium de dicha API.

Además, otra opción para obtener más datos, complementario a lo anterior, es la de añadir más portales verificadores en la búsqueda de tweets desinformativos. De esta forma, aparte de aumentar de por sí la cantidad de tweets, también aumenta la variedad de estos ya que Politifact es un portal enfocado a la política estadounidense, por lo tanto, si añadimos uno enfocado a la política española o de otro tema se podrían incluso comparar el comportamiento de los usuarios.

En cuanto a la recomendación, se pueden realizar mejoras en dos aspectos. Por un lado, con un mayor poder computacional se podrían realizar pruebas más extensas, ya que, como se mencionó en el trabajo, han existido limitaciones tanto a nivel de la CPU, que provocan unos tiempos de ejecución muy altos, como a nivel de RAM, que impide completamente obtener resultados de algunos recomendadores (problema que ocurre cuando se añade neutralidad, limitando su estudio). Por otro lado, también se podrían añadir una mayor variedad de recomendadores, desde algebraicos, basados en grafos o incluso híbridos, esto permitiría conocer más a fondo cómo se comporta la desinformación en una variedad más amplia aún de entornos.

Por último, un añadido muy interesante que, a su vez, podría ser un trabajo de investigación totalmente independiente, es el de estudiar el contenido y el contexto de las noticias o textos desinformativos para así conocer qué reglas se suelen dar a la hora de desinformar o cuáles son más propensas para su viralización. Tal como se comentó en la sección 2.3.2, esto se podría hacer a partir del uso de técnicas basadas en el procesamiento del lenguaje natural (NLP), para estudiar el contenido, y basadas en el análisis de redes sociales (SNA), para estudiar el contexto.

Bibliografía

[1] A. P. Sukhodolov and A. M. Bychkova, "Fake news as a modern media phenomenon: definition, types, role of fake news and ways of counteracting it," *Cuestiones sobre la teoría y la práctica del periodismo*, vol. 6, no. 2, 2017.

[2] B. Liu, *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data. Second Edition*. Data-Centric Systems and Applications, Springer, 2011.

[3] S. C. Fuentes Reyes and M. Ruiz Lobaina, "Minería Web: un recurso insoslayable para el profesional de la información," *ACIMED*, vol. 16, pp. 0 – 0, 10 2007.

[4] Redacción, "Ee.uu. teme que se expongan a luz pública sus programas de ciberespionaje," *La vanguardia*, Jun 2013.

[5] A. Serena, "Francia buceará en las redes sociales para combatir el fraude fiscal," *La voz de Galicia*, Oct 2019.

[6] C. G. y. A. R.-G. Ernestita Menasalvas, "Big data en salud: Retos y oportunidades," *Universidad Politécnica de Madrid*, 2017.

[7] C. M. Álvarez and L. P. M. José, *Derecho al olvido y a la intimidad en Internet: el nuevo paradigma de la privacidad en la era digital*. Reus, 2015.

[8] "What is a web crawler and how does it work?." https://en.ryte.com/wiki/Crawler.

[9] "W3schools about xpath." https://www.w3schools.com/xml/xpath_intro.asp.

[10] "Official jsoup documentation." https://jsoup.org/apidocs/.

[11] "Official pyspider documentation." http://docs.pyspider.org.

[12] "Official scrapy documentation." https://docs.scrapy.org.

[13] R. T. Fielding and R. N. Taylor, *Architectural Styles and the Design of Network-Based Software Architectures*. PhD thesis, 2000. AAI9980887.

[14] A. Campan, T. Atnafu, T. M. Truta, and J. Nolan, "Is data collection through twitter streaming api useful for academic research?," in *2018 IEEE International Conference on Big Data (Big Data)*, pp. 3638–3643, 2018.

[15] "Official tweepy documentation." https://docs.tweepy.org/en/stable/.

[16] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker, "Constraint-based recommender systems," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 161–190, Springer, 2015.

[17] Y. Koren and R. M. Bell, "Advances in collaborative filtering," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 77–118, Springer, 2015.

[18] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl, "Application of dimensionality reduction in recommender systems, a case study," 2000.

[19] S. Funk, "Netflix update: Try this at home," Dec 2006.

[20] K. Kato and T. Hosino, "Singular value decomposition for collaborative filtering on a GPU," vol. 10, p. 012017, jun 2010.

[21] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," 2012.

[22] Y. Wu, C. DuBois, A. X. Zheng, and M. Ester, "Collaborative denoising auto-encoders for top-n recommender systems," in *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, WSDM '16, (New York, NY, USA), p. 153–162, Association for Computing Machinery, 2016.

[23] "Autoencoder structure image from wikipedia." https://en.wikipedia.org/wiki/Autoencoder#/media/File:Autoencoder_structure.png.

[24] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara, "Variational autoencoders for collaborative filtering," in *Proceedings of the 2018 World Wide Web Conference*, WWW '18, (Republic and Canton of Geneva, CHE), p. 689–698, International World Wide Web Conferences Steering Committee, 2018.

[25] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2014.

[26] S. Kumar and N. Shah, "False information on web and social media: A survey," *arXiv preprint arXiv:1804.08559*, 2018.

[27] A. Alemanno, "How to counter fake news? a taxonomy of anti-fake news approaches," *European journal of risk regulation*, vol. 9, no. 1, pp. 1–5, 2018.

[28] T. O. Lerbæk and B. E. V. Olsen, "Fake news on twitter related to the refugee crisis 2016: An exploratory case study," Master's thesis, University of Agder, 2020.

[29] M. J. U. Ruiz, P. S. Bautista, and J. I. C. de Julián, "El negocio de las noticias falsas. el caso de el mundo today," *Historia y Comunicación Social*, vol. 24, no. 2, pp. 561–579, 2019.

[30] A. Zubiaga and A. Jiang, "Early detection of social media hoaxes at scale," *ACM Trans. Web*, vol. 14, Aug. 2020.

[31] A. Zubiaga, A. Aker, K. Bontcheva, M. Liakata, and R. Procter, "Detection and resolution of rumours in social media: A survey," *ACM Comput. Surv.*, vol. 51, Feb. 2018.

[32] S. Zannettou, S. Chatzis, K. Papadamou, and M. Sirivianos, "The good, the bad and the bait: Detecting and characterizing clickbait on youtube," in *2018 IEEE Security and Privacy Workshops (SPW)*, pp. 63–69, 2018.

[33] Redacción, "¿qué son los bots o seguidores fantasma que asedian el twitter de los políticos?," *La voz de Galicia*, Sep 2014.

[34] V. Rubin, N. Conroy, Y. Chen, and S. Cornwell, "Fake news or truth? using satirical cues to detect potentially misleading news," in *Proceedings of the Second Workshop on Computational Approaches to Deception Detection*, (San Diego, California), pp. 7–17, Association for Computational Linguistics, June 2016.

[35] S. Volkova, K. Shaffer, J. Y. Jang, and N. Hodas, "Separating facts from fiction: Linguistic models to classify suspicious and trusted news posts on Twitter," in *Proceedings of the 55th Annual Meeting*

of the Association for Computational Linguistics (Volume 2: Short Papers), (Vancouver, Canada), pp. 647–653, Association for Computational Linguistics, July 2017.

[36] J. Diaz-Garcia, C. Fernandez-Basso, M. Ruiz, and M. Martin-Bautista, "Mining text patterns over fake and real tweets," *Communications in Computer and Information Science*, vol. 1238 CCIS, pp. 648–660, 2020. cited By 0.

[37] J. B. Singer, "Border patrol: The rise and role of fact-checkers and their challenge to journalists' normative boundaries," vol. 22, pp. 1929–1946, June 2020.

[38] "Facebook's third-party fact-checking program." https://www.facebook.com/journalismproject/programs/third-party-fact-checking.

[39] M. A. Amazeen, "Checking the fact-checkers in 2008: Predicting political ad scrutiny and assessing consistency," vol. 15, pp. 433–464, Oct. 2014.

[40] "Official selenium website." https://www.selenium.dev/.

[41] M. Fernández, A. Bellogín, and I. Cantador, "Analysing the effect of recommendation algorithms on the amplification of misinformation," *CoRR*, vol. abs/2103.14748, 2021.

[42] D. Jannach, L. Lerche, I. Kamehkhosh, and M. Jugovac, "What recommenders recommend: an analysis of recommendation biases and possible countermeasures," *User Model. User Adapt. Interact.*, vol. 25, no. 5, pp. 427–491, 2015.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID