UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

21 | 22

Trabajo fin de grado

Aplicación de técnicas de aprendizaje profundo a sistemas de recomendación de moda

Patricia Matos Meza

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

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Aplicación de técnicas de aprendizaje profundo a sistemas de recomendación de moda

Autor: Patricia Matos Meza
Tutor: Alejandro Bellogín Kouki

abril 2022

Patricia Matos Meza
Aplicación de técnicas de aprendizaje profundo
a sistemas de recomendación de moda

Patricia Matos Meza
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi familia

El éxito en la vida no se mide por lo que logras
sino por los obstáculos que superas.

Laureano Gallardo

Agradecimientos

En primer lugar me gustaría agradecer a mi tutor Alejandro Bellogín Kouki por darme la oportunidad de llevar a cabo este Trabajo de Fin de Grado y orientarme durante todo el proceso. Así como al resto de profesores de la Escuela Politécnica Superior por impartirme el conocimiento necesario para poder ejecutar este proyecto y todos los que vengan en un futuro.

También quiero agradecerle a mi madre, María Coromoto, y a mi padre, Franklin Jesús, quienes me han servido de inspiración durante toda mi vida y representan la futura informática y persona en la que me quiero convertir. A mi tío, Jose Rafael, por recibirme con brazos abiertos en su casa y gracias a quien he conseguido llegar donde estoy. Finalmente, a mis abuelos, Rafael Ángel, Claudia María, Apolinar y, en especial, a mi abuela María Rosa, que han estado siempre allí para cuidarme y consentirme, aunque algunos ya no están presentes, se que si pudieran, darían todo por leer este documento.

Por otro lado, agradecer a mis amigos y compañeros con quienes he tenido el honor de disfrutar estos años de carrera. A Eric Morales, por haberme ofrecido su mano en los malos y buenos momentos, y por prestarme su ayuda en el desarrollo de este TFG. A Maialen Herrera, por haberme soportado como compañera de prácticas en casi todas las asignaturas. A Elena Diego, por todas las tardes y noches que pasamos en los laboratorios programando. Por último, a Ana Medina, quien aunque no sea de la carrera, siempre ha estado a mi lado y se ha convertido en un pilar fundamental.

Para concluir, quiero agradecer al resto de personas que no he mencionado, pero que saben lo importante que han sido y seguirán siendo para mi.

Resumen

El comercio electrónico se ha ido incrementando en los últimos años, más incluso en estos tiempos de pandemia mundial. Como resultado, los comercios buscan mejorar la experiencia online del cliente, centrándose en que este haga un esfuerzo mínimo en conseguir ese producto ideal que está buscando. Una de las áreas que más se beneficiaría de esto sería el ámbito de la moda.

Los métodos de recomendación son aquellas herramientas que permiten llevar a cabo esta mejora. Estos sistemas pueden ser de diferentes tipos, dependiendo del objetivo y de la información que se tenga. En este proyecto, trataremos específicamente los modelos de filtrado colaborativo, con un enfoque clásico, y los modelos de aprendizaje profundo, como son las redes neuronales para poder explotar y aprovechar las imágenes de los productos de moda.

Los modelos de filtrado colaborativo que se utilizarán se basan únicamente en las preferencias del usuario, mientras que las redes neuronales toman en cuenta también la similitud entre los productos, extrayendo características de sus imágenes. Se estudiará el comportamiento de los modelos en ambos casos, y los resultados que consiguen al enfrentarse a un problema común de recomendación de moda.

Además, el dominio de la recomendación de moda se presta a una representación visual tanto de los datos como de las interacciones de los usuarios con el sistema. Para ello, en el contexto de este trabajo, se ha desarrollado una plataforma web que nos permite mostrar todo el proceso de generación de las recomendaciones, enfrentando y comparando los modelos explorados, así como las preferencias previas de los usuarios que pertenecen a dicho sistema. De esta manera, conseguimos una prueba de concepto de cómo funcionaría un sistema de recomendación de moda con datos e interacciones cercanos al mundo real.

Palabras clave

Sistemas de recomendación, aprendizaje profundo, redes neuronales convolucionales, filtrado colaborativo, recomendación de moda

Abstract

Electronic commerce (e-commerce) has been increasing in recent years, even more so in these times of global pandemic. As a result, business owners seek to improve their customers' online experience by ensuring that customers put in minimal effort to get the ideal product they are looking for. One of the areas that would benefit the most from this would be the field of fashion.

Recommendation systems are those tools that allow us to carry out this improvement. These systems can be classified into different types, depending on the objective and the information available. In this project, we will specifically deal with collaborative filtering models, with a classic approach, and deep learning models, such as neural networks to be able to exploit and take advantage of the images of fashion products.

The collaborative filtering models that will be used herein are based only on user preferences, while neural networks also take into account the similarity between products, by extracting features from their images. The behavior of the models in both cases will be studied, as well as the results they achieve when faced with a common fashion recommendation problem.

Furthermore, the fashion recommendation domain lends itself to a visual representation of both data and user interactions with the system. For that purpose, in the context of this work, a web platform has been developed that allows us to show the entire process of generating the recommendations, confronting and comparing the models explored, as well as the previous preferences of the users that belong to said system. Thus, we get a proof of concept of how a fashion recommendation system would work with data and interactions close to those from the real world.

Keywords

Recommendation systems, deep learning, convolutional neural networks, collaborative filtering, fashion recommendation

Índice

Listas

Lista de ecuaciones

Lista de figuras

Lista de tablas

1

Introducción

1.1. Motivación del proyecto

El comercio electrónico es uno de los avances más importantes de los últimos tiempos, que ha ido aumentando cada vez más de manera natural, y en particular, debido a las cuarentenas ocasionadas por la pandemia mundial del virus COVID-19. A raíz de ello, también ha incrementado el interés en mejorar cada vez más estos servicios ofrecidos, poniendo el foco en los gustos del cliente. Es en este punto donde los sistemas de recomendación entran en juego.

Los sistemas de recomendación son técnicas que nos permiten generar sugerencias sobre un elemento a un individuo en concreto, basándonos en las preferencias del mismo. Estos sistemas pueden ser implementados de diferentes maneras, dependiendo del objetivo que se tenga.

En este proyecto, indagaremos acerca de estas implementaciones, enfocadas en la recomendación de moda, basándonos en una parte fundamental del comercio electrónico, como lo son los productos y sus imágenes. Estas imágenes nos dan la oportunidad de explorar y juntar el campo de las redes neuronales con las recomendaciones basadas en usuarios. En concreto, trataremos las redes neuronales convolucionales, claves en las investigaciones recientes dentro del aprendizaje profundo.

1.2. Objetivos

El objetivo principal de este trabajo es el estudio de los sistemas de recomendación de moda. Enfocado en los modelos de aprendizaje profundo y cómo es su comportamiento frente a modelos más tradicionales como el filtrado colaborativo. Estos modelos de aprendizaje profundo son las redes neuronales convolucionales, utilizadas para clasificar y recomendar a partir de imágenes.

Además, se pretende realizar una plataforma web desde cero, en donde se puedan observar empíricamente las recomendaciones generadas por estos modelos, tanto clásicos como neuronales, y las preferencias del usuario al cual van dirigidas estas recomendaciones.

1.3. Estructura del trabajo

Este documento está estructurado de la siguiente manera:

Capítulo 1. Introducción. Se describe el problema, los objetivos del proyecto y la estructura del mismo.

Capítulo 2. Estado del arte. Se explica el marco teórico del proyecto, detallando los conceptos necesarios para su entendimiento.

Capítulo 3. Análisis, diseño e implementación. Contiene el análisis del problema, las decisiones de diseño tomadas y la implementación final del sistema.

Capítulo 4. Pruebas y resultados. Se describe cómo se han realizado los experimentos, los datos utilizados y, sus resultados.

Capítulo 5. Conclusiones y trabajo futuro. Contiene la conclusión del proyecto y sus posibles ampliaciones.

2

Estado del arte

En este capítulo se exploran los conceptos fundamentales para el entendimiento de este trabajo. Se introducen los sistemas de recomendación, en concreto, cuando se aplican a recomendación de moda y las tecnologías necesarias para presentarlos en una aplicación Web. Además, se estudian las redes neuronales convolucionales, su arquitectura, así como también su contribución a los problemas de recomendación.

2.1. Sistemas de recomendación

Los sistemas de recomendación son un conjunto de herramientas, tanto de software como de técnicas de aprendizaje automático, que nos permiten generar sugerencias bien formadas de productos basadas en el gusto propio de cada usuario. Estos sistemas actúan como guía al usuario recomendando los productos más relevantes sin necesidad de que el usuario los busque explícitamente [1].

En los últimos años, las investigaciones acerca de estas técnicas ha aumentado, así como también su uso en grandes aplicaciones que hoy son indispensables en nuestro día a día. Por ejemplo, las recomendaciones de películas en Netflix, de videos en Youtube, productos en Amazon, entre otros.

Los sistemas de recomendación pueden dividirse, principalmente, en cinco grandes categorías:

Basados en contenido. Estos sistemas recomiendan ítems que son similares a los que le ha gustado al usuario en ocasiones anteriores. Se basa en comparar las características de esos productos [1].

Basados en filtrado colaborativo. Estos sistemas recomiendan apoyándose en los items que les han gustado a usuarios similares. Esta similitud entre usuarios se suele calcular en base al historial de valoraciones [1].

Demográficos. Estos sistemas generan recomendaciones basadas en el demográfico de un usuario. Se asume que por cada nicho demográfico, debería generarse recomendaciones distintas [1].

Basados en el conocimiento. Estos sistemas se basan en conocimientos previos que se

tengan acerca de cómo utilizar cierto ítem, y cómo se ajusta a las necesidades del usuario [1].

Híbridos. Estos sistemas, como indica su nombre, recomienda en base a combinaciones de las técnicas anteriores, tomando las ventajas de cada una de las anteriores y generando una mezcla entre ellas. [1].

En este trabajo, se han seleccionado los sistemas de recomendación basados en filtrado colaborativo, debido a que es la categoría que más se ajusta a los datos que disponemos. Además, se considera que nos ofrecerá un mejor contraste a la hora de comparar resultados con los recomendadores basados en aprendizaje profundo.

2.1.1. Algoritmos de recomendación no personalizados

Se han establecido unos algoritmos que actúan como líneas base para determinar el grado de mejora que presentan los modelos de recomendación seleccionados. Estos algoritmos son el recomendador **aleatorio** y el recomendador por **popularidad**.

El recomendador aleatorio recomienda productos de manera aleatoria a cada usuario, mientras que, el recomendador de popularidad, recomienda los productos dándole más relevancia a aquellos con mayor cantidad de valoraciones.

2.1.2. Filtrado colaborativo y modelo SVD

Los métodos de recomendación basados en filtrado colaborativo destacan por generar recomendaciones de productos basados en patrones de múltiples lugares: compras, grandes repositorios de datos, etc. En otras palabras, se basa en agrupar usuarios con gustos similares y generar estas recomendaciones en base a ello [2].

Pensemos en el caso de un grupo de usuarios que frecuentan una tienda de ropa virtual, y, cada vez que realizan una compra, valoran dichos productos comprados. Si una cantidad de usuarios de este grupo compra y valora un jersey que el resto del grupo aún no comprado, entonces el recomendador colaborativo sugerirá al resto del grupo comprar este jersey. Este método resulta familiar porque se basa en el modelo de vecinos próximos, centrado en el propio usuario, al decidir quienes de ellos tienen gustos similares.

Las técnicas de filtrado colaborativo se pueden agrupar en tres grandes categorías: basadas en memoria, basadas en modelos o híbridas [1]. En este trabajo, se utilizará un filtrado colaborativo basado en el modelo de factorización de matrices mediante Singular Value Decomposition (SVD) o descomposición en valores singulares [1], que se describirá en la siguiente sección.

Modelo SVD

SVD es una técnica de factorización de matrices bastante reconocida en el ámbito de sistemas de recomendación, cuyo objetivo es factorizar una matriz de ciertas dimensiones en tres matrices que se relacionan de la siguiente manera:

$$A = USV^t \tag{2.1}$$

donde U y V son matrices ortogonales, y S es una matriz que contiene todos los valores singulares de A como su diagonal [3].

SVD es la técnica escogida para el filtrado colaborativo y trabajará directamente con las valoraciones de los usuarios. Esta técnica, en esencia, generará una aproximación de las valoraciones que daría cada usuario a todos los productos presentes en nuestro dataset, guardándolas en una matriz A. Esta matriz será utilizada para generar las recomendaciones de productos para cada usuario, basado en las valoraciones (*ratings*) que se hayan predecido para cada producto [3].

2.1.3. Aprendizaje profundo y redes convolucionales

El aprendizaje profundo es un subconjunto de técnicas del aprendizaje automático con múltiples niveles de representación y abstracción de los datos. Cualquier arquitectura neuronal que optimice una función utilizando alguna variante de descenso por gradiente estocástico, es considerada perteneciente al aprendizaje profundo [4].

Existen múltiples ventajas al utilizar aprendizaje profundo en los problemas de recomendación. Dos de las más importantes son: a) capaces de modelar datos con relaciones no lineales y b) capturar relaciones complejas en los patrones de comportamiento de los usuarios. Además, muestran resultados favorables en las tareas de modelado sequencial, como lo son los problemas de reconocimiento del discurso, chatbots, entre otros [4].

Para este trabajo se han seleccionado tres redes neuronales convolucionales, puesto que se trabaja con las imágenes de prendas de ropa. Estas redes son ResNet 50, VGG 16 e Inception V3 y, cada una de estas redes, se conectan con un modelo de vecinos próximos para generar las recomendaciones a partir de la información generada por estas redes. Esta información se ampliará en secciones posteriores.

Arquitectura

Como hemos mencionado en el apartado anterior, las redes convolucionales (CNN) se basan en recibir imágenes como entrada, para ser tratadas en el resto de capas de la red. Las CNN están compuestas por tres tipos de capas diferentes: las capas convolucionales, las capas de pooling (pooling

layers) y las capas completamente conectadas (fully-connected).

En la figura 2.1, podemos ver un ejemplo simple de la arquitectura de una red convolucional.

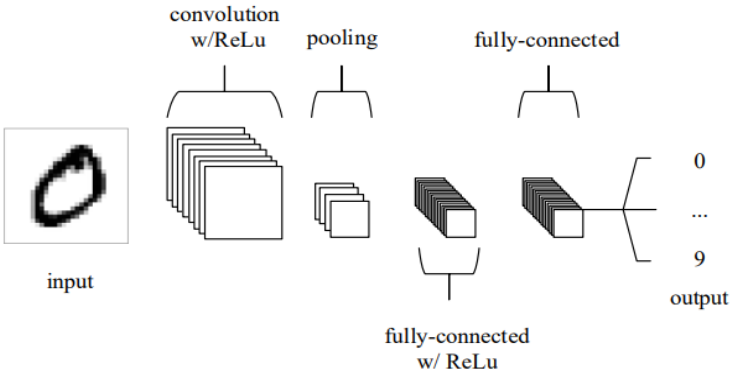


Figura 2.1: Arquitectura simple de una CNN [5].

La capa input que se puede observar en la imagen, contiene los píxeles de las imágenes de entrada.

Las capas convolucionales trabajan con filtros, llamados también kernels, que son aplicados a las imágenes. Estos kernels suelen tener pequeñas dimensiones y cuando son aplicados a las imágenes generan mapas de activación 2D.

Las capas pooling tienen el trabajo de reducir la dimensionalidad de las representaciones (mapas de activación generados por la capa convolucional) y, por lo tanto, reducen el número de parámetros y la complejidad del modelo. Esto lo hacen generalmente utilizando la función MAX, y por ello se les llama Max-pooling layers.

Finalmente, las capas fully-connected contienen neuronas que están conectadas directamente a neuronas en capas adyacentes, sin estar conectadas a otras capas intermedias [5].

Embeddings

Anteriormente, hemos mencionado que las capas de pooling generan mapas de activación con menor dimensionalidad que la entrada. Los embeddings son un claro ejemplo de estos mapas.

Los embeddings no son más que representaciones de menor dimensionalidad de una imagen, de tal forma que, a partir de esa imagen obtenemos un vector pequeño pero muy denso, donde cada valor representa una característica diferente. Son útiles en problemas de clasificación, y serán útiles en este problema de recomendación, para encontrar las similitudes entre imágenes y poder recomendar los productos. Si los trasladamos a nuestro problema, calcularemos los embeddings utilizando las tres redes neuronales seleccionadas por separado, y luego calcularemos la similitud utilizando el modelo de vecinos próximos. Esto se explicará más a fondo en la próxima sección.

Cabe destacar que los embeddings se pueden aplicar a cualquier objeto, pero en este trabajo

se aplicarán únicamente a las imágenes que recibimos como entrada, es decir, las imágenes de los productos. Para ello, se utilizará el modelo de vecinos próximos.

K vecinos próximos (KNN, del inglés *K nearest neighbors*) es un modelo de aprendizaje supervisado basado en instancias, es decir, se basa únicamente en muestras de datos. Suele ser utilizado para problemas de clasificación. Según un dato d recibido, este modelo recupera los k vecinos más próximos a ese dato, formando el vecindario para el dato d [6].

En este TFG, utilizaremos este modelo para calcular la similitud entre las imágenes de dos posibles productos. Esto se hará mediante los embeddings. Cada red devuelve un embedding para cada imagen que se le de como entrada. Estos embeddings son con los que se entrena el modelo KNN, al que luego se le pasará otro embedding de un producto, y devolverá la similitud con cada uno de los elementos con los que ha sido entrenado. Esas distancias son las que luego se utilizarán para calcular el peso de cada recomendación.

Cada red neuronal calcula los embeddings de una manera diferente. Por ende, en las siguientes secciones se describirán algunas de ellas.

Modelo neuronal ResNet 50

El modelo ResNet proviene de las Redes neuronales residuales (Residual Networks). El 50 viene dado por el número de capas de neuronas en la red.

Las redes residuales son redes neuronales artificiales que utilizan atajos en sus conexiones (*shortcuts*) para saltarse capas. Generalmente son implementadas con saltos dobles o triples entre capas que contienen Batch Normalization y ReLu.

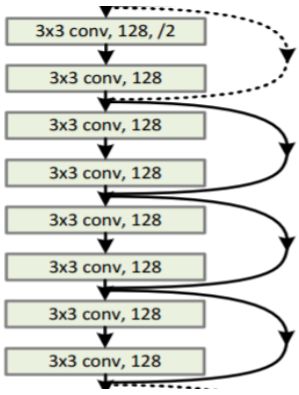


Figura 2.2: Ejemplo de arquitectura de una ResNet [7].

En la figura 2.2, se pueden observar las flechas negras que indican ese salto entre dos capas dentro de la misma red. Estos saltos permiten a la red evitar el problema de desvanecimiento del gradiente. Este problema viene dado porque al incrementar la profundidad de una red (añadir más capas), es más probable que haya un error en entrenamiento alto debido a la propagación de este gradiente. [7]

Como dato interesante, la arquitectura ResNet-152 ganó el reto ImageNET en 2015. Su triunfo viene dado ya que nos ha permitido entrenar redes con muchas capas ocultas (redes extremadamente profundas) correctamente. En este caso, se utiliza una versión reducida con tan solo 50 capas.

Modelo neuronal Inception V3

Inception V3 es una red neuronal convolucional usualmente utilizada en el análisis de imágenes y en la detección de objetos. Proviene de un módulo de la red GoogleNET, que fue introducido en el reto de ImageNET. Contiene aproximadamente 25 millones de parámetros [8].

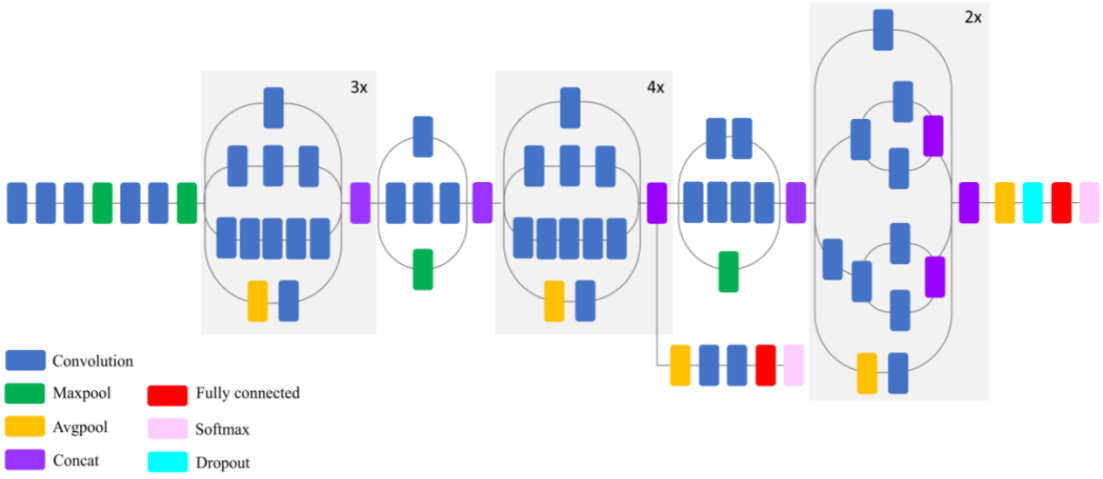


Figura 2.3: Ejemplo de arquitectura de Inception V3 [9].

La red de GoogleNET fue creada con la premisa de que muchas conexiones entre capas son poco efectivas y redundantes debido a la correlación que pueda haber entre ellas. Utiliza un módulo de *Inception*, con capas en paralelo y beneficiándose de los clasificadores auxiliares en las capas intermedias.

En esta red, las capas fully-connected son sustituidas por capas de pooling, lo cual disminuye el número de parámetros significativamente [9].

Modelo neuronal VGG 16

VGG-16 es un red convolucional introducida en el año 2014. Sus siglas hacen referencia a *Visual Geometry Group*, que es el nombre del grupo de investigadores de la universidad de Oxford que desarrollaron su estructura. El 16 implica el número de capas de la red.

Esta red logró alcanzar el 92.7 % de precisión en el reto de ImageNET en el 2014. Su implementación es bastante sencilla, con lo cual es utilizada frecuentemente para la clasificación de imágenes [10].

Su arquitectura, en general, se caracteriza por su profundidad con un filtro pequeño de 3 x 3, en comparación a su predecesor AlexNET [9].

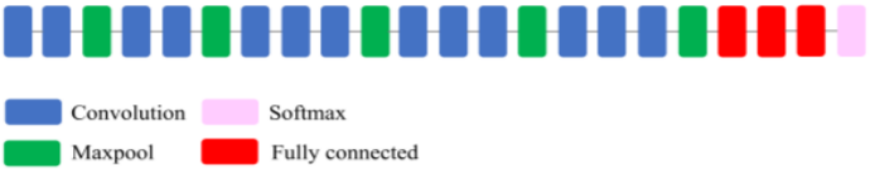


Figura 2.4: Arquitectura de VGG-16 [9].

En la figura 2.4, se muestra que la arquitectura de la red consiste en 13 capas convolucionales (en azul) y tres capas fully connected (en rojo). Además, también posee pilas de filtros pequeños de 3x3 con stride de 1, seguidos de capas no lineales.

Esta red ha permitido revelar al mundo de la investigación que la profundidad de una red es un factor importante en la obtención de alta precisión [9].

2.1.4. Métricas

Las métricas son herramientas que se utilizan para medir qué tan bueno, o qué tan preciso, es aquello generado por un modelo de aprendizaje automático. Para los sistemas de recomendación, existen métricas que nos ayudan a medir qué tan relevantes son estas recomendaciones. Para evaluar nuestros algoritmos se han seleccionado dos de las más famosas: *Precision@k* y *Recall@k* [11].

Precisión @ k

La precisión de un modelo se define como la representación de la probabilidad de que un elemento recomendado sea relevante. En este caso, *Precision@k* le añade el concepto de evaluar el conjunto de los k primeros elementos recomendados [11].

Matemáticamente tenemos que,

$$Precision@k = \frac{Nrelevantes_{@k}}{Nrecomendados_{@k}} \tag{2.2}$$

Donde $Nrelevantes_{@k}$ se refiere al número de productos relevantes en el top-k y, $Nrecomendados_{@k}$ al número total de elementos recomendados en el top-k.

Recall @ k

El recall de un modelo es la proporción entre los elementos relevantes seleccionados frente al total de elementos relevantes disponibles. Al igual que en precisión, *Recall@k* le añade el concepto de evaluar el conjunto de los k primeros elementos recomendados [11].

Matemáticamente tenemos que,

$$Recall@k = \frac{Nrelevantes_{@k}}{Nrelevantes} \tag{2.3}$$

Donde $Nrelevantes_{@k}$ se refiere al número de productos relevantes en el top-k y, $Nrelevantes$ al número total de elementos relevantes disponibles.

2.2. Recomendación de moda

La recomendación de moda es un tema que en los últimos años se ha empezado a investigar y, a utilizar en páginas de comercio electrónico, aplicaciones de música, entre otros (ver [12–14]).

Existen múltiples problemas en los que se pone a prueba la recomendación de moda. Uno de ellos, por ejemplo, es la recomendación de un outfit completo. En este problema, lo que se busca es generar una recomendación con ítems que congenien entre sí y, no sean funcionalmente redundantes (ver [15]). Por otro lado, tenemos los problemas de "rellenar el espacio en blanco" o Fill in the blanks (FITB) , los cuales presentan como entrada un outfit incompleto, y se espera una que se genere un recomendación para ese ítem faltante (ver [16]).

Finalmente, el más común es el problema de generar recomendaciones de un producto de moda en base a gustos del usuario o características del propio producto. Está orientado a la experiencia del usuario al utilizar una herramienta cotidiana. En este TFG, se va a tratar este problema en específico.

Este tipo de recomendación facilita la experiencia del usuario, puesto que este ya no tiene que buscar ese producto ideal explícitamente, sino que la propia página o aplicación hace el trabajo, y recomienda ese producto basado en características propias del mismo.

Estas recomendaciones pueden tomar en cuenta múltiples características de los productos, así como los gustos del usuario, el número de validaciones, la categoría y su similitud en cuanto a otros items existentes.

Uno de los principales problemas con los que nos podemos encontrar es el arranque en frío o *Cold Start* [17]. Esto se produce cuando el modelo no tiene la suficiente información, tanto de gustos de usuario o de los propios productos, como para generar buenas recomendaciones.

Si lo trasladamos a nuestro escenario, es posible que existan usuarios que no hayan valorado productos, con lo cual no tenemos información de su actividad ni de sus gustos. También puede darse el caso de que un producto sea completamente nuevo y no se haya valorado todavía y, por tanto, a la hora de calcular pesos, no se tomará tan en cuenta como productos con mayor antigüedad y mayor cantidad de ratings.

Existen múltiples maneras de mitigar este problema, y una de ellas podría ser un modelo híbrido

entre un filtrado colaborativo y uno basado en contenido [17]. En este trabajo, se ha optado por implementar pesos para disminuir o aumentar lo mucho que "importará" el rating del producto, la similitud y lo reciente que pueda ser en el resultado final. Esto se explicará más en detalle en la sección 3.3.2.

2.3. Tecnologías Web

Al igual que se hizo en un TFG reciente (ver [18]), y como se ha comentado en la introducción (capítulo 1), uno de los objetivos de este proyecto es crear una interfaz gráfica en donde se puedan ver explícitamente las recomendaciones generadas. Para ello, en las secciones 2.3.1 y 2.3.2, se analizan algunas tecnologías que se han revisado para tomar una decisión de cara a la implementación de esta interfaz.

Realizaremos los desarrollos, tanto de la interfaz como de los algoritmos de recomendación, en el lenguaje de programación Python, ya que nos ofrece un grado de moldeabilidad bastante elevado y es un lenguaje rico en librerías que serán útiles para llevar a cabo la funcionalidad. Se discutirá más en detalle en la sección de implementación 3.3.

2.3.1. Librerías para encapsular modelos

Existen numerosas alternativas para encapsular los modelos de aprendizaje automático. Una de ellas es utilizar los frameworks de Django o Flask, que nos permiten desarrollar ese servidor que recibe y responde peticiones a través de modulos en Python.

En el caso de *Django*, es un framework gratis, de código abierto y de alto nivel que nos permite desarrollar rápidamente con diseños pragmáticos. Su objetivo principal es facilitar el proceso de creación de una aplicación a los desarrolladores web. Sus ventajas son: garantiza la rapidez al desarrollar; fomenta la escalabilidad del software con gran demanda de tráfico y, al tener componentes de seguridad creadas, evita que las aplicaciones puedan sufrir ataques comunes (inyección de código SQL, secuencia de comandos en sitios cruzados, ...) [1].

Uno de los problemas que se han considerado a la hora de desarrollar la interfaz en Django, es que al ofrecernos una gran variedad de componentes que vienen con la propia librería que no necesitamos, se estarían desaprovechando esos recursos. La interfaz que se quiere desarrollar es bastante sencilla y no se busca que sea una aplicación web que sea utilizada por un usuario final (al menos no en estos momentos, aunque se podría plantear para el futuro). Su objetivo es que sirva como una presentación de los algoritmos y que simule la aplicación que podrían tener en el mundo real.

En cuanto a *Flask*, es similar a Django, con la diferencia de que es mucho más simple y con menor

[1] "Django" https://www.djangoproject.com/. Accessed: 2022-03-18

cantidad de componentes pre-hechos. Es decir, es posible que al instalar flask la primera vez no venga con toda la funcionalidad que se ofrece. Se trabaja a base de instalar sub-librerías según se vaya necesitando. En otras palabras, es escalable [2]. Por ello, se considera un punto a favor en cuanto a utilizar Django, ya que al ir instalando según se va desarrollando, se garantiza que no se derrochen recursos, que el proyecto no sea tan complejo ni pesado, y, por ende, que el rendimiento del mismo no se vea afectado por estas razones.

La naturaleza de este TFG es centrarnos en los sistemas de recomendación y cómo podemos mejorar su funcionamiento. Django y Flask, aunque aseguran un desarrollo eficaz, y, en el caso de Flask, el poder escalarlo según se considere, son soluciones demasiado complejas para lo que se quiere y no concuerdan con el objetivo del trabajo.

A raíz de lo anterior, se ha contemplado una tercera opción. Se trata de una librería de python llamada FastAPI. Esta sería la librería que se ha seleccionado para encapsular los algoritmos. En el punto siguiente, se comentará el por qué de la decisión y se ampliará la información acerca de FastAPI.

FastAPI

Una solución sencilla para encapsular los modelos, podría ser disponer de una API REST, que recibiera peticiones, dirigidas a los modelos, y respondiera a las mismas con los resultados obtenidos.

FastAPI nos permite crear rápidamente esta parte del servidor. Esta API REST, se comunica a través de URLs con la parte frontal de la aplicación, donde cada URL corresponde a un método distinto. Su principal ventaja es que solo facilita la parte de la API, por lo que su rendimiento es muy alto. Al centrarse únicamente en la conexión con los modelos existentes, su desarrollo es más rápido en comparación con Django y Flask, en donde se tendría que configurar muchos parámetros al comenzar. Es un software bastante intuitivo, facilita el desarrollo y contiene maneras de revisar el estatus de las peticiones y, dado el caso, llevar una traza para corrección de posibles errores *FastAPI* [3].

En conclusión, después de analizar esta librería, se ha escogido esta opción, ya que su implementación es fácil y rápida, perfecta para lo que se necesita.

2.3.2. Librerías para interfaz Web

Sabiendo que se tiene el *backend* encapsulado en una API REST, se necesita escoger la herramienta a utilizar para desplegar el *frontend* de la aplicación.

Una de las opciones podría ser un desarrollo web tradicional. Es decir, una página con HTML5 manejada dinámicamente con JavaScript, y peticiones AJAX que se comunicarían con nuestra API.

[2] "Flask" https://flask.palletsprojects.com/en/2.0.x/. Accessed: 2022-03-18

[3] "FastAPI" https://fastapi.tiangolo.com/. Accessed: 2022-03-18

Esta opción, es una de las más comunes, pero desarrollar una página Web completamente desde cero no es una tarea fácil y consumiría una gran cantidad de tiempo innecesario.

Otra opción, es utilizar los frameworks de Django o Flask, pero únicamente para la parte frontal de la aplicación. Nuevamente, se tendría el problema de recursos innecesarios que afectarían al rendimiento de la interfaz. En este caso, sería aún mayor este problema porque tampoco estaríamos utilizando las facilidades de backend que nos ofrecen.

Siguiendo el análisis anterior, la mejor opción parece ser un desarrollo software tradicional. Sin embargo, se pueden utilizar frameworks para facilitar el proceso de diseño de la web, disminuir el tiempo de desarrollo y no crear la interfaz desde cero.

Uno de los frameworks que se han valorado es *Bootstrap* [4]. Esta librería permite tener un estilo predeterminado en todos los componentes de HTML5 que se quieran utilizar. Todos estos componentes son personalizables. Como segunda opción, se tiene la posibilidad de utilizar *ReactJS*, que es una librería de JavaScript desarrollada por Facebook.

No existe demasiada diferencia entre las dos librerías anteriores, pero se ha decidido utilizar ReactJS, ya que es una herramienta que se está utilizando cada vez más y genera curiosidad estudiar como funciona, aún cuando no se tiene experiencia en ella. En el siguiente apartado, describiremos sus características.

Cabe mencionar que se ha utilizado una plantilla para la interfaz, y reducir aún más el tiempo de desarrollo. Se discutirá en detalle en la sección de implementación 3.3.

React

ReactJS, o React, es una librería de JavaScript orietada únicamente al desarrollo de interfaces de usuario. Se encarga de que estas sean interactivas, y de manejar los componentes de forma óptima cuando estos se actualicen o sufran alguna interacción [5].

Es similar a un lenguaje orientado a objetos, ya que se basa en componentes. Cada componente tiene una funcionalidad y un estado encapsulado. Al estar basada en JavaScript, se pueden pasar directamente los datos de forma sencilla a la aplicación o para enseñarlos en pantalla. Su gran ventaja es que, al estar el servidor encapsulado en URLs, se pueden hacer peticiones a esos URLs desde los componentes y de esta manera, traer esa información para enseñarla al usuario en la interfaz.

Una de las desventajas que posee, es que su código no es demasiado intuitivo y, es fácil perder la traza de cualquier flujo que pueda tener la aplicación. Sin embargo, esto no dificulta demasiado el proceso en nuestro caso, ya que la aplicación web no tiene flujos complejos.

[4] "Bootstrap" https://getbootstrap.com/. Accessed: 2022-03-18

[5] "React" https://es.reactjs.org/. Accessed: 2022-03-18

3

Análisis, diseño e implementación

En este capítulo se discutirá el análisis de requisitos de todo el software desarrollado. Se detallará el diseño de la aplicación, su estructura y su flujo de comunicaciones. Finalmente, se hablará de la implementación de los modelos de recomendación y de la aplicación web.

3.1. Análisis

En esta sección, se analizarán los requisitos funcionales y no funcionales de la aplicación, y de los algoritmos desarrollados. Para entender mejor esto, en la figura 3.1 se muestra un esquema que define la estructura de la aplicación por módulos.

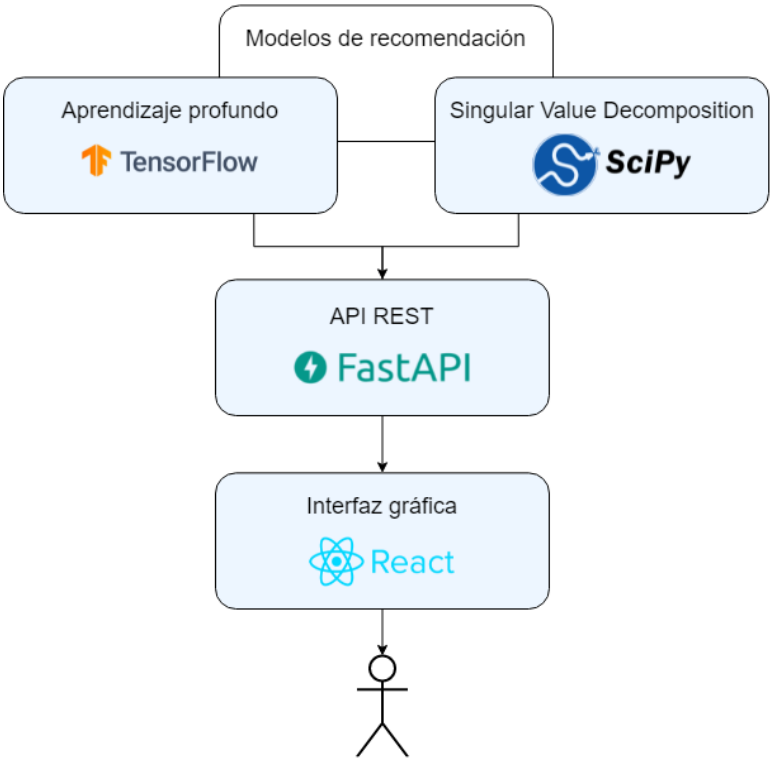


Figura 3.1: Estructura del proyecto.

3.1.1. Requisitos funcionales

Los requisitos funcionales de este TFG se pueden dividir en tres grupos, correspondientes a los diferentes módulos presentados anteriormente: modelos de recomendación, API REST y la interfaz gráfica.

Requisitos funcionales de los modelos de recomendación

RF-1.– Los modelos de recomendación deberán generar un ranking ordenado según la relevancia de las recomendaciones.

RF-2.– La relevancia de las recomendaciones se medirá en base al rating del producto, su similitud con los gustos del usuario y la fecha de su última valoración.

RF-3.– Los modelos deben soportar las peticiones dado un usuario específico.

RF-4.– Los modelos de recomendación basados en aprendizaje automático utilizarán redes neuronales pre-entrenadas.

Requisitos funcionales de la API REST

RF-5.– La API facilitará una lista de usuarios definidos por su ID. El usuario de la página web podrá elegir entre ellos.

RF-6.– La API facilitará una lista de los posibles algoritmos de recomendación entre los cuales el usuario podrá escoger.

RF-7.– La API facilitará una lista de los productos que han sido valorados por el usuario seleccionado.

RF-8.– La API recibirá peticiones con los mismos parámetros para todos los modelos de recomendación. Estos parámetros son el usuario y el recomendador escogido.

RF-9.– La API responderá a las peticiones con el mismo tipo de datos para todos los modelos de recomendación. Estos datos serán la lista con el ranking de recomendaciones.

RF-10.– La API dispondrá de una documentación en la que se pueda consultar información acerca de los métodos y sus parámetros.

Requisitos funcionales de la interfaz gráfica

RF-11.– La interfaz constará de una página principal con las preferencias del usuario seleccionado y las recomendaciones generadas por los modelos.

RF-12.– La interfaz permitirá seleccionar el ID del usuario sobre el que se quiere generar recomendaciones.

RF-13.– La interfaz debe mostrar los productos que han sido valorados por el usuario seleccionado.

RF-14.– La interfaz permitirá seleccionar dos algoritmos de recomendación para ser comparados.

RF-15.– La interfaz debe mostrar el ranking de las recomendaciones generadas por los algoritmos seleccionados, uno debajo del otro para poder comparar resultados.

RF-16.– Se mostrarán los 11 primeros productos en el ranking de recomendaciones para cada algoritmo de recomendación.

3.1.2. Requisitos no funcionales

RNF-1.– El back-end de la aplicación estará desarrollado en su totalidad utilizando el lenguaje Python.

RNF-2.– El front-end de la aplicación estará desarrollado en ReactJS.

RNF-3.– La aplicación se ejecutará utilizando el entorno proporcionado por *Anaconda* [1].

RNF-4.– La aplicación tendrá un diseño *responsive*, para ser correctamente accesible desde distintos dispositivos con distintas resoluciones.

RNF-5.– Los tiempos de respuesta de la API deben de ser bajos, por ello se guardarán todas las recomendaciones generadas en una caché.

3.2. Diseño

En esta sección se plasmará el diseño de la aplicación. Para ello, se discutirá la estructura de la aplicación, con los módulos que la conforman. Se detallará cuáles son los flujos de comunicación dentro de la misma y el ciclo de vida que se ha elegido al desarrollarla.

3.2.1. Estructura de la aplicación

Como se puede ver en la figura 3.1, para desarrollar la aplicación se ha optado por una estructura modular. De esta manera, se garantiza que todos los objetos que la componen estén debidamente encapsulados, sea flexible por si en algún momento se quiere mantener o incluso añadir más funcionalidad. Los tres módulos más importantes son los siguientes:

Modelos de recomendación. Este módulo contiene todos los modelos de recomendación que se utilizarán en este proyecto. Estos modelos son los siguientes:

- **Redes neuronales:** corresponde a la implementación de las redes ResNet, Inception-V3 y VGG-16, como se ha visto en la sección 2.1.3. Estas redes se han desarrollado con ayuda de la librería Tensorflow.
- **Filtrado colaborativo:** corresponde a la implementación de los algoritmos no personalizados Aleatorio y basado en popularidad (sección 2.1.1) y del modelo SVD (sección 2.1.2).

API Rest. Este módulo contiene la funcionalidad de la API y, por ende, el back-end del sistema, como se ha discutido en la sección 2.3.1. Contiene dos niveles:

- **Métodos propios de la API:** corresponde a la implementación de los métodos que van a ser llamados desde la aplicación web con peticiones a sus URLs asociadas. Se hablará más en detalle de ellos en la sección 3.3. Estos métodos utilizan las funciones del apartado siguiente para comunicarse con los modelos.
- **Métodos de comunicación con los modelos:** corresponde a la implementación de los métodos que comunican la API con los modelos de recomendación. De

[1] "Anaconda" https://www.anaconda.com/. Accessed: 2022-03-18

esta manera, la API no llama directamente a los recomendadores, sino que pasa por una capa intermedia.

Aplicación Web. Corresponde a la implementación del front-end de la aplicación, como se vio en la sección 2.3.2. Realiza las peticiones necesarias a la API y le permite al usuario interactuar con la web.

Diagrama de secuencia

En la figura 3.2, se define el diagrama de secuencia disponible para los usuarios. Este diagrama muestra el orden de las interacciones que tienen lugar en un caso de uso típico de esta aplicación. Se muestran todas las llamadas a la API que se hacen desde la interfaz de usuario, lo que estas devuelven y en qué punto se enseñan en la pantalla del usuario.

En este diagrama, primero se accede a la web y se selecciona el usuario al cual se quiere recomendar productos. Una vez seleccionado, se verán las preferencias de este usuario y se podrá seleccionar los dos recomendadores. Cada modelo devolverá las recomendaciones que este genere.

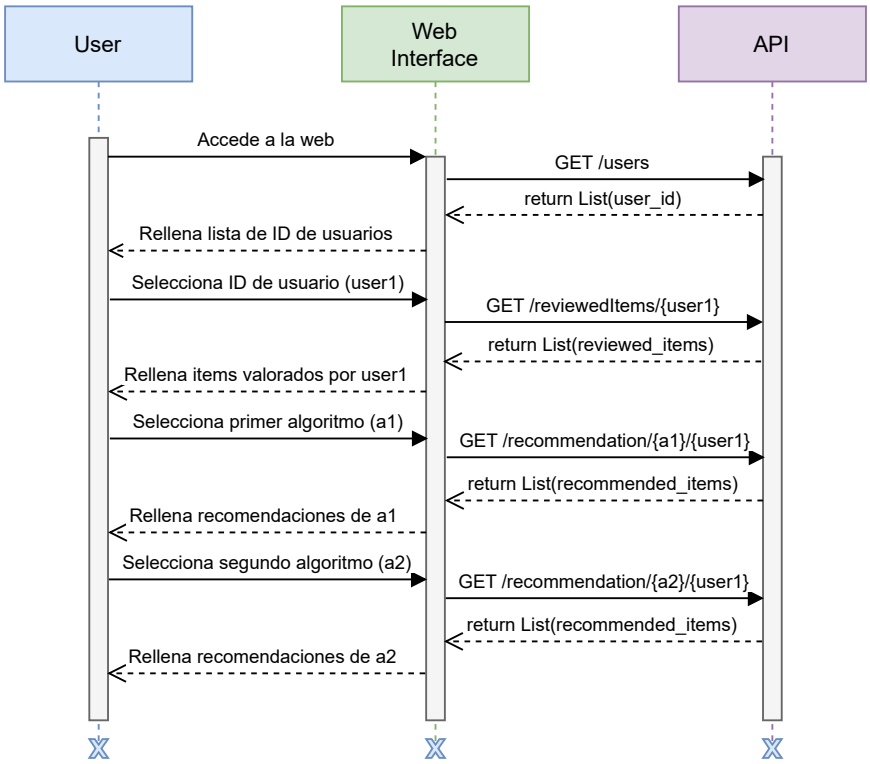


Figura 3.2: Diagrama de secuencia de la aplicación.

3.2.2. Ciclo de vida

Para el desarrollo de la aplicación se ha decidido seguir un ciclo de vida en *cascada*, ya que se trata de un sistema muy simple, cuyos requisitos han estado bien definidos desde un primer momento. Estos requisitos no han sufrido cambios en ninguna etapa del desarrollo, lo cual nos asegura que se ha escogido un ciclo de vida que se ajusta perfectamente a la naturaleza del proyecto. En la figura 3.3, podremos ver las etapas que conforman este tipo de ciclo de vida.

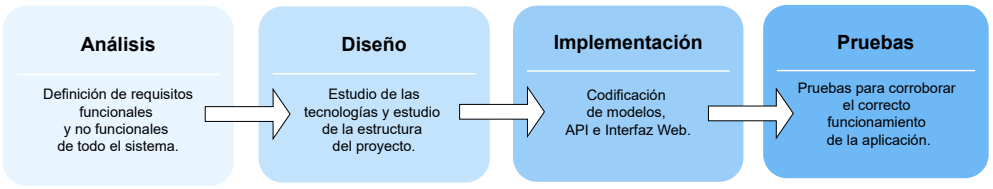


Figura 3.3: Ciclo de vida en cascada.

3.3. Implementación

En esta sección se detallará la implementación de la aplicación. Se explicará el flujo que sigue la aplicación, cómo se han desarrollado los modelos de recomendación y la aplicación web.

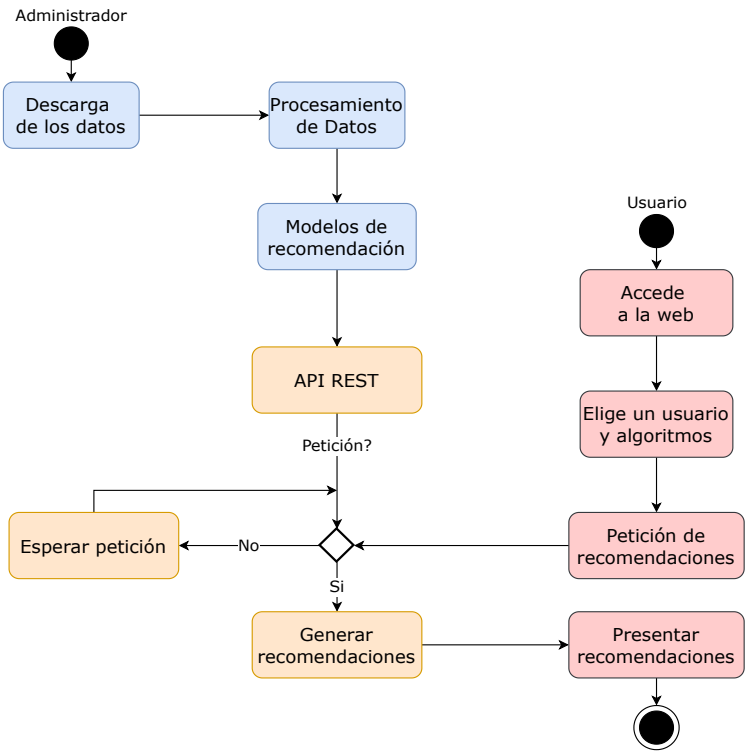


Figura 3.4: Flujo de ejecución del sistema.

3.3.1. Flujo de la aplicación

En la figura 3.4 se muestra el diagrama de flujo de la aplicación. Actualmente, el sistema tiene dos posibles puntos de partida: el del usuario que accede a la web desde la interfaz (en rojo) y el que sigue el administrador del sistema (en azul). Ambos caminos dependen del funcionamiento de la API (en naranja).

3.3.2. Modelos de recomendación

En esta sección se discutirán los distintos modelos de recomendación seleccionados, su implementación y funcionamiento.

Modelos de aprendizaje automático

En este apartado se explicarán cómo se han manejado las redes neuronales convolucionales detalladas en la sección 2.1.3. Cada red neuronal calcula los embeddings utilizando diferentes métodos, y es esta diversidad la que nos permitirá comparar los resultados entre las propias redes. En las siguientes secciones, se darán detalles de la estructura real de cada red que se ha utilizado en este proyecto, y cómo afectan al embedding resultante.

Además, estos embeddings, después de ser generados por las redes, pasarán por un modelo de vecinos próximos, el cual calculará la similitud entre los productos.

Resnet50

Para la implementación de esta red neuronal se ha utilizado la versión que proporciona la librería *Tensorflow-Keras* [2]. Esta implementación permite añadir más capas, si se necesitara. En este caso, se ha añadido una capa de Max-Pooling para reducir la dimensionalidad del resultado y poder utilizarlo como embedding de cada imagen de entrada.

En cuanto al entrenamiento de una red profunda, como lo es Resnet, es un proceso que puede consumir una gran cantidad de tiempo, debido a la estructura y profundidad que pueda poseer la propia red. Una de las razones por las cuales se ha escogido la librería Keras, es porque permite pre-entrenar las redes utilizando los pesos proporcionados por ImageNET.

En la figura 3.5, se encuentra el detalle de la red proporcionada por Tensorflow. Este modelo se compone de dos capas: la primera, sería el propio modelo resnet50, y la segunda sería una capa max pooling, que producirá los embeddings. Devolverá un embedding para cada producto, donde cada uno será un vector de dimensión 2048.

[2] "ResNet50" https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50/ResNet50. Accessed: 2022-03-13

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Model)             (None, 7, 7, 2048)        23587712
_________________________________________________________________
global_max_pooling2d_1 (Glob (None, 2048)              0
=================================================================
Total params: 23,587,712
Trainable params: 0
Non-trainable params: 23,587,712
_________________________________________________________________
```

Figura 3.5: Arquitectura real de ResNet-50 utilizada.

Inception

Nuevamente, hemos utilizado una versión pre-entrenada de la misma librería mencionada anteriormente [3].

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Model)         (None, 5, 5, 2048)        21802784
_________________________________________________________________
global_max_pooling2d_2 (Glob (None, 2048)              0
=================================================================
Total params: 21,802,784
Trainable params: 0
Non-trainable params: 21,802,784
_________________________________________________________________
```

Figura 3.6: Arquitectura real de Inception-V3 utilizada.

Como indica la figura 3.6, hemos añadido una capa de pooling al final del todo para reducir la dimensionalidad del resultado. Los embeddings resultantes ($N_{productos}$) tienen una dimensión de 2048.

VGG

Al igual que en los dos modelos anteriores, hemos utilizado una versión pre-entrenada de la misma librería [4].

Como indica la figura 3.7, hemos añadido una capa de pooling al final para reducir la dimensiona-

[3] "Inception-V3" https://www.tensorflow.org/api_docs/python/tf/keras/applications/inception_v3/InceptionV3. Accessed: 2022-03-13

[4] "VGG-16" https://www.tensorflow.org/api_docs/python/tf/keras/applications/vgg16/VGG16. Accessed: 2022-03-14

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
global_max_pooling2d_1 (Glob (None, 512)               0
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figura 3.7: Arquitectura real de VGG-16 utilizada.

lidad del resultado. En este caso, la dimensión de los embeddings cambia. Son $N_{productos}$ vectores de 512 elementos.

Estos embeddigns que se generan, únicos para cada red, se pasan como entrada al modelo de vecinos próximos (KNN). La estructura de KNN se explica a continuación.

KNN

Se ha utilizado la librería de *Scikit-Learn* para implementar este modelo [5]. Este modelo debe ser entrenado con los datos específicos que se utilizarán en el experimento, a diferencia de los modelos neuronales que son pre-entrenados con un conjunto de imágenes mucho más grande que el seleccionado para este proyecto (descrito en 4.2.1). Para ello, se utilizarán los embeddings calculados para todos los productos que generan las redes. Sabiendo esto, se necesita un modelo de KNN por red, entrenado con los embeddings correspondientes.

Para controlar los hiper-parámetros, se ha fijado el método de las distancias: mientras más cerca esté el elemento, mayor influencia tendrá en los pesos. El número de vecinos que se utiliza es 11.

Generación del ranking final

Una vez se han generado las recomendaciones para todos los productos valorados por el usuario, y hemos pasado los embeddings por el modelo de KNN, debemos ahora crear un ranking con estas recomendaciones ordenadas por la relevancia que puedan tener. Para ello, se asigna un peso a cada una de ellas, tomando en cuenta ciertos parámetros.

Estos parámetros son tres. En primer lugar, las distancias entre cada una de las imágenes de los productos recomendados y el producto inicial. En segundo lugar, el rating que le ha asignado el usuario al que se quiere recomendar, al producto inicial. Finalmente, el tiempo que ha pasado desde su valo-

[5] "KNN" https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. Accessed: 2022-03-15

ración. Estos parámetros son conjugados en una relación lineal en donde cada uno, dependiendo del valor que tome, aportará más o menos al ranking calculado por recomendación. Por ejemplo, mientras más pequeña sea la distancia entre las imágenes, mayor será lo que aporte este parámetro. A mayor rating y mientras más reciente sea la valoración, mayor valor aportarán estos parámetros.

Se asume que esta información estará disponible en el conjunto de datos de entrada a utilizar, puesto que, en la mayoría de datasets públicos de recomendación, incluyen este tipo de datos. Por lo tanto, los modelos desarrollados en este proyecto deberían poder aplicarse a cualquiera de estos.

Para poder realizar las distintas pruebas, se le asignan a cada uno de estos parámetros una relevancia que se mide en una escala del 0 al 1. Es decir, si se decide que distancia, rating y tiempo de valoración tengan los siguientes pesos en el ranking por recomendación [0.2, 0.2, 0.5], estamos diciendo que el valor que tengan cada uno de ellos va a verse perjudicado, o no, según esos pesos. De tal manera que, se puede jugar con la relevancia de cada parámetro y realizar pruebas para calcular cuál es la mejor manera de valorar las recomendaciones.

Modelos de filtrado colaborativo

En este apartado se detallará como se han manejado los modelos de recomendación no personalizados, y el modelo Singular Value Decomposition (SVD), previamente mencionado en 2.1.1 y 2.1.2.

Para la implementación del modelo SVD, se han utilizado múltiples funciones de tres librerías de Python, estas son: *Numpy* [6], *Pandas* [7] y la sección de álgebra lineal de la librería *Scipy* [8]. El desarrollo de este modelo implica seguir la ecuación definida en 2.1.2 con la ayuda de las librerías anteriores.

Como sabemos, se han escogido los algoritmos de recomendación **aleatorio** y **por popularidad** como líneas base para poder medir qué tan buenos son los modelos seleccionados. Como decisión de diseño, estos algoritmos se han implementado en base a SVD. Es decir, se generan las recomendaciones como si utilizáramos este modelo, pero en el caso del recomendador aleatorio, las desordenamos aleatoriamente, y, en el caso de popularidad, las ordenamos por las veces que el producto ha sido valorado. De esta manera, se aprovecha la estructura de SVD para implementar otros dos modelos, obteniendo el mismo resultado que si hubiesen sido implementados de manera independiente.

3.3.3. Aplicación web

En esta sección, se explicará cómo se ha implementado la aplicación web. Se detallará la estructura que posee tanto en el back-end como en el front-end.

[6] "Numpy" https://numpy.org/. Accessed: 2022-03-22

[7] "Pandas" https://pandas.pydata.org/. Accessed: 2022-03-22

[8] "Scipy Sparse LinAlg SVDs" https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.svds.html.Accessed: 2022-03-13

Back-end

Uno de los primeros problemas con los que nos encontramos al empezar a analizar cómo hacer la interfaz gráfica, fue el cómo encapsular los 3 modelos neuronales, el modelo SVD y los dos algoritmos no personalizados, de tal manera, que estos pudieran ser llamados para generar estas recomendaciones desde un cliente web. Actualmente, esto es un tema que se trata frecuentemente, ya que se está empezando a incorporar el aprendizaje automático cada vez más en aplicaciones que se usan en el día a día.

Si lo pensamos desde el punto de vista del software tradicional, los modelos actuarán como lo haría un servidor. Recibirán peticiones por parte de la interfaz gráfica, ya sea para generar recomendaciones, o elegir usuarios, entre otras acciones. Y estos deberán responder con lo que se haya pedido para poder enseñarlo al público, que en este caso, seremos nosotros.

Como se ha mencionado en la sección 2.3.1, se ha escogido la librería FastAPI de Python para implementar este servidor que se conectará a la parte front de la web. Esta API contiene todas las peticiones que se pueden hacer desde la web. Además, se comunica con todos los modelos de recomendación y con el dataset utilizado. Debido a la gran cantidad de usuarios en el dataset, la API se encarga de hacer una pre-selección de un número finito de usuarios con los cuales se podrá interactuar desde el front-end.

Debido a los tiempos de ejecución de los modelos al generar las recomendaciones, la API guarda en un fichero pickle serializable [9], que actúa como una memoria caché, todas esas recomendaciones generadas. De tal manera que, si se vuelven a pedir las mismas, para el mismo usuario, se recogen del fichero y no se ejecuta el modelo de nuevo.

Los servicios que ofrece la API son los siguientes:

- **GET /users**: devuelve los usuarios con los cuales se puede interactuar desde la web.
- **GET /reviewedItems/{user}**: devuelve los productos que ha valorado el usuario, recibiendo el ID del mismo.
- **GET /recommendation/{recommender}/{user}**: devuelve las recomendaciones utilizando el nombre del recomendador y el ID de usuario.

Front-end

En cuanto al desarrollo de la interfaz de cara al usuario, como ha sido establecido en 2.3.2, se ha utilizado la librería ReactJS. Esta librería nos permite añadir información a la web de forma dinámica y crear las peticiones a la API, gracias a que se basa en el lenguaje JavaScript.

Además, se ha utilizado una plantilla pre-hecha, de una página genérica de venta de ropa, la cual se ha ido modificando según nuestro propósito [10].

[9] "Pickle" https://docs.python.org/es/3/library/pickle.html. Accessed: 2022-03-23

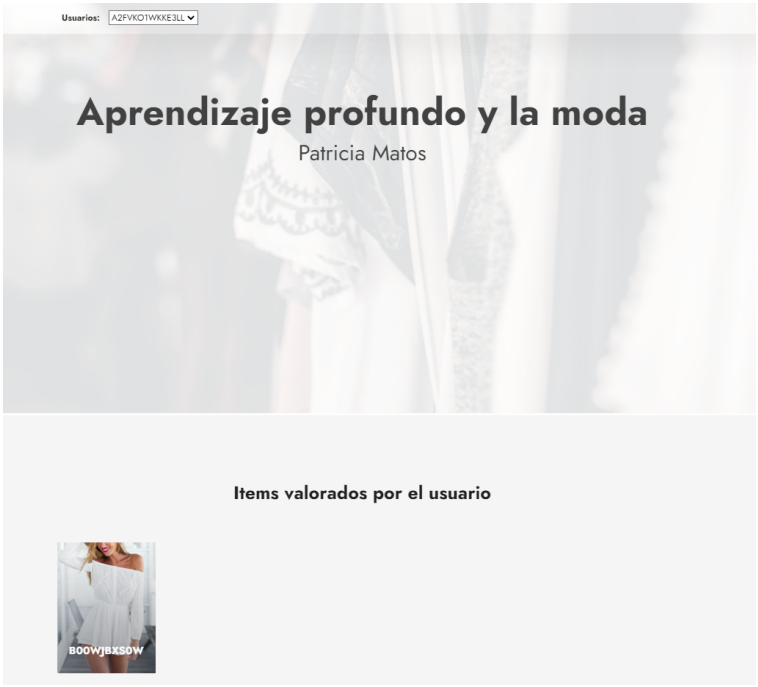
[10] "React Template E-Commerce" https://therichpost.com/reactjs-best-ecommerce-template-free-2021/. Accessed:

La página web consiste de una única interfaz principal desde donde se pueden realizar todas las acciones posibles. En la figura 3.8(a), se encuentra la portada de la página web, conformado por una barra de navegación, en la parte de arriba y el título de la página. En la barra de navegación se tiene un desplegable que permite seleccionar el usuario al cual se le recomendarán los productos.

En la figura 3.8(b), se puede ver el lugar donde se pueden comparar las recomendaciones. En la zona marcada con un recuadro naranja, se tienen dos desplegables donde se deben seleccionar los dos recomendadores que se quieran comparar. Por defecto, vienen seleccionados VGG-16 y el algoritmo aleatorio. Así mismo, en la zona azul, se tienen dos secciones donde aparecerán las recomendaciones generadas y se pueden distinguir por el título de la sección. Se muestran un total de 11 recomendaciones por cada uno de los modelos. Para ver todos los productos, se debe pulsar en las flechas que aparecen en la primera y última imagen de cada sección (marcadas con recuadros rojos).

Como se puede ver en el diagrama de secuencia de la figura 3.2, cada vez que ocurre un evento de los explicados anteriormente, la interfaz dispara una petición asíncrona hacia el servidor (AJAX), a través de las URLs que proporciona la API y, al recibirse la respuesta, se enseñan en pantalla los resultados de la consulta. El tiempo de respuesta de la API varía dependiendo de la consulta que se haga, por ejemplo, la petición para generar recomendaciones tarda mucho más que la petición de buscar los items valorados por el usuario. Más adelante, en el capítulo 4 de Pruebas y Resultados, se explicará cómo se ha manejado este problema y qué solución se ha buscado.

2022-03-16



(a) Interfaz de la página web - Primera parte.

(b) Interfaz de la página web - Segunda parte.

Figura 3.8: Interfaz gráfica.

4

Pruebas y resultados

En este capítulo, se exponen y analizan los resultados obtenidos para cada modelo de recomendación tratado en este proyecto. Para ello, se describirá el entorno donde han sido realizados los experimentos y se detallará cómo se han seleccionado los datos para estas pruebas dentro del dataset.

4.1. Entorno

A la hora de escoger el entorno para realizar las pruebas, se han tenido en cuenta varias posibilidades. En primer lugar, al estar el código desarrollado en Python, se pueden realizar las pruebas en uno o varios notebooks de jupyter. Dentro de esta posibilidad, se contemplaron dos plataformas en donde se podrían ejecutar estos notebooks: Google Colab o en el propio servidor local de jupyter. La plataforma Google Colab, aunque ofrece recursos como una GPU, se ha descartado ya que la sesión caduca al transcurrir un tiempo de inactividad demasiado pequeño, y las ejecuciones que se quieren realizar tienen una duración muy elevada. Por otro lado, ejecutar el notebook en local puede ser una buena idea, pero tendremos que tener en cuenta que el proceso de ejecutar con varios parámetros y varias veces, puede hacerse un poco pesado.

A raíz de lo anterior, ya que vamos a ejecutar en local, se ha preferido optar por la utilización de un script que recibe por pantalla todos los parámetros que se pueden cambiar de ejecución a ejecución. Además, este script nos permite mantener la estructura modular del proyecto, a diferencia del notebook.

Todas las ejecuciones se hacen dentro de un entorno preparado de Anaconda, como es descrito en los requisitos no funcionales (sección 3.1.2). En cuanto al dispositivo utilizado, en la tabla 4.1 se pueden observar las características del mismo.

Elemento	Característica
Sistema operativo	Windows 10 Home
Entorno de python	Python 3.7.9 conda
CPU	Intel(R) Core(TM) i7 @ 2.60GHz
RAM	16,0 GB

Tabla 4.1: Características del ordenador.

4.2. Experimentos

En esta sección discutiremos más a fondo los experimentos realizados para los modelos de recomendación, así como también el dataset utilizado y el manejo de sus datos.

Cabe mencionar, que estos experimentos no se realizarán utilizando la interfaz gráfica presentada, sino con un script genérico que nos permitirá configurar los parámetros necesarios, como se ha explicado en la sección anterior.

4.2.1. Amazon Fashion Dataset

En primer lugar, presentaremos el conjunto de datos que hemos utilizado para evaluar el problema de recomendación. Este dataset fue publicado por Amazon, la primera vez, en 2014, y actualizado en 2018. En este trabajo se ha utilizado la versión actualizada del 2018 [1].

Contiene datos de productos (descripción, imágenes, precio, marca...) y de reseñas (*reviews*) de usuarios (valoración, comentarios, votos, ...) desde el 2014 al 2018, en archivos separados, por ende, utilizaremos dos datasets distintos, pero que apuntan a los mismos datos de partida.

Estos datasets, al ser de grandes dimensiones, vienen en distintas presentaciones, que se ajustan a las necesidades de cada uno. Los investigadores ofrecen unos datasets más pequeños, divididos por distintas temáticas y que se corresponden con las secciones del sitio de Amazon, como por ejemplo moda, libros, electrónica, entre otros. Esto nos resulta muy útil ya que únicamente necesitaríamos los productos de moda.

En cuanto a las valoraciones, también ofrecen datasets en donde los investigadores se han asegurado de que cada producto y usuario haya sido o haya valorado un número específico de veces. En este caso, se ha escogido uno en donde los productos y los usuarios tienen al menos 5 valoraciones. Esto reduce considerablemente el impacto que el *cold start* pueda ocasionar, ya que no habrá productos nuevos sin valoración, ni usuarios que no hayan valorado ningún producto.

Cada producto posee un identificador, título, precio, descripción, imágenes en distinta resolución, productos relacionados, categoría, entre otros muchos atributos. Después de realizar un análisis de

[1] N. Jianmo, "Amazon review data (2018)." http://deepyeti.ucsd.edu/jianmo/amazon/. Accessed: 2021-11-08

estos, se ha decidido que los atributos que más se ajustan a las necesidades del problema son los de identificador (**asin**), título (**title**) y la imagen del producto (**imageURLHighRes**) [19].

Finalmente, cada review posee el identificador del producto que se está valorando, el identificador y nombre del usuario que hace la valoración, el rating que se le ha puesto, comentarios, la fecha y hora en la que se valoró dicho producto, entre otros. Al igual que en el caso anterior, hemos decidido quedarnos con los atributos de identificador de producto (**asin**), identificador de usuario (**reviewerID**), rating (**overall**) y la fecha y hora de la valoración (**unixReviewTime**) [19].

Para los experimentos, se ha dividido el dataset en entrenamiento y test, con una proporción configurable, aunque por defecto se ha dejado 80 % train y 20 % test. De tal manera, que el usuario al cual queremos recomendarle productos, es escogido de la parte de test, y la información utilizada del mismo, de la parte de entrenamiento. Si este usuario no aparece en la parte de entrenamiento, no podremos generar recomendaciones para el mismo.

En la figura 4.1, se pueden ver las características de este dataset con respecto a la distribución de sus valoraciones o ratings.

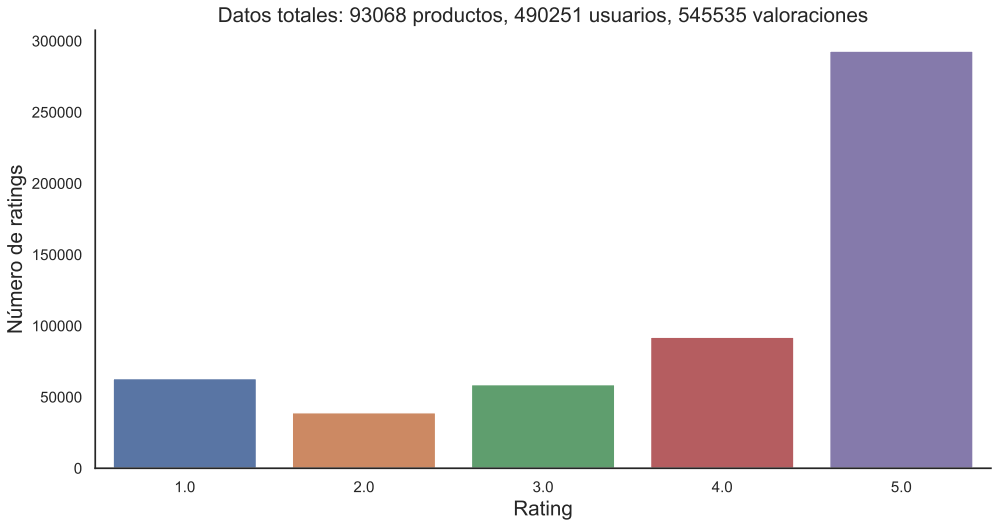


Figura 4.1: Datos totales.

4.2.2. Análisis de tiempos de ejecución

Como se ha visto en la sección 4.2.1, el dataset contiene una cantidad elevada de usuarios. El dispositivo en el que se quieren realizar los experimentos no dispone de los recursos suficientes para manejar dicha cantidad. Por lo tanto, se debe escoger una muestra finita más pequeña de usuarios, que sea acorde a los recursos disponibles.

En la tabla 4.2 se muestran las pruebas realizadas para escoger esa muestra. En ella, se le han asignado un valor fijo a todos los hiper-parámetros, excepto al número de usuarios, que es el que

interesa ir variando.

Se ha seleccionado la red neuronal con menor tiempo de ejecución, Inception V3, y se ha probado con 4 cantidades distintas de usuarios, para evaluar los tiempos de ejecución. Además, se debe tomar en cuenta que las redes neuronales tardan mucho menos en ejecutar, en comparación al modelo de filtrado colaborativo SVD.

Como fue mencionado en la sección 3.3.2, en la columna de pesos se muestran tres elementos: el primero hace referencia a las distancias, el segundo al rating del producto y el tercero al la cantidad de tiempo que ha pasado desde la última valoración.

Número de usuarios	Cut-off	Threshold	Número de vecinos	Tamaño del conjunto de test	Pesos	Tiempo de ejecución (min)
4000	5	[1.5]	11	0.2	[1, 1, 1]	85
3000	5	[1.5]	11	0.2	[1, 1, 1]	61
2000	5	[1.5]	11	0.2	[1, 1, 1]	40
1000	5	[1.5]	11	0.2	[1, 1, 1]	25

Tabla 4.2: Tabla de tiempos de ejecución.

En general, los tiempos de ejecución no son demasiado elevados. Con 4000 usuarios, las recomendaciones se generan en aproximadamente 1 hora y 20 minutos. Si se sigue reduciendo el número de usuarios hasta llegar a 1000, este tiempo se va reduciendo, como era de esperar.

Aún tomando en cuenta que la opción con más usuarios no tiene un tiempo de ejecución excesivamente elevado, finalmente se ha seleccionado la muestra de 1000 usuarios. De tal manera que, se puedan realizar una mayor cantidad de experimentos, y así poder variar los hiper-parámetros en mayor medida sin impactar en exceso en los tiempos de desarrollo de este trabajo.

4.2.3. Estudio de hiper-parámetros de Inception V3

Con el objetivo de estudiar cómo afectan los distintos hiper-parámetros a la generación de las recomendaciones, se ha utilizado nuevamente la red Inception V3 para realizar los primeros experimentos, y así seleccionar algunos de ellos y poder trasladarlos al resto de redes neuronales.

En la tabla 4.3, se muestran los seis experimentos realizados. En los primeros tres (E1, E2 y E3), se varía el número de vecinos: 5, 11 y 22. En los últimos tres (E4, E5 y E6), se varían los pesos dejando siempre el primero de ellos en 1, ya que hace referencia a las distancias calculadas a partir de la red neuronal.

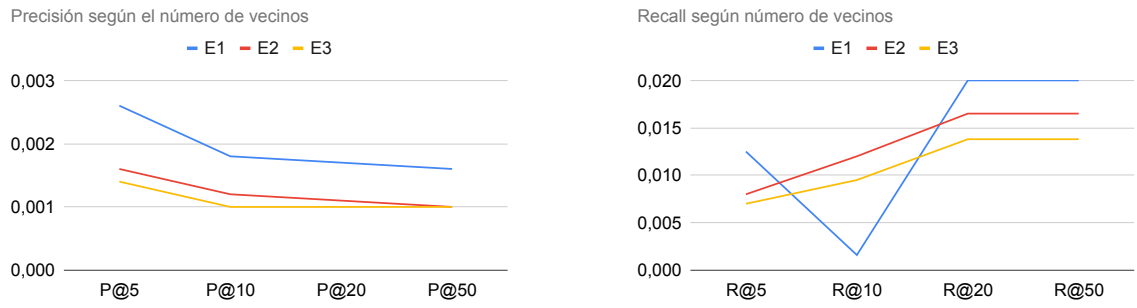
	Número de usuarios	Cut-off	Threshold	Número de vecinos	Tamaño del conjunto de test	Pesos
E1	1000	5, 10, 20, 50	1.5	5	0.2	[1, 1, 1]
E2	1000	5, 10, 20, 50	1.5	11	0.2	[1, 1, 1]
E3	1000	5, 10, 20, 50	1.5	22	0.2	[1, 1, 1]
E4	1000	5, 10, 20, 50	1.5	11	0.2	[1, 1, 0]
E5	1000	5, 10, 20, 50	1.5	11	0.2	[1, 0, 1]
E6	1000	5, 10, 20, 50	1.5	11	0.2	[1, 0, 0]

Tabla 4.3: Hiper-parámetros con Inception V3.

Para facilitar el proceso de comparación de los resultados, hemos dividido los dos grupos de experimentos en dos tablas diferentes. Los resultados para los experimentos E1-3, se encuentran en la tabla 4.4, y los resultados para E4-6 se encuentran en la tabla 4.5. Además, se disponen de gráficas explicativas para cada uno de los grupos, según la métrica.

	P@5	P@10	P@20	P@50	R@5	R@10	R@20	R@50
E1	**0.0026**	**0.0018**	**0.0017**	**0.0016**	**0.0125**	0.0016	**0.0200**	**0.0200**
E2	0.0016	0.0012	0.0011	0.0010	0.0080	**0.0120**	0.0165	0.0165
E3	0.0014	0.0010	0.0010	0.0010	0.0070	0.0095	0.0138	0.0138

Tabla 4.4: Métricas según número de vecinos. En negrita se indica el mejor resultado para cada métrica.



(a) Precisión

(b) Recall

Figura 4.2: Gráficas según número de vecinos.

En la figura 4.2(a), se muestra el progreso de la métrica de Precisión@k en los experimentos E1-3. En general, el experimento que más precisión consigue es el primero, E1. En este experimento, se calculan las distancias de las imágenes con los 5 vecinos más próximos.

A medida que se aumenta el valor del cutoff, se puede observar cómo, para todos los experimentos, la precisión disminuye. Entre P@5 y P@10, disminuye de manera más pronunciada, y entre P@10 y P@50, se mantiene prácticamente en el mismo valor.

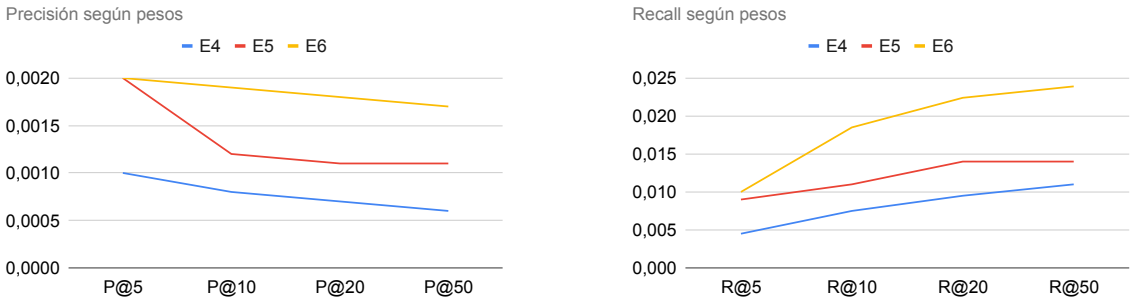
En la figura 4.2(b), que corresponde a la evolución de Recall@k, se puede observar un comporta-

miento más brusco para el experimento 1. Mientras que, para los experimentos 2 y 3, los cambios en la métrica son prácticamente los mismos.

De ambas gráficas se puede concluir que el primer experimento (E1) obtiene mejores resultados, con diferencia, que los dos restantes. En otras palabras, la red genera mejores recomendaciones con 5 vecinos que con 11 y 22. Esto ocurre porque, a mayor cantidad de vecinos, habrá más focos de comparación, separados por distancias más grandes. Por ende, se genera un infrajuste en los resultados al introducirse imágenes más distintas, que generan más ruido en las recomendaciones.

	P@5	P@10	P@20	P@50	R@5	R@10	R@20	R@50
E4	0.0010	0.0008	0.0007	0.0006	0.0045	0.0075	0.0095	0.0110
E5	**0.0020**	0.0012	0.0011	0.0011	0.0090	0.0110	0.0140	0.0140
E6	**0.0020**	**0.0019**	**0.0018**	**0.0017**	**0.0100**	**0.0185**	**0.0224**	**0.0239**

Tabla 4.5: Métricas según pesos.



(a) Precisión

(b) Recall

Figura 4.3: Gráficas según pesos.

En las figuras 4.3(a) y 4.3(b), se encuentran las gráficas para los experimentos en donde se varían la importancia de cada uno de los factores que contribuyen a la generación de los pesos.

Se puede observar, que el experimento con mejores resultados es el E6. En este experimento, únicamente se toman en cuenta las distancias de las imágenes para generar los pesos. En otras palabras, obtendremos mejores recomendaciones para un usuario, si el factor con mayor importancia en la generación de los pesos de cada producto, son las distancias calculadas por las redes.

4.2.4. Comparativa de eficacia de distintos modelos neuronales

Según los resultados obtenidos en la sección 4.2.3, se han seleccionado 4 experimentos para los dos modelos neuronales restantes: VGG y ResNet. Se han escogido los dos valores con mejor resultado para el número de vecinos y las dos mejores combinaciones para los pesos. Estos experimentos se describen en la tabla 4.6.

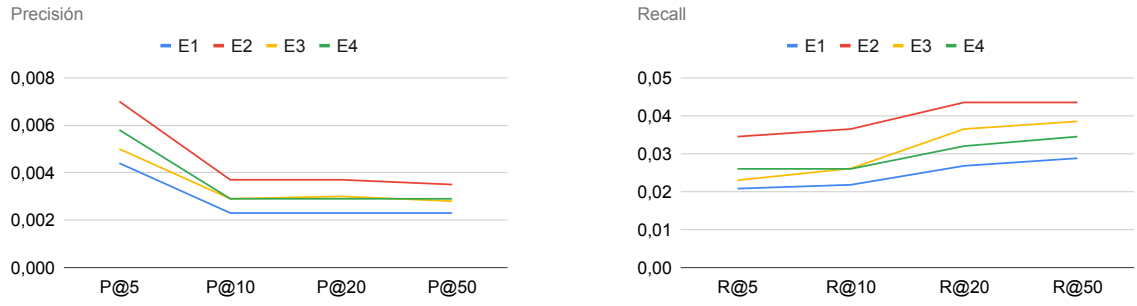
	Número de usuarios	Cut-off	Threshold	Número de vecinos	Tamaño del conjunto de test	Pesos
E1	1000	5, 10, 20, 50	[1.5]	5	0.2	[1, 1, 1]
E2	1000	5, 10, 20, 50	[1.5]	5	0.2	[1, 0, 0]
E3	1000	5, 10, 20, 50	[1.5]	11	0.2	[1, 1, 1]
E4	1000	5, 10, 20, 50	[1.5]	11	0.2	[1, 0, 0]

Tabla 4.6: Experimentos realizados con VGG y Resnet.

En la tabla 4.7, se muestran los valores de las métricas de estos experimentos para la red **VGG 16**.

	P@5	P@10	P@20	P@50	R@5	R@10	R@20	R@50
E1	0,0044	0,0023	0,0023	0,0023	0,0208	0,0218	0,0268	0,0288
E2	**0,0070**	**0,0037**	**0,0037**	**0,0035**	**0,0345**	**0,0365**	**0,0435**	**0,0435**
E3	0,0050	0,0029	0,0030	0,0028	0,0230	0,0261	0,0365	0,0385
E4	0,0058	0,0029	0,0029	0,0029	0,0260	0,0260	0,0320	0,0345

Tabla 4.7: Métricas de VGG.



(a) Precisión

(b) Recall

Figura 4.4: Evolución de las métricas VGG.

Como se puede observar en las gráficas de la figura 4.4, el segundo experimento consigue generar recomendaciones más acertadas. Esto no sorprende, ya que este experimento reúne las características que mejores resultados tuvieron con la red Inception V3: 5 vecinos más próximos y que solo se evalúen las distancias en la generación de los pesos del ranking.

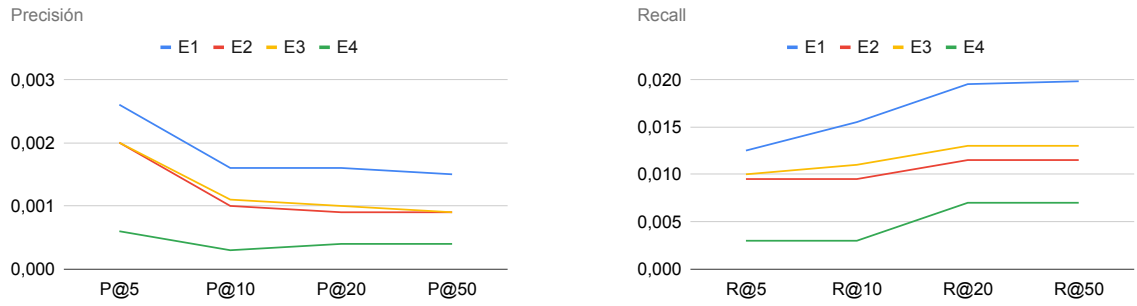
Los valores de precisión de la gráfica 4.4(a), disminuyen drásticamente en la métrica P@10 y, se mantienen constantes para el resto. Consiguiendo, en todos los experimentos, su máximo en P@5. En

el caso del Recall (gráfica 4.4(b)), el E2 obtiene también los valores más altos. En general, el recall de todos los experimentos se mantiene constante en todas las métricas.

En el caso de la red **ResNet**, se muestran los resultados en la tabla 4.8.

	P@5	P@10	P@20	P@50	R@5	R@10	R@20	R@50
E1	**0,0026**	**0,0016**	**0,0016**	**0,0015**	**0,0125**	**0,0155**	**0,0195**	**0,0198**
E2	0,0020	0,0010	0,0009	0,0009	0,0095	0,0095	0,0115	0,0115
E3	0,0020	0,0011	0,0010	0,0009	0,0100	0,0110	0,0130	0,0130
E4	0,0006	0,0003	0,0004	0,0004	0,0030	0,0030	0,0070	0,0070

Tabla 4.8: Métricas de ResNet.



(a) Precisión

(b) Recall

Figura 4.5: Evolución de las métricas ResNet.

Como se puede detallar en las gráficas de la figura 4.5, en esta red el E2 no consigue tan buenos resultados como en VGG. En este caso el experimento con mejor precisión y recall es el E1. Este experimento consiste en 5 vecinos próximos y tomando todos los factores por igual, en consideración, para los pesos del ranking. Esto podría deberse a que los embeddings generados por la red no son tan buenos como los de Inception V3 o VGG, y al tomar en cuenta el rating y el tiempo de valoración, la precisión del resultado se ve beneficiada.

En la gráfica de precisión 4.5(a), se puede observar cómo para los cuatro experimentos los valores se reducen en P@10 y, luego, se mantienen constantes. Por otro lado, en la gráfica 4.5(b), los valores van incrementando hasta R@20, donde empiezan a ser constantes.

4.2.5. Discusión y comparativa con algoritmos de filtrado colaborativo

Los experimentos para los modelos basados en filtrado colaborativo dependen de factores diferentes con respecto a los modelos neuronales. Para comprobar el funcionamiento del modelo SVD, se variará la cantidad de vectores que utilizará el mismo. En la tabla 4.9, se describen estos experimentos.

Una vez realizados los experimentos, se han obtenido los resultados mostrados en la tabla 4.10.

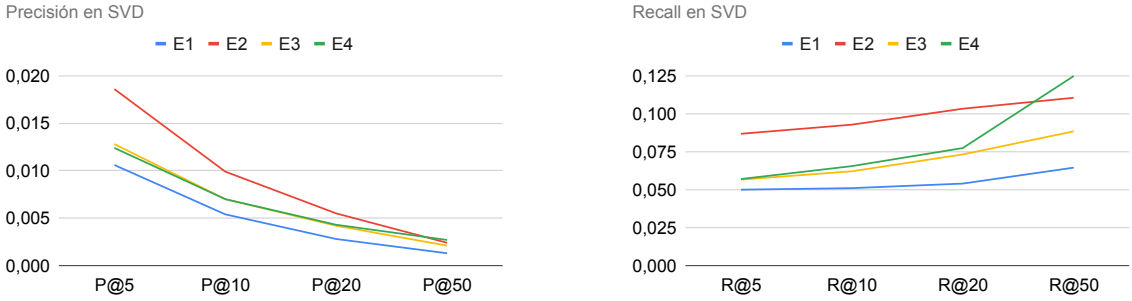
	Número de usuarios	Cut-off	Threshold	Número de vectores singulares	Tamaño del conjunto de test
E1	1000	5, 10, 20, 50	[1.5]	5	0.2
E2	1000	5, 10, 20, 50	[1.5]	11	0.2
E3	1000	5, 10, 20, 50	[1.5]	22	0.2
E4	1000	5, 10, 20, 50	[1.5]	50	0.2

Tabla 4.9: Experimentos para los modelos de filtrado colaborativo.

	P@5	P@10	P@20	P@50	R@5	R@10	R@20	R@50
E1	0,0106	0,0054	0,0028	0,0013	0,0500	0,0510	0,0540	0,0645
E2	**0,0186**	**0,0099**	**0,0055**	0,0024	**0,0868**	**0,0928**	**0,1033**	0,1105
E3	0,0128	0,0070	0,0042	0,0021	0,0566	0,0621	0,0732	0,0884
E4	0,0124	0,0070	0,0043	**0,0027**	0,0570	0,0655	0,0774	**0,1248**

Tabla 4.10: Métricas de SVD.

Para los valores que se observan en las gráficas 4.6(a) y 4.6(b), se destaca el E2 por conseguir las métricas más altas, con respecto al resto. En 4.6(a), es importante mencionar que en P@50, todos los valores obtienen un valor bastante bajo. Por otro lado, en 4.6(b), cabe mencionar cómo el experimento 4 logra superar al experimento 2 en R@50.



(a) Precisión

(b) Recall

Figura 4.6: Evolución de las métricas SVD.

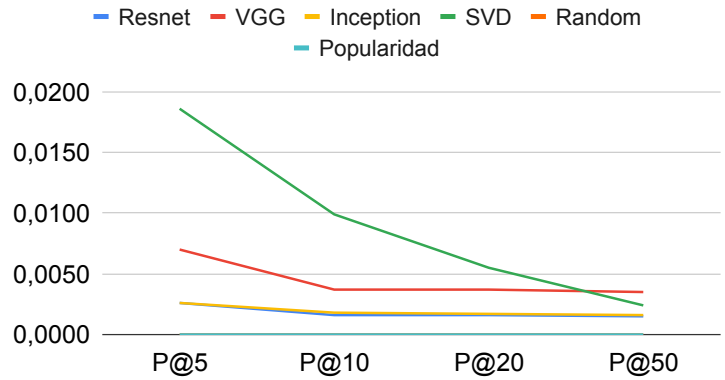
Para poder realizar una comparativa, se debe tomar en cuenta también los resultados de los algoritmos no personalizados. Para ello, se muestran en la tabla 4.11, un resumen con los experimentos con mejor resultado de cada modelo, incluyendo el algoritmo aleatorio y de popularidad, y en la figura 4.7, la representación de los mismos.

No es difícil observar que los resultados de las métricas son bastante bajos en general para todos los recomendadores. Esto sugiere que este problema de recomendación, con los datos utilizados, es difícil de resolver. Esta hipótesis es respaldada por los algoritmos que actúan como líneas base, que no consiguen acertar ninguna recomendación, y por ende, todas sus métricas son nulas. Estos resultados podrían mejorar si se incrementase el tamaño de la muestra de usuarios.

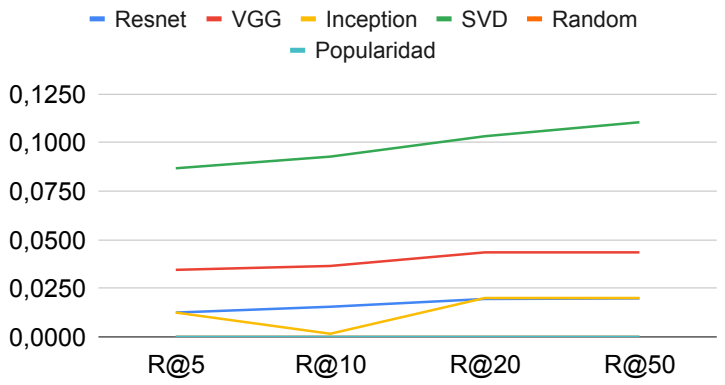
	P@5	P@10	P@20	P@50	R@5	R@10	R@20	R@50
Random	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
Popularidad	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
Inception	0,0026	0,0018	0,0017	0,0016	0,0125	0,0016	0,0200	0,0200
VGG	0,0070	0,0037	0,0037	**0,0035**	0,0345	0,0365	0,0435	0,0435
Resnet	0,0026	0,0016	0,0016	0,0015	0,0125	0,0155	0,0195	0,0198
SVD	**0,0186**	**0,0099**	**0,0055**	0,0024	**0,0868**	**0,0928**	**0,1033**	**0,1105**

Tabla 4.11: Resumen de métricas.

Si se valoran los resultados de las gráficas 4.7(a) y 4.7(b), se puede concluir que SVD es el modelo con mejores resultados, y por ende, es el modelo cuyas recomendaciones se acercan más a las preferencias de los usuarios en la muestra. El segundo mejor modelo es la red neuronal VGG 16 y, por último, la red Inception V3.



(a) Precisión



(b) Recall

Figura 4.7: Gráfica del resumen de las métricas.

En líneas generales, aunque los resultados obtenidos para los modelos neuronales no hayan sido elevados, podemos concluir que la red neuronal que consigue un mejor comportamiento ante los experimentos, es la red VGG 16. Incluso su resultado más bajo en precisión, es mayor al cualquier resultado de las redes ResNet e Inception V3.

5

Conclusiones y trabajo futuro

En esta sección se plantearán las conclusiones a las que se ha llegado durante el desarrollo de este proyecto, así como también, el trabajo futuro a realizar para ampliar este trabajo.

5.1. Conclusiones

Los sistemas de recomendación son cada vez más frecuentes en nuestro día a día. Cada vez que se accede a un servicio, ya sea en Internet o en una aplicación, hacemos uso de esta tecnología para elegir esos productos que nos ofrecen. En este proyecto se han tratado, en especial, aquellos sistemas de recomendación centrados en el ámbito de la moda, y que tanto se utilizan en el comercio electrónico.

Tras analizar a fondo estos sistemas de recomendación, y las distintas maneras en que se pueden implementar, se decidió optar por tres modelos neuronales y un modelo de filtrado colaborativo. El modelo de filtrado colaborativo es una de las técnicas más utilizadas, que se basa únicamente en las valoraciones del usuario previas a la recomendación. En base a lo anterior, se quiso investigar el camino de las redes neuronales, escogiendo una arquitectura en donde no solo se tomase en cuenta esa valoración, sino la similitud del producto que se quiere recomendar con los que le han gustado al usuario en otras ocasiones.

Las redes neuronales permiten analizar y explotar las imágenes de estos productos, algo que los modelos clásicos basados en valoraciones no pueden hacer. Sin embargo, como se ha podido observar en los experimentos, estas redes consiguen peores resultados que algoritmos más sencillos de filtrado colaborativo. Aún queda mucho que investigar acerca de las redes convolucionales, pero no es descabellado pensar, que en un futuro no muy lejano, estos modelos consigan ser mucho más precisos.

Al utilizar imágenes para generar las recomendaciones, también se ha creado una interfaz gráfica que nos permite observar de una manera más directa dichas recomendaciones, y sus diferencias entre los modelos seleccionados.

Finalmente, todo el código desarrollado se puede encontrar en el siguiente enlace:

`https://github.com/patriciamatosm/TFG.git`

5.2. Trabajo futuro

A lo largo de la realización de este proyecto, se han encontrando diversas áreas y posibles mejoras al mismo, pero por falta de tiempo y de recursos, no se han explorado. Estas áreas, podrían ser el fruto de una futura ampliación de este TFG.

Una de las primeras mejoras encontradas sería aumentar el número de usuarios en la muestra que se utiliza en cada experimento. De esta manera, podríamos estudiar de una manera más exacta los resultados. A raíz de ello, también se podrían realizar una mayor cantidad de experimentos, y así, hacer un estudio más exhaustivo de los hiper-parámetros para conseguir la combinación que maximice las métricas de los modelos.

En segundo lugar, se podrían utilizar más tipos de métricas para evaluar los modelos, como por ejemplo, Mean Reciprocal Rank (MRR) o Discounted Cumulative Gain (DCG) . De esta forma, podríamos obtener más información acerca del comportamiento de los modelos.

Por otro lado, puede ser interesante la implementación de modelos neuronales que no estén pre-entrenados, y que se entrenen con la partición de entrenamiento de los datos utilizados. El entrenamiento de una red neuronal convolucional con el dataset escogido, implicaría poseer un dispositivo con los recursos necesarios y el tiempo suficiente para poder realizarlo. Adicionalmente, podría probarse a entrenar solo ciertas capas de la red, según convenga, dejando el resto con el entrenamiento previo.

Finalmente, otra posible ampliación sería corroborar el funcionamiento de los modelos con problemas específicos de la recomendación de moda, como por ejemplo Fill in the blanks (FITB), en donde se podrían generar recomendaciones a partir de un outfit incompleto, tomando en cuenta el item faltante y el resto de factores. Otro ejemplo de tarea específica del dominio de la moda sería la generación de un outfit desde cero. De esta forma, se mostraría el gran potencial que tiene el aprendizaje profundo en los problemas de recomendación, haciendo especial hincapié en la recomendación de moda.

Bibliografía

[1] F. Ricci, L. Rokach, and B. Shapira, "Recommender systems: Introduction and challenges," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 1–34, Springer, 2015.

[2] L. Terveen and W. Hill, "Beyond recommender systems: Helping people help each other," 02 2001. Descargar.

[3] H. Polat and W. Du, "Svd-based collaborative filtering with privacy," in *Proceedings of the 2005 ACM Symposium on Applied Computing*, SAC '05, (New York, NY, USA), p. 791–795, Association for Computing Machinery, 2005. Descargar.

[4] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Comput. Surv.*, vol. 52, feb 2019. Descargar.

[5] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," vol. abs/1511.08458, 2015. Descargar.

[6] G. Guo, H. Wang, D. A. Bell, Y. Bi, and K. Greer, "KNN model-based approach in classification," in *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003* (R. Meersman, Z. Tari, and D. C. Schmidt, eds.), vol. 2888 of *Lecture Notes in Computer Science*, pp. 986–996, Springer, 2003.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778, IEEE Computer Society, 2016.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pp. 1–9, IEEE Computer Society, 2015.

[9] M. MahdianPari, B. Salehi, M. Rezaee, F. Mohammadimanesh, and Y. Zhang, "Very deep convolutional neural networks for complex land cover mapping using multispectral remote sensing imagery," *Remote. Sens.*, vol. 10, no. 7, p. 1119, 2018.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[11] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 5–53, 2004.

[12] I. Kerenidis and A. Prakash, "Quantum recommendation systems," in *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA* (C. H. Papadi-

mitriou, ed.), vol. 67 of *LIPIcs*, pp. 49:1–49:21, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[13] M. Deac-Petrusel, "A lexicon-based collaborative filtering approach for recommendation systems," in *Proceedings of the 14th International Conference on Agents and Artificial Intelligence, ICAART 2022, Volume 3, Online Streaming, February 3-5, 2022* (A. P. Rocha, L. Steels, and H. J. van den Herik, eds.), pp. 203–210, SCITEPRESS, 2022.

[14] W. Shafqat and Y. Byun, "A hybrid gan-based approach to solve imbalanced data problem in recommendation systems," *IEEE Access*, vol. 10, pp. 11036–11047, 2022.

[15] W. Chen, P. Huang, J. Xu, X. Guo, C. Guo, F. Sun, C. Li, A. Pfadler, H. Zhao, and B. Zhao, "POG: personalized outfit generation for fashion recommendation at alibaba ifashion," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019* (A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis, eds.), pp. 2662–2670, ACM, 2019.

[16] X. Han, Z. Wu, Y. Jiang, and L. S. Davis, "Learning fashion compatibility with bidirectional lstms," in *Proceedings of the 2017 ACM on Multimedia Conference, MM 2017, Mountain View, CA, USA, October 23-27, 2017* (Q. Liu, R. Lienhart, H. Wang, S. K. Chen, S. Boll, Y. P. Chen, G. Friedland, J. Li, and S. Yan, eds.), pp. 1078–1086, ACM, 2017.

[17] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, "Methods and metrics for cold-start recommendations," in *SIGIR 2002: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 11-15, 2002, Tampere, Finland* (K. Järvelin, M. Beaulieu, R. A. Baeza-Yates, and S. Myaeng, eds.), pp. 253–260, ACM, 2002.

[18] E. Morales Agostinho, "Sistemas de recomendación de noticias basados en aprendizaje profundo," B.S. thesis, Escuela Politécnica Superior, Universidad Autónoma de Madrid, 2021.

[19] J. Ni, J. Li, and J. J. McAuley, "Justifying recommendations using distantly-labeled reviews and fine-grained aspects," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019* (K. Inui, J. Jiang, V. Ng, and X. Wan, eds.), pp. 188–197, Association for Computational Linguistics, 2019.

Acrónimos

AJAX Asynchronous JavaScript And XML.

API Application Programming Interface.

CNN Convolutional Neural Network.

DCG Discounted Cumulative Gain.

FITB Fill in the blanks.

KNN K Nearest Neighbors.

MRR Mean Reciprocal Rank.

ReLu Rectified Linear Unit.

ResNet Residual Network.

REST Representational State Transfer.

SQL Structured Query Language.

SVD Singular Value Decomposition.

TFG Trabajo de Fin de Grado.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID