Facultad de ciencias

Sistemas de recomendacion en un marketplace de APIs

Recommendation systems in a marketplace of APIs

Trabajo Fin de Grado para acceder al
Grado en Matemáticas

Autor: David Robledo Imhof
Director: Santos Bringas Tejero
Codirector: Francisco Santos Leal

7 de julio de 2023

Resumen

Los marketplaces de APIs son plataformas en línea que permiten a los desarrolladores acceder a una amplia variedad de APIs de terceros, y utilizarlas para integrar funcionalidades en sus propias aplicaciones y servicios.

Estos marketplaces son importantes porque permiten a los desarrolladores ahorrar tiempo y recursos al evitar tener que desarrollar sus propias APIs desde cero, y les permiten acceder a una amplia variedad de funcionalidades para enriquecer sus propias aplicaciones. Estas plataformas también son importantes para las empresas que proporcionan las APIs, ya que les permiten llegar a una audiencia más amplia y promover su uso.

La introducción de filtros colaborativos en los marketplaces de APIs es útil porque ayuda a los desarrolladores a encontrar las APIs adecuadas para sus necesidades. Los filtros colaborativos utilizan el poder del crowdsourcing para permitir a los desarrolladores ver las calificaciones y reseñas de otras personas que han utilizado la API, y así obtener una idea más clara de su calidad y utilidad. Además, se podrán establecer relaciones entre los distintos perfiles de usuarios, ofreciendo así una experiencia más personalizada. Esto puede ayudar a los consumidores a tomar decisiones más informadas sobre qué APIs utilizar en sus proyectos, y puede aumentar la satisfacción y la eficacia general de su trabajo.

Este trabajo se centra en el estudio y análisis de los sistemas de recomendación más populares en la actualidad, utilizando técnicas como los filtros colaborativos y su aplicación combinada con la descomposicion en valores singulares de las matrices de interacciones. Además, se llevará a cabo su implementación en un marketplace de APIs con el objetivo de comprender y evaluar su utilidad en este contexto específico.

Palabras clave

Sistemas de recomendación, Filtros colaborativos, Descomposición en valores singulares, API, Marketplace,

Abstract

API marketplaces are online platforms that enable developers to access a wide variety of third-party APIs and utilize them to integrate functionalities into their own applications and services.

These marketplaces are important because they allow developers to save time and resources by avoiding the need to develop their own APIs from scratch, and they provide access to a diverse range of functionalities to enrich their applications. API marketplaces are also crucial for API providers, as they enable them to reach a broader audience and promote API usage.

The introduction of collaborative filtering in API marketplaces is beneficial because it helps developers find the right APIs for their needs. Collaborative filtering harnesses the power of crowdsourcing, allowing developers to see ratings and reviews from other individuals who have used the API, providing them with a clearer understanding of its quality and usefulness. Additionally, establishing relationships between user profiles offers a more personalized experience. This can assist consumers in making informed decisions about which APIs to use in their projects, leading to increased satisfaction and overall effectiveness in their work.

This work focuses on studying and analyzing the most popular recommendation systems currently in use, using techniques such as collaborative filtering and singular value decomposition of the interaction matrices. Furthermore, their implementation will be carried out in an API marketplace to comprehend and evaluate their utility in this specific context.

Class words

Recommendation Systems, Collaborative Filtering, Singular value decomposition, API, Marketplace

Índice general

1. Introducción **7**
1.1. Qué es una API . . . 7
1.2. Qué es un marketplace . . . 10
1.3. Sistemas de recomendación . . . 11
1.4. Este trabajo . . . 12

2. Filtros colaborativos **13**
2.1. Esquema general del filtrado colaborativo . . . 13
2.2. Formalización del problema . . . 14
2.2.1. Métrica de similitud . . . 14
2.2.2. Búsqueda de vecinos . . . 15
2.2.3. Crear tabla de recomendaciones y realizar recomendación . . . 15
2.3. Ejemplo de recomendación . . . 16
2.3.1. Definición de conjuntos . . . 16
2.3.2. Selección de la métrica . . . 17
2.3.3. Busqueda de vecinos . . . 19
2.3.4. Realizar una recomendación . . . 20
2.4. Análisis de resultados . . . 20

3. Descomposición en valores singulares (SVD) **21**
3.1. Conceptos básicos de SVD . . . 21
3.2. Demostración SVD . . . 22
3.3. Cálculo y propiedades de las matrices de SVD . . . 26
3.3.1. Matriz V . . . 26
3.3.2. Matriz Σ . . . 27
3.3.3. Matriz U . . . 27
3.3.4. Propiedades . . . 28
3.4. Ejemplo de descomposición SVD . . . 28
3.4.1. Definición de los conjuntos . . . 28
3.4.2. Cálculo de V . . . 29
3.4.3. Cálculo de U . . . 33
3.4.4. Cálculo de Σ . . . 35
3.5. Realizacion de una recomendación . . . 35
3.5.1. Búsqueda de una métrica . . . 35
3.5.2. Búsqueda de vecinos . . . 38

3.5.3. Recomendación 38

4. Implementación en el portal **39**
4.1. Conceptos iniciales 39
4.1.1. Qué es un CMS 39
4.1.2. Qué es Drupal 39
4.2. Entidades creadas 39
4.2.1. Contenidos 40
4.2.2. Taxonomías. 40
4.3. Métodos para crear APIs 41
4.3.1. Clase ApiProcessor 41
4.3.2. Método newApi 41
4.3.3. Método taxonomyComplete 41
4.4. Creación de una API desde la web 42
4.4.1. Uploadyaml.info.yml 42
4.4.2. UploadYamlForm.php 42
4.4.3. UploadYamlBlock.php 42
4.5. Crear una API mediante instrucciones HTTP 43
4.6. Implementación de filtro colaborativo con SVD 43
4.6.1. Preparación de los datos 44
4.6.2. Calcular descomposición SVD 45
4.6.3. Búsqueda de los vecinos 45
4.6.4. Recomendación final 45

5. Conclusiones y lineas futuras **47**
5.1. Sistemas de recomendación 47
5.2. Implementación en el portal 48

Bibliografía **50**

Apéndices **52**

A. Módulo Upload Yaml **52**
A.1. Upload_yaml.info.yml 52
A.2. UploadYamlBlock.php 52
A.3. UploadYamlForm.php 54

B. Métodos de recomendación **56**
B.1. Metodo main de recomendación (Crear conjunto de datos) 56
B.2. Metodo de obtención de vecinos 58
B.3. Método para mostrar la recomendación 58

C. Reglas personalizadas **60**
C.1. Clase ApiProcessor 60
C.2. createNewApi.php 64
C.3. DeleteUnusedTerm.php 65

C.4. DeleteUnusedFile.php 66
C.5. UploadApi.php 67

Capítulo 1

Introducción

Las APIs (Application Programming Interfaces) han existido desde los primeros días de la programación y han evolucionado significativamente a lo largo de la historia. En la actualidad, numerosas empresas y servicios las utilizan diariamente para gestionar información, brindar experiencias a los clientes y desarrollar servicios internos. Con el interés de las compañías en "apificar"sus servicios, se han creado una gran cantidad de APIs, lo que ha dado lugar a la aparición de los marketplaces de APIs.

Estos marketplaces ofrecen una amplia selección de APIs organizadas por categorías, funcionalidades y tipos de clientes. Con la gran cantidad de APIs disponibles, surge la necesidad de ofrecer a cada usuario una experiencia personalizada. Por lo tanto, la implementación de sistemas de recomendación en estos marketplaces se vuelve vital para ayudar a los usuarios a descubrir las APIs más relevantes y adecuadas a sus necesidades.

Durante sus prácticas extracurriculares en la empresa AXPE Consulting el autor de este trabajo ha realizado una implementación de un sistema de recomendación en un marketplace, con un enfoque especial en el mundo de las APIs. El presente Trabajo Fin de Grado explica por un lado los fundamentos teóricos de dicho sistema y por otro algunos detalles de la implementación.

1.1. Qué es una API

Una API (Application Programming Interface) es un conjunto de reglas y protocolos que permite a distintas aplicaciones o sistemas interactuar y comunicarse entre sí. Funciona como una interfaz que define la forma en que los componentes de software deben interactuar y compartir información.

En términos más sencillos, una API especifica cómo un software puede acceder a las funciones y datos de otro software para utilizarlos de manera controlada y segura. Esto permite que diferentes aplicaciones o servicios se conecten y trabajen juntos, incluso si están desarrollados en lenguajes de programación diferentes o se ejecutan en plataformas distintas.

Las APIs pueden proporcionar diferentes tipos de funcionalidades, como acceso a bases de datos, servicios web, funciones específicas de una aplicación o control de hardware. Estas interfaces definen los métodos, parámetros, formatos de datos y reglas de interacción necesarios para que las aplicaciones se comuniquen y compartan información de manera coherente.

En este proyecto la información de las APIs va ha ser gestionada mediante su especificación en forma de archivos yaml y json. Este tipo de archivo se caracteriza por tener un formato de fácil comprension para los humanos.
Un ejemplo sencillo de yaml es:

```
animal:
  especie: oso
  raza: pardo
  edad: 8
```

Un ejemplo de json sería:

```
{
  "animal": {
    "especie": "oso",
    "raza": "pardo",
    "edad": 8
  }
}
```

En este trabajo se utilizarán APIs basadas en la metodología REST (Representational State Transfer).

Qué es una API REST

Una API REST es una API que cumple con el diseño del estilo de arquitectura REST. Esta está dividida en seis puntos:

1. **Interfaz uniforme**. Cada elemento debe estar identificado de forma única por URI. Así cada usuario, contenido o dato tendrá su localizador propio.

2. **Desvinculamiento cliente-proveedor**. Las aplicaciones del cliente y del proveedor han de ser independientes entre sí, es decir, el único recurso que puede conocer la aplicación del cliente de la aplicación del proveedor es el URI y de la misma forma la del proveedor no podrá modificar la aplicación del cliente. Además, estas comunicaciones han de ser a través de HTTP.

3. **Sin estado**. Esto significa que cada solicitud debe contener toda la información necesaria para ser ejecutada y el proveedor no podrá registrar ninguna información relacionada con la solicitud del cliente.

4. **Almacenamiento en caché**. Los datos deben almacenarse en la memoria caché para así conseguir un menor tiempo de respuesta por parte del proveedor.

5. **Sistema por capas**. Es decir, puede que al ejecutar una orden, el proveedor tenga que realizar más acciones mediante solicitudes a otros proveedores. De esta manera, la comunicación entre cliente y proveedor no siempre tiene por qué ser directa.

6. **Código bajo demanda**. El proveedor puede tener la capacidad de enviar codigos ejecutables al servidor del cliente si es necesario. Por ejemplo, en la aplicación de GPS al introducir las coordenadas si éstas no existen, se puede comunicar el error antes de que sean enviadas.

Cómo funciona una API REST

Las API REST utilizan como medio de comunicación con las bases de datos solicitudes HTTP, las cuales sirven, por ejemplo, para crear, eliminar, editar y obtener información. Las principales acciones son:

- GET: Para obtener información.
- POST: Para crear nuevo contenido.
- DELETE: Para eliminar cierto contenido

Las instrucciones HTTP se realizan en una URL o directorio, conocido como *endpoint*. Este es esencial para establecer el lugar en el que se realiza la acción.

Por otra parte, las cabeceras de las solicitudes HTTP también son de gran importancia ya que tienen la información sobre el contenido que se quiere crear o autorizaciones que puede requerir el proveedor. Estas cabeceras son conocidas como *headers*.

Los principales empleados en este proyecto son:

- **Content-Type**: Indica el tipo de contenido que se retorna.
- **Accept**: Tipo de contenido el cliente puede procesar.
- **Content-Disposition**: Indica la disposición del contenido que por ejemplo, que quiere utilizar en un POST.
- **Authorization**: Credenciales del usuario en la web.

Los *headers* y la petición HTTP son enviados desde la terminal en forma de orden *curl*.

Las ordenes *curl* son una herramienta utilizada para intercambiar datos entre un cliente y una URL desde la terminal de Linux. Un ejemplo de una orden *curl* con un POST es:

```
curl --location --request POST 'https://axpe-developer.ddev.site/
    jsonapi/media/api_yaml/field_media_document' \
--header 'Content-Type: application/octet-stream' \
--header 'Accept: application/vnd.api+json' \
--header 'Content-Disposition: file;filename="AccountsManager_1
    .0.0.yaml";' \
--header 'X-CSRF-Token: 4
    HXW2OwIMBK3kAfHkSWWfsxKLxx6Uq6nwHvpU_pilXU' \
--header 'Authorization: Basic YWRtaW46MTIzNA==' \
--data-binary '@/home/david/Escritorio/Yamls/AccountsManager_1
    .0.0.yaml'
```

Beneficios de usar una API REST

Utilizar una API REST es beneficioso tanto para el cliente como para el proveedor dado que establece un método de comunicación sencillo y seguro entre cliente y proveedor sin importar el lenguaje de programación utilizado por éstos. Además, al no tener estado no es necesaria la carga de la información de las acciones anteriores ejecutadas por el usuario, consiguiendo una mejor respuesta por parte del proveedor.

1.2. Qué es un marketplace

Un *marketplace* es una plataforma en línea donde se reunen proveedores y compradores para realizar transacciones comerciales. En este espacio, los proveedores ofrecen productos o servicios, y los compradores pueden buscar, comparar y adquirir esos productos o servicios directamente del proveedor.

Los *marketplaces* pueden ser generales, abarcando una amplia variedad de productos o servicios, o especializados, centrándose en un nicho de mercado específico. Algunos ejemplos populares de marketplaces son Amazon, eBay y Airbnb.

Estas plataformas ofrecen ventajas tanto para los proveedores como para los compradores. Los proveedores pueden aprovechar la infraestructura existente del *marketplace* para llegar a una amplia base de clientes, mientras que los compradores pueden acceder a una amplia gama de productos en un solo lugar, facilitando la comparación de servicios y la toma de decisiones de compra.

Los *marketplaces* han experimentado un crecimiento significativo en los últimos años debido a su conveniencia y eficiencia en la facilitación del comercio en línea. Por ello son de gran utilidad los *marketplaces* de APIs.

En nuestro caso, los usuarios podrán cargar las especificaciones de la API en el *marketplace* y otros usuarios podrán descargarlas.

1.3. Sistemas de recomendación

Un sistema de recomendación es una herramienta o algoritmo diseñado para ayudar a los usuarios a descubrir y seleccionar elementos de su interés entre un conjunto más amplio de opciones disponibles. Estos elementos pueden ser productos, servicios, películas, canciones, libros, artículos, contactos, entre otros.
El objetivo principal de un sistema de recomendación es proporcionar sugerencias personalizadas y relevantes a cada usuario, teniendo en cuenta sus preferencias, historial de interacciones, comportamientos y características individuales. Al analizar estos datos, el sistema busca identificar patrones y similitudes entre los usuarios y los elementos, de manera que pueda realizar recomendaciones precisas y adaptadas a cada persona.

En un sistema de recomendación son fundamentales tres cosas:

- **Usuarios:** Un conjunto de usuarios del que se tiene una cierta información y a los que interesa hacer una recomendación.
- **Elementos:** Son los items que se recomendarán al usuario, es de gran importancia la información de como interactua cada usuario con cada uno de estos elementos.
- **Interacciones:** Será la acción o acciones en las que se basará la predicción. Esta interacción podría ser la visita de un usuario a una página, la descarga de un elemento, un reseña positiva...

Existen diferentes enfoques y técnicas utilizadas en los sistemas de recomendación. Nosotros vamos a usar filtros colaborativos basados en métricas entre usuarios. Lo haremos de dos maneras: por un lado comparando los usuarios directamente y por otro descomponiendo primero la matriz de interacciones por medio de sus valores singulares.
Los más comunes son:

- **Filtros colaborativos:** Estos sistemas analizan las interacciones y calificaciones pasadas de los usuarios para encontrar similitudes y relaciones entre ellos. Si dos usuarios tienen historiales de interacción similares, es probable que tengan gustos y preferencias similares. Por lo tanto, si un usuario ha calificado o interactuado positivamente con ciertos elementos, el sistema puede recomendar elementos similares a otros usuarios con perfiles similares.
- **Filtros basados en contenido:** Estos sistemas analizan las características y atributos de los elementos para encontrar similitudes entre ellos. Si un usuario ha mostrado interés en un elemento en particular, el sistema busca elementos con características similares y los recomienda. Por ejemplo, si a un usuario le gustan ciertas películas de acción, el sistema buscará otras películas con características similares, como el género, los actores principales o el director, y las sugerirá.
- **Filtrado basado en conocimiento:** Estos sistemas utilizan modelos de conocimiento o reglas predefinidas para realizar recomendaciones. Estas reglas se basan en el conocimiento experto o en teorías específicas sobre los elementos y los usuarios. Por ejemplo, un sistema de recomendación de películas puede utilizar un modelo de conocimiento sobre las características de los diferentes tipos de películas y las preferencias de los usuarios para hacer recomendaciones adecuadas.

- **Filtrado demográfico:** Estos sistemas utilizan información demográfica o características de los usuarios, como edad, género, ubicación o nivel educativo, para realizar recomendaciones. Se basan en patrones de comportamiento similares entre grupos demográficos para sugerir elementos relevantes.
- **Sistemas de recomendación híbridos:** Estos sistemas combinan múltiples técnicas y enfoques para mejorar la precisión de las recomendaciones. Pueden utilizar tanto filtros colaborativos como basados en contenido, además de incorporar otros factores como la popularidad de los elementos o las tendencias actuales.

Los sistemas de recomendación se aplican en una amplia variedad de plataformas y aplicaciones. Por ejemplo, en sitios web de comercio electrónico, como Amazon, se utilizan para sugerir productos relacionados o complementarios a los usuarios. En servicios de transmisión de contenido, como Netflix o Spotify, se utilizan para recomendar películas, series o canciones basadas en los gustos y preferencias de los usuarios. Incluso en redes sociales, se pueden utilizar sistemas de recomendación para sugerir conexiones o amistades basadas en intereses comunes.

1.4. Este trabajo

En el capítulo 2 estudiamos filtros colaborativos basados en métricas y cómo se utilizan en un sistema de recomendaciones. La sección 2.3 muestra un ejemplo de su uso.

En el capítulo 3 enunciamos y demostramos el Teorema de Descomposición en Valores Singulares, que dice (en el caso real) que toda matriz A se puede escribir como producto UDV donde U y V son matrices unitarias y D es una matriz diagonal con entradas no negativas. Las filas de U y las columnas de V son los autovectores de M^* y de M, donde $M = A^*A$, y los elementos de la diagonal de D son la raíz cuadrada de los autovalores de M.

Aunque en la práctica solo nos interesa el caso real el Teorema vale también para matrices complejas, que es el contexto en el que lo demostraremos, sin más que cambiar "ortogonales" por "unitarias", y "traspuesta" por "traspuesta conjugada".

En nuestro caso, A es la matriz de interacciones (descarga o no descarga) entre usuarios y APIs, de modo que las columnas de V son autovectores en el espacio de usuarios y las filas de U en el de APIs. Esto nos permite crear un filtro colaborativo también basado en métricas pero en el que el usuario al que queremos hacer la recomendación se compara con los autovectores en vez de con los usuarios individuales. Damos un ejemplo de cómo hacer esto en las Secciones 3.4 y 3.5.

Por último, el Capítuo 4 documenta la implementación realizada en el portal, que se ha hecho en el sistema de gestión de contenidos (CMS) Drupal. Se incluyen como apéndices parte del código de la implementación, en concreto los módulos que el autor ha implementado prácticamente desde cero (hay muchos otros en los que simplemente se ha modificado código de librerías preexistentes).

Capítulo 2

Filtros colaborativos

Un filtro colaborativo es un método de sistema de recomendación utilizado para predecir cuales son los gustos o preferencias de un usuario y asi poder ofrecer una experiencia personalizada. En estos filtros la información es proporcionada mediante la busqueda de patrones entre usuarios o elementos similares.

El filtrado colaborativo utiliza técnicas estadísticas o algoritmos de aprendizaje automático para analizar patrones en las clasificaciones o interacciones pasadas de los usuarios con los elementos. Estos patrones se utilizan para hacer predicciones sobre qué elementos serán de interés para un usuario en particular. Por ejemplo, si varios usuarios similares a un usuario en términos de preferencias han calificado positivamente un artículo o han realizado una compra específica, es probable que ese artículo también sea recomendado al usuario en cuestión.

El filtro colaborativo se puede implementar de diferentes formas, como filtrado basado en usuarios (que compara la similitud entre usuarios) o filtrado basado en elementos (que compara la similitud entre elementos). También existen enfoques híbridos que combinan múltiples técnicas para mejorar la precisión de las recomendaciones.

2.1. Esquema general del filtrado colaborativo

Para la aplicación de un filtrado colaborativo son fundamentales ciertos pasos:

- **Definición de similitud entre usuarios:** Esto implica definir una metrica que establezca el grado de parecido entre usuarios o elementos.
- **Búsqueda de los vecinos:** Utilizando la métrica anterior se seleccionan los usuarios más cercanos y por lo tanto los mas similares. Este conjunto de usuarios será denominado vecinos del usuario.
- **Establecer las predicciones:** A partir de la información que proporcionan los vecinos, se podrá realizar una predicción, que podrá ser vista como una probabilidad.

- **Realizar recomendación:** En base a las predicciones se recomendarán los elementos que más se ajusten al usuario, tomando los n elementos con mas probabilidad o estableciendo un umbral minimo a partir del cual todos los elementos que superen dicho umbral serán recomendados.

Habiendo establecido estos pasos se puede proceder a la formalización matemática del problema.

2.2. Formalización del problema

Para tener una mejor perspectiva del problema, es oportuno detenerse y considerar el problema general de la recomendación desde un enfoque formal. La tarea de recomendación puede ser comprendida de manera simple como la asignación de una probabilidad a que a un usuario le guste o vaya a utilizar un elemento o servicio. Para ello se establecen las siguientes definiciones:

$$U = \{u : 1 \leq u \leq n\}, \text{ donde } n \text{ es la cantidad total de usuarios.}$$

$$I = \{i : 1 \leq i \leq m\}, \text{ donde } m \text{ es la cantidad total de items.}$$

$$R \in \mathbb{R}^{n \times m}, \text{ donde } r_{i,j} \in R \text{ representa la interacción del usuario } i \text{ con el elemento } j$$

A través de estos elementos seremos capaces de analizar su información de forma que se pueda realizar una recomendación lo más acertada posible.

2.2.1. Métrica de similitud

Es necesario establecer una regla que defina cómo de similares son dos usuarios, para ello se define una métrica que se ajuste a nuestro problema. Algunas de las más utlizadas son:

- **Diferencia cuadrática media:**

$$sim(x, y) = \sqrt{\frac{1}{|I|} \sum_{i \in I} \left(\frac{r_{x,i} - r_{y,i}}{max - min} \right)^2}$$

  Donde max y min son los valores máximos y mínimos respectivamente que puede tomar $r_{i,j}, 1 \leq i \leq |U|, 1 \leq j \leq |I|$.

- **Correlación de Pearson:**

$$sim(x, y) = \frac{\sum_{i \in I} (r_{x,i} - \overline{r}_x) \cdot (r_{y,i} - \overline{r}_y)}{\sqrt{\sum_{i \in I} (r_{x,i} - \overline{r}_x)^2} \cdot \sqrt{\sum_{i \in I} (r_{y,i} - \overline{r}_y)^2}}$$

  Donde $\overline{r}$ representa la media de cada una de las componenetes de r

- **Coseno**

$$sim(x,y) = \frac{\sum_{i \in I} r_{x,i} \cdot r_{y,i}}{\sqrt{\sum_{i \in I} r_{x,i}^2} \cdot \sqrt{\sum_{i \in I} r_{y,i}^2}}$$

Como veremos más adelante, la diferencia cuadrática media es una métrica en el sentido matemático. La correlación y el coseno no son métricas porque por ejemplo pueden ser negativas, pero aún así también nos indican cómo de parecidos son los vectores. Por ejemplo, el coseno es 1 si uno de los vectores es múltiplo positivo del otro, cero si son ortogonales, y -1 si son opuestos. A la correlación de Pearson le pasa lo mismo pero con los vectores $r - \overline{r}$.

Al calcular la similitud entre elementos o usuarios, es posible establecer umbrales o criterios para identificar cuáles son los más similares. A partir de esta selección, se puede realizar una predicción basada en las preferencias de los usuarios similares, partiendo del supuesto de que aquellos con preferencias similares tendrán gustos o comportamientos similares.

Posteriormente veremos cómo seleccionamos los vecinos más próximos a cada usuario.

2.2.2. Búsqueda de vecinos

Como se mencionó anteriormente, un vecino de un usuario A es otro usuario B cuyos intereses podrían ser similares. La similitud entre usuarios se determina mediante el uso de una métrica específica.

Al emplear la métrica, podemos establecer relaciones entre usuarios y crear vecindarios. Estos vecindarios consisten en un grupo de vecinos (usuarios) y el número de vecinos puede ser decidido, siempre siendo menor al número total de usuarios.

Una vez que se han construido los vecindarios correspondientes para cada usuario, podemos proceder a realizar predicciones. Utilizando la información de los vecinos, podemos estimar las preferencias o interacciones que un usuario determinado podría tener con elementos o ítems con los que aún no ha interactuado. Estas predicciones se basan en el comportamiento y las interacciones pasadas de los vecinos similares. De esta manera, se aprovecha el conocimiento de usuarios similares para realizar recomendaciones personalizadas.

2.2.3. Crear tabla de recomendaciones y realizar recomendación

Para hacer la recomendación final se construye una matriz con la misma dimensión que R. En esta matriz, se asignan valores comprendidos entre 0 y 1, los cuales representan la probabilidad de que un elemento específico (columna) deba ser recomendado al usuario correspondiente (fila). Por lo tanto, los elementos con los que el usuario aún

no ha interactuado tendrán asignado el valor medio de las interacciones de sus vecinos con ese elemento. Por otro lado, los elementos con los que el usuario ya ha interactuado tendrán un valor de 0, con el propósito de no recomendarle elementos con los que ya ha tenido alguna interacción previa.

Finalmente, las recomendaciones se pueden concluir seleccionando los elementos con las puntuaciones más altas en cada fila de la matriz o estableciendo un umbral a partir del cual se considera que un elemento debe ser recomendado. Estos elementos seleccionados o que superen el umbral establecido se recomendarán al usuario correspondiente. De esta manera, se proporcionan recomendaciones personalizadas y relevantes al usuario.

2.3. Ejemplo de recomendación

En este ejemplo, utilizaremos un filtro colaborativo para recomendar a cada usuario las cuatro APIs que mejor se adapten a sus necesidades. La interacción que tomaremos en cuenta para realizar la recomendación será la descarga de la especificación API. Basándonos en las interacciones de descarga de las especificaciones API por parte de otros usuarios similares, identificaremos las APIs que son más populares o más relevantes para cada usuario en particular.

2.3.1. Definición de conjuntos

Comenzamos definiendo los distintos conjuntos de datos que se utilizarán en el problema. En este caso, hemos tomado 5 usuarios y 15 APIs. La interacción de descarga entre cada usuario y cada API ha sido generada de manera aleatoria.

Estos datos de interacción son fundamentales para analizar los patrones de comportamiento y generar recomendaciones personalizadas basadas en las preferencias de los usuarios.

Con esta información de descargas, podemos utilizar técnicas de filtrado colaborativo y métricas de similitud para identificar patrones y establecer relaciones entre los usuarios y las APIs. Esto nos permitirá generar predicciones y recomendaciones más precisas y relevantes para cada usuario en función de las descargas realizadas por usuarios similares en relación a las APIs.

Los conjuntos de datos quedarían de la siguiente forma:

$$U = \{u : 1 \leq u \leq 5\}$$
$$I = \{i : 1 \leq i \leq 15\}$$

$$R = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Para mostrar de manera más intuitiva los datos, se puede construir la siguiente tabla:

$\boldsymbol{R}$	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i_{12}	i_{13}	i_{14}	i_{15}
u_1	1	0	1	1	0	1	1	1	1	0	0	0	0	1	1
u_2	1	1	1	0	0	1	0	1	0	0	1	1	0	1	1
u_3	0	0	0	1	0	0	0	1	0	1	1	1	1	0	1
u_4	1	1	1	1	1	0	0	1	1	0	0	1	1	1	0
u_5	1	0	1	1	1	1	1	0	0	1	1	0	0	1	1

La tabla muestra la interacción de los usuarios con las APIs, donde un valor de 1 indica que el usuario ha descargado la API y un valor de 0 indica que no lo ha hecho.

2.3.2. Selección de la métrica

Por otra parte, es necesario definir una métrica para buscar similitudes entre pares de usuarios. En este caso, utilizamos la diferencia cuadrática media (RMSE, por sus siglas en inglés), que se calcula mediante la siguiente fórmula:

$$sim(x,y) = \sqrt{\frac{1}{|I|}\sum_{i\in I}\left(\frac{r_{x,i}-r_{y,i}}{max-min}\right)^2} = \sqrt{\frac{1}{15}\sum_{i=1}^{15}\left(\frac{r_{x,i}-r_{y,i}}{1-0}\right)^2}$$
$$= \sqrt{\frac{1}{15}\sum_{i=1}^{15}\left(\frac{r_{x,i}-r_{y,i}}{1}\right)^2} = \sqrt{\frac{1}{15}\sum_{i=1}^{15}\left(r_{x,i}-r_{y,i}\right)^2}$$

En adelante se tomará como refencia el valor positivo de la raíz cuadrada.

Para demostrar que la fórmula cumple con la definición de métrica, debemos verificar si cumple con las propiedades necesarias. Una métrica debe satisfacer las siguientes condiciones:

- **No negatividad:** Los valores nunca pueden ser negativos, es decir, $sim(x,y) \geq 0$:

$$0 \leq r_{i,j} \leq 1 \Rightarrow 0 \leq \left(r_{x,i}-r_{y,i}\right)^2 \leq 1 \Rightarrow 0 \leq \sum_{i=1}^{15}\left(r_{x,i}-r_{y,i}\right)^2 \leq 15$$
$$\Rightarrow 0 \leq \frac{1}{15}\sum_{i=1}^{15}\left(r_{x,i}-r_{y,i}\right)^2 \leq 1 \Rightarrow 0 \leq \frac{1}{15}\sum_{i=1}^{15}\left(r_{x,i}-r_{y,i}\right)^2 \Rightarrow \frac{1}{15}\sum_{i=1}^{15}\left(r_{x,i}-r_{y,i}\right)^2 \in \mathbb{R}_{\geq 0}$$
$$\Rightarrow +\sqrt{\frac{1}{15}\sum_{i=1}^{15}\left(r_{x,i}-r_{y,i}\right)^2} \in \mathbb{R}_{\geq 0} \Rightarrow sim(x,y) \geq 0$$

- **Identidad de indiscernibles:** La métrica debe ser igual a cero si y solo si los elementos comparados son idénticos, es decir, $sim(x, y) = 0 \Leftrightarrow x = y$:

$$sim(x, y) = 0 \Leftrightarrow \sqrt{\frac{1}{15}\sum_{i=1}^{15}(r_{x,i} - r_{y,i})^2} = 0 \Leftrightarrow \frac{1}{15}\sum_{i=1}^{15}(r_{x,i} - r_{y,i})^2 = 0$$

$$\Leftrightarrow \sum_{i=1}^{15}(r_{x,i} - r_{y,i})^2 = 0 \Leftrightarrow (r_{x,i} - r_{y,i})^2 = 0, \forall i \in 1, ..., 15 \Leftrightarrow r_{x,i} = r_{y,i} \forall i \in 1, ..., 15 \Leftrightarrow x = y$$

Observación 2.1. *Puede ocurrir que dos usuarios distintos tengan exactamente el mismo vector asociado. En este caso, la RSME es una métrica en el espacio de los vectores pero es solo una pseudo-métrica en el conjunto de los usuarios.*

- **Simetría:** La métrica debe ser hermítica, es decir, el orden de los elementos comparados no debe afectar el resultado, por lo tanto $sim(x, y) = sim(y, x)$:

$$sim(x, y) = \sqrt{\frac{1}{15}\sum_{i=1}^{15}(r_{x,i} - r_{y,i})^2} =$$

$$\sqrt{\frac{1}{15}\sum_{i=1}^{15}(-1 \cdot (r_{x,i} - r_{y,i}))^2} = \sqrt{\frac{1}{15}\sum_{i=1}^{15}(r_{y,i} - r_{x,i})^2} = sim(y, x)$$

- **Desigualdad triangular:** La métrica debe cumplir con la desigualdad triangular, lo que significa que la distancia entre dos elementos siempre es menor o igual a la suma de las distancias entre esos elementos y un tercer elemento. Con lo que $sim(x, z) \leq sim(x, y) + sim(y, z)$:

$$sim(x, z) \leq sim(x, y) + sim(y, z)$$

$$\Leftrightarrow$$

$$\sqrt{\frac{1}{15}\sum_{i=1}^{15}(r_{x,i} - r_{z,i})^2} \leq \sqrt{\frac{1}{15}\sum_{i=1}^{15}(r_{x,i} - r_{y,i})^2} + \sqrt{\frac{1}{15}\sum_{i=1}^{15}(r_{y,i} - r_{z,i})^2}$$

$$\Leftrightarrow$$

$$\sqrt{\sum_{i=1}^{15}(r_{x,i} - r_{y,i})^2} + \sqrt{\sum_{i=1}^{15}(r_{y,i} - r_{z,i})^2} \geq \sqrt{\sum_{i=1}^{15}(r_{x,i} - r_{z,i})^2}$$

donde haciendo el cambio $r_{x,i} - r_{y,i} = a_i$ y $r_{y,i} - r_{z,i} = b_i$ se obtiene $r_{x,i} - r_{z,i} = a_i + b_i$ y por lo tanto:

$$\sqrt{\sum_{i=1}^{15} a_i^2} + \sqrt{\sum_{i=1}^{15} b_i^2} \geq \sqrt{\sum_{i=1}^{15}(a_i + b_i)^2}$$

Lo cual podemos verificar utilizando la desigualdad de Cauchy-Buniakovski-Schwarz:

$$\sum_{i=1}^{15} a_i^2 \cdot \sum_{i=1}^{15} b_i^2 \geq \left(\sum_{i=1}^{15} a_i \cdot b_i \right)^2$$

Empleando esta desigualdad en nuestro problema obtendremos el ultimo paso paso de esta demostración:

$$\sqrt{\sum_{i=1}^{15} (a_i + b_i)^2} = \sqrt{\sum_{i=1}^{15} a_i^2 + \sum_{i=1}^{15} b_i^2 + 2 \sum_{i=1}^{15} a_i b_i} \leq \tag{2.1}$$

$$\sqrt{\sum_{i=1}^{15} a_i^2 + \sum_{i=1}^{15} b_i^2 + 2\sqrt{\sum_{i=1}^{15} a_i^2 \cdot \sum_{i=1}^{15} b_i^2}} = \sqrt{\left(\sqrt{\sum_{i=1}^{15} a_i^2} + \sqrt{\sum_{i=1}^{15} b_i^2} \right)^2} = \sqrt{\sum_{i=1}^{15} a_i^2} + \sqrt{\sum_{i=1}^{15} b_i^2}$$

Con todo esto queda visto que estamos empleando una métrica para calcular la similitud.

Con esta fórmula, es posible calcular de manera sencilla la similitud entre dos usuarios y generar la siguiente tabla:

$sim(u_i, u_j)$	u_1	u_2	u_3	u_4	u_5
u_1	0.0	0.4	0.67	0.47	0.33
u_2	0.4	0.0	0.53	0.47	0.47
u_3	0.67	0.53	0.0	0.6	0.6
u_4	0.47	0.47	0.6	0.0	0.67
u_5	0.33	0.47	0.6	0.67	0.0

Con las similitudes ya calculadas podemos procedes a buscar los usuarios más similares entre sí.

2.3.3. Busqueda de vecinos

En este ejemplo, cada usuario tendrá dos vecinos, que serán aquellos usuarios que hayan obtenido las menores puntuaciones en cuanto a similitud, lo que significa que son los mas similares a ellos. Sin embargo, se excluye al propio usuario para evitar recomendarle algo que ya ha descargado, como se mencionó anteriormente.

Las relaciones entre usuarios quedarían así:

usuario	vecinos
u_1	u_5 u_2
u_2	u_1 u_4
u_3	u_2 u_4
u_4	u_1 u_2
u_5	u_1 u_2

Con los valores de estos vecinos, podemos proceder a crear la tabla de recomendaciones en la que se muestran las probabilidades de que una API sea la idónea para el usuario.

2.3.4. Realizar una recomendación

$\boldsymbol{R}$	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i_{12}	i_{13}	i_{14}	i_{15}
u_1	0	0.5	0	0	0.5	0	0	0	0	0.5	1	0.5	0	0	0
u_2	0	0	0	1	0.5	0	0.5	0	1	0	0	0	0.5	0	0
u_3	1	1	1	0	0.5	0.5	0	0	0.5	0	0	0	0	1	0
u_4	0	0	0	0	0	1	0.5	0	0	0	0.5	0	0	0	1
u_5	0	0.5	0	0	0	0	0	1	0.5	0	0	0.5	0	0	0

Utilizando los valores asignados, es posible realizar recomendaciones considerando las APIs que tengan un valor mayor a 0.5. En este caso, las recomendaciones finales para cada usuario serían las siguientes:

usuario	vecinos
u_1	i_{11}
u_2	i_4, i_9
u_3	i_1, i_2, i_3, i_{14}
u_4	i_6, i_{15}
u_5	i_8

2.4. Análisis de resultados

En este ejemplo, se ha mostrado una aplicación de un filtro colaborativo de forma básica con el proposito de observar su funcionamiento y versatilidad. Sin embargo, hay casos en los que a algunos usuarios se les recomendará solo un ítem, mientras que a otros se les recomendarán cuatro ítems. Además, puede haber situaciones en las que no se pueda realizar una recomendación.

Esto se debe a que se ha establecido un umbral como criterio para realizar las recomendaciones. Si se hubiera seleccionado el ítem más similar como recomendación, se aseguraría que cada usuario siempre recibiera una recomendación. Sin embargo, esto conlleva ciertos inconvenientes, como posibles empates. Es decir, si hay dos ítems con la misma similitud y sólo se desea recomendar uno, habría que seleccionar uno de ellos.

Capítulo 3

Descomposición en valores singulares (SVD)

En este capítulo se abordará y explorará el método conocido como SVD (Descomposición en Valores Singulares, por sus siglas en inglés), brindando una explicación detallada de sus conceptos, aplicaciones y propiedades. Usaremos los libros [1, 3] como referencias básicas de álgebra lineal.

3.1. Conceptos básicos de SVD

La Descomposición en Valores Singulares es una técnica matemática basada en el álgebra lineal y ampliamente utilizada en el campo del análisis de datos. Proporciona una descomposición fundamental de una matriz en tres componentes principales: una matriz de vectores singulares izquierdos, una matriz de valores singulares y una matriz de vectores singulares derechos. La matriz original A de tamaño $m \times n$ se descompone en la forma:

$$A = U \cdot \Sigma \cdot V^*$$

Donde U es una matriz unitaria de tamaño $m \times m$, Σ es una matriz diagonal rectangular de tamaño $m \times n$ con valores singulares no negativos en la diagonal principal, y V^* es la matriz transpuesta conjugada (también llamada la adjunta) de otra matriz unitaria V de tamaño $n \times n$. Los conceptos de matrices unitarias, adjuntas y hermíticas se definen en la siguiente sección. Si las matrices son reales unitario es lo mismo que ortogonal y hermítica es lo mismo que hermítica.

Los valores singulares en la matriz Σ representan las magnitudes o importancias relativas de los componentes principales de la matriz original. Estos valores singulares se ordenarán de manera descendente, lo que significa que los primeros valores singulares capturan la mayor parte de la variabilidad de los datos, mientras que los últimos valores singulares representan componentes menos importantes.

Los vectores singulares izquierdos en la matriz U y los vectores singulares derechos en la matriz V representan las direcciones y estructuras características de los datos. Los vectores singulares izquierdos correspondientes a los valores singulares más grandes

son los vectores principales que capturan la mayor parte de la variabilidad en los datos.

Veamos a continuación la existencia de esta descomposición.

3.2. Demostración SVD

Para la desmotración de la existencia de la descomposición en valores singulares de una matriz primero es necesario establecer unas definiciones y conceptos básicos.

Definición 3.1 (Adjunto de un vector o una matriz)**.** *Sea A una matriz compleja. Llamamos adjunta de A a la conjugada de la traspuesta. Es decir $A^* := \overline{A}^*$. Del mismo modo, si $\vec{v}$ es un vector columna, su adjunto $\vec{v}^*$ es el conjugado del mismo vector como fila.*

Se observa que tanto para matrices como para vectores se cumple que $(AB)^* = B^*A^*$.

Definición 3.2 (Norma vectorial)**.** *Dados dos vectores complejos $\vec{u}, \vec{v} \in \mathbb{C}^n$ su producto interno se define como*

$$\langle \vec{u}, \vec{v} \rangle := \vec{v}^* \cdot \vec{u} = \sum_{i=1}^{n} u_i \bar{v}_i.$$

Sea $\vec{v}$ un vector, se llama norma vectorial a:

$$\|\vec{v}\|_2 = \sqrt{\langle \vec{v}, \vec{v} \rangle} = \sqrt{\sum_{i=1}^{n} |v_i|^2}.$$

Por tanto se tiene:

$$\|\vec{v}\|_2^2 = \vec{v}^* \cdot \vec{v}$$

Dos vectores son ortogonales si su producto interno es cero. Un vector es unitario si su norma es uno.

Definición 3.3 (Base ortonormal)**.** *Se dice que una base de vectores $S = \{v_1, v_2, ..., v_k\} \in \mathbb{C}^n$ es una base ortonormal si sus vectores son ortogonales entre sí y su norma es 1, es decir:*

$$\begin{cases} \langle \vec{v}_i, \vec{v}_j \rangle = 0 & si\ i \neq j \\ \langle \vec{v}_i, \vec{v}_j \rangle = 1 & si\ i = j \end{cases}$$

Definición 3.4 (Matriz unitaria)**.** *Sea A una matriz compleja. Se dice que A es unitaria si es una matriz cuadrada y $A^* = A^{-1}$.*

Equivalentemente, si sus filas y sus columnas forman

Lema 3.5. *Sea A una matriz cuadrada compleja. Las siguientes propiedades son equivalentes:*

1. *A es unitaria.*

2. *$A^* \cdot A = Id$.*

3. *Las columnas de A forman una base ortonormal.*

4. *Las filas de A forman una base ortonormal.*

Demostración. La equivalencia entre las propiedades 1 y 2 se puede demostrar de forma inmediata ya que utilizando la propia definición de una matriz unitaria 3.4 se tiene que $A^* = A^{-1}$, en consecuencia, $A^* \cdot A = Id$.

Por otra parte podemos ver la matriz A como un conjunto de vectores columna:

$$A = \left(\vec{v}_1 \vec{v}_2 \dots \vec{v}_n\right)$$

donde n representa el número de columnas de A.

Con la propiedad 1 tenemos que $A^* \cdot A = Id$, esto solo es posible si:

$$A^* \cdot A = \begin{pmatrix} \vec{v}_1^* \\ \vec{v}_2^* \\ \vdots \\ \vec{v}_n^* \end{pmatrix} \cdot \left(\vec{v}_1 \vec{v}_2 \dots \vec{v}_n\right) = \begin{pmatrix} \vec{v}_1^* \cdot \vec{v}_1 & \vec{v}_1^* \cdot \vec{v}_2 & \dots & \vec{v}_1^* \cdot \vec{v}_n \\ \vec{v}_2^* \cdot \vec{v}_1 & \vec{v}_2^* \cdot \vec{v}_2 & \dots & \vec{v}_2^* \cdot \vec{v}_n \\ \vdots & \vdots & & \vdots \\ \vec{v}_n^* \cdot \vec{v}_1 & \vec{v}_n^* \cdot \vec{v}_2 & \dots & \vec{v}_n^* \cdot \vec{v}_n \end{pmatrix} = Id_n$$

Donde se observa que haciendo uso de la definición 3.3 se tiene que las columnas de A forman una base ortonormal.

Del mismo modo tomando A como un conjunto de vectores fila se tiene:

$$A \cdot A^* = \begin{pmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vdots \\ \vec{v}_m \end{pmatrix} \cdot \left(\vec{v}_1^* \vec{v}_2^* \dots \vec{v}_m^*\right) = \begin{pmatrix} \vec{v}_1 \cdot \vec{v}_1^* & \vec{v}_1 \cdot \vec{v}_2^* & \dots & \vec{v}_1 \cdot \vec{v}_m^* \\ \vec{v}_2 \cdot \vec{v}_1^* & \vec{v}_2 \cdot \vec{v}_2^* & \dots & \vec{v}_2 \cdot \vec{v}_m^* \\ \vdots & \vdots & & \vdots \\ \vec{v}_m \cdot \vec{v}_1^* & \vec{v}_m \cdot \vec{v}_2^* & \dots & \vec{v}_m \cdot \vec{v}_m^* \end{pmatrix} = Id_m$$

Dada la conmutatividad del producto escalar se obtiene que las filas de A son una base ortonormal. □

Definición 3.6 (Hermítica)**.** *Dada una matriz $A \in \mathbb{C}^{n \times n}$ diremos que es hermítica o autoadjunta si, $A^* = A$.*

Lema 3.7 ([3, Prop. 3.1.1])**.** *Sea A una matriz hermítica.*

1. *Todos los autovalores de A son reales.*

2. *Si u y v son autovectores con distinto autovalor, entonces son ortogonales.*

Demostración. Para la parte 1, sea λ un autovalor y v uno de sus autovectores. Es decir, $Av = \lambda v$. Entonces, se tiene también que $v^* A^* = \bar{\lambda} v^*$ y por tanto:

$$\lambda ||v||^2 = \lambda v^* \cdot v = v^* A v = v^* A^* v = v^* \bar{\lambda} v = \bar{\lambda} v^* \cdot v = \bar{\lambda} ||v||^2.$$

Como $||v||^2$ no es cero, concluimos que $\lambda = \bar{\lambda}$.

Para la segunda parte, si λ y μ son dos autovalores distintos, con autovectores u y v respectivamente, tenemos que

$$v^*Au = (v^*A)v = (v^*A^*)u = \bar{\mu}\, v^* \cdot u = \mu\, v^* \cdot u$$

pero también

$$v^*Au = v^*(Au) = \lambda\, v^* \cdot u.$$

Como $\lambda \neq \mu$, concluimos que $v^* \cdot u = 0$, o sea, que son ortogonales. □

Obsérvese que si A tiene entradas reales entonces hermítica significa lo mismo que "simétrica".

Definición 3.8 (Semi-definida positiva)**.** *Dada una matriz hermítica $A \in \mathbb{C}^{n\times n}$ diremos que es semi-definida positiva si, $x^*Ax \geq 0$ para todo $x \in \mathbb{C}^n$.*

La siguiente propiedad es un caso particular del Teorema 9.12 de [1].

Lema 3.9. *Los autovalores de una matriz A semidefinida positiva son positivos o cero.*

Demostración. Si λ es un autovalor con autovector v se tiene que

$$\lambda||v||^2 = v^*\lambda v = v^*Av \geq 0.$$

Como $||v||^2 > 0$ (porque un autovector no puede ser cero), esto implica que $\lambda \geq 0$. □

Propiedades 3.10. *Si A es una matriz cuadrada semi-definida positiva se tiene:*

- *Todos sus elementos diagonales son no negativos.*
- *Su determinante es no negativo.*
- *Sus menores principales (determinantes de las submatrices cuadradas superior-izquierda) son no negativos o cero. De hecho, esta propiedad es una caracterización de las matrices semidefinidas positivas.*

Teorema 3.11 (Teorema espectral)**.** *Toda matriz hermítica A puede ser diagonalizada mediante una matriz unitaria. Es decir, existe una matriz unitaria P tal que $P^*AP = D$, donde D es una matriz diagonal no negativa.*

De hecho, las columnas de P son una base ortonormal de autovectores de A y las entradas de D son los autovalores.

Omitimos la demostración de este teorema, que se puede encontrar en [1, Teorema 8.18, p. 311] o en [3, Teorema 3.2.1].

Conociendo estos conceptos y propiedades podemos proceder a la formulación y demostracion del teorema SVD. El Teorema aparece enunciado como Teorema 7.7.1 en [3] pero la demostración que incluimos está basada en []

Teorema 3.12 (Singular Value Descomposition)**.** *Sea $A \in \mathbb{C}^{m\times n}$, existen matrices unitarias $U \in \mathbb{C}^{m\times m}$ y $V \in \mathbb{C}^{n\times n}$ y $\Sigma \in \mathbb{C}^{m\times n}$, matriz diagonal positiva, tales que:*

$$A = U \cdot \Sigma \cdot V^*$$

Demostración. Sea $A \in \mathbb{C}^{m\times n}$, definimos la matriz $M \in \mathbb{C}^{n\times n}$ como $M = A^* \cdot A$. Veamos algunas propiedades de M:

- **Hermítica:** Tendremos que ver que $M^* = M$:

$$M^* = (A^*A)^* = A^*A^{*^*} = A^*A = M$$

- **Semidefinida positiva:** Tendremos que ver que $x^*Mx \geq 0, \forall x \in \mathbb{C}^n$:

$$x^*Mx = x^*A^*Ax = (Ax)^*(Ax) = \|Ax\|_2^2 \geq 0$$

Por ser M una matriz hermítica y semidefinida positiva se tiene que existe una matriz unitaria $V \in \mathbb{R}^{n\times n}$ tal que $V^*MV = D$ con $D \in \mathbb{R}^{n\times n}$ diagonal y formada por los autovalores de M (Teorema espectral 3.11). En consecuencia, tendremos:

$$V^*MV = D$$

Si r es el rango de M, tendrá $n - r$ autovalores nulos. En función de los autovalores asociados nulos descomponemos V de la siguiente manera:

$$V_{(n\times n)} = \left[V_{1(n\times r)} V_{2(n\times n-r)}\right]$$

De la misma manera descompondremos D:

$$D_{(n\times n)} = \begin{bmatrix} \widetilde{D}_{r\times r} & 0_{r\times(n-r)} \\ 0_{(n-r)\times r} & 0_{(n-r)\times(n-r)} \end{bmatrix}$$

Definimos ahora a la matriz diagonal $\widetilde{\Sigma} = \sqrt{\widetilde{D}}$, es decir, $\widetilde{\Sigma}^2 = \widetilde{D}$, por lo tanto:

$$\begin{bmatrix} V_1^* \\ V_2^* \end{bmatrix} \cdot A^* \cdot A \cdot \left[V_1 V_2\right] = D = \begin{bmatrix} \widetilde{\Sigma} \cdot \widetilde{\Sigma} & 0_{r\times(n-r)} \\ 0_{(n-r)\times r} & 0_{(n-r)\times(n-r)} \end{bmatrix} = \begin{bmatrix} V_1^*A^*AV_1 & V_1^*A^*AV_2 \\ V_2^*A^*AV_1 & V_2^*A^*AV_2 \end{bmatrix}$$

Donde podemos observar:

$$\Sigma^2 = \begin{bmatrix} \widetilde{\Sigma}^2 & 0_{r\times(n-r)} \\ 0_{(n-r)\times r} & 0_{(n-r)\times(n-r)} \end{bmatrix} = \begin{bmatrix} V_1^*A^*AV_1 & 0_{r\times(n-r)} \\ 0_{(n-r)\times r} & 0_{(n-r)\times(n-r)} \end{bmatrix}$$

Una matriz diagonal sin ceros en la diagonal es invertible, tomando simplemente los inversos de las Cntradas. Por lo tanto, al ser $\widetilde{\Sigma}$ diagonal y con todas las entradas de la diagonal positivas podremos continuar de la siguiente manera:

$$\begin{aligned} &\widetilde{\Sigma}^2 = V_1^*A^*AV_1 \Rightarrow \\ &\Rightarrow Id_{r\times r} = \widetilde{\Sigma}^{-1}\widetilde{\Sigma} \cdot \widetilde{\Sigma}\widetilde{\Sigma}^{-1} = \widetilde{\Sigma}^{-1}V_1^*A^*AV_1\widetilde{\Sigma}^{-1} = (AV_1\widetilde{\Sigma}^{-1})^* \cdot (AV_1\widetilde{\Sigma}^{-1}) \end{aligned}$$

Denotamos $U_1 = AV_1\widetilde{\Sigma}^{-1}, U \in \mathbb{C}^{m\times r}$, además podemos notar que U_1 es unitaria, ya que $U_1^* \cdot U_1 = Id$. Multiplicando $\widetilde{\Sigma}$ en ambos lados de la ecuación obtendremos:

$$U_1\widetilde{\Sigma} = AV_1$$

Al no ser V_1 cuadrada no podemos multiplicar por su inversa, pero como $A \cdot V_2 = 0$ podemos continuar:

$$U_1\widetilde{\Sigma} = AV_1 \Rightarrow \begin{bmatrix} U_1\widetilde{\Sigma} & 0_{r\times(n-r)} \\ 0_{(n-r)\times r} & 0_{(n-r)\times(n-r)} \end{bmatrix} = \begin{bmatrix} AV_1 & AV_2 \end{bmatrix} = AV \Rightarrow$$
$$\Rightarrow \begin{bmatrix} U_1\widetilde{\Sigma} & 0_{r\times(n-r)} \\ 0_{(n-r)\times r} & 0_{(n-r)\times(n-r)} \end{bmatrix} V^* = AVV^* = A$$

Por lo tanto podríamos tomar Σ y U:

$$\Sigma = \begin{bmatrix} \widetilde{\Sigma} & 0_{r\times(n-r)} \\ 0_{(n-r)\times r} & 0_{(n-r)\times(n-r)} \end{bmatrix}, U = \begin{bmatrix} U_1 & U_2 \end{bmatrix}$$

Con U_2 cualquier matriz perteneciente a $\mathbb{C}^{m\times(m-r)}$ obtenida extendiendo las columnas de U_1 a una base ortonormal de C^m.

Finalmente, podremos concluir con la expresión:

$$A = U \cdot \Sigma \cdot V^*$$

□

3.3. Cálculo y propiedades de las matrices de SVD

De la demostración previa se puede deducir la forma y el cálculo de cada una de las matrices que componen la factorización del SVD (Descomposición en Valores Singulares).

En esta sección se expliacará la factorización SVD de $A \in \mathbb{C}^{m\times n}$, de rango r, la obtención de sus tres componentes y se analizarán algunas de sus propiedades principales.

3.3.1. Matriz V

En la demostración se vio que haciendo uso del teorema espectral 3.11 se puede obtener el valor de V, ya que esta matriz cumple que $V^*MV = D$, donde $M = A^*A$, V es una matriz formada por una base ortonormal de los autovectores de propios de M y D una matriz diagonal, en la que la diagonal principal está formada por los valores propios asociados a M.

Por lo tanto para obtener V sería necesario calcular $M = A^*A, M \in \mathbb{C}^{n\times n}$. Seguidamente bastaría con resolver la ecuación:

$$det(M - \lambda \cdot Id_{n\times n}) = 0, \lambda \in \mathbb{R}$$

Los valores de λ son los autovalores de M y habrá tantos (contados con multiplicidad) como la dimensión de M por ser Hermítica, por lo tanto habrá n autovalores, de los cuales r serán no nulos. A su vez, habrá n autovectores asociados a estos autovalores, los

cuales como ya se mencionó anteriormente, formarán V. Para la obtención de dichos autovectores basta con resolver el sistema de ecuaciones que proporciona una de las propiedades de los autovalores y autovectores ($\vec{v}$) de una matriz y ortonormalizar el resultado :

$$M\vec{v} = \lambda\vec{v}$$

Una vez obtenidos los n autovectores $\vec{v}_i, 1 \leq i \leq n$ se podrá construir la matriz V, ya que los autovectores formarán sus filas. De forma que la matriz finalmente quedaría como:

$$V = \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vdots \\ \vec{v}_n \end{bmatrix} = \begin{bmatrix} \vec{v}_{1_1} & \vec{v}_{1_2} & \cdots & \vec{v}_{1_n} \\ \vec{v}_{2_1} & \vec{v}_{2_2} & \cdots & \vec{v}_{2_n} \\ \vdots & \vdots & \ddots & \vdots \\ \vec{v}_{n_1} & \vec{v}_{n_2} & \cdots & \vec{v}_{n_n} \end{bmatrix}$$

3.3.2. Matriz Σ

Para obtener la matriz Σ, se requiere obtener la matriz diagonal de autovalores D, mencionada anteriormente. Recordemos que estos autovalores son el resultado del sistema de ecuaciones 5.15, por lo tanto hay r autovalores, que serán no nulos por ser M hermítica. Con ellos se construye la diagonal principal de $\overline{\Sigma}$, colocando la raíz cuadrada de cada uno de estos autovalores, en orden descendente.

En otras palabras, si D es la matriz diagonal de autovalores de M, con los autovalores $\lambda_1, \lambda_2, \ldots, \lambda_r$, no nulos, en su diagonal principal, la matriz $\overline{\Sigma}$ se construye de la siguiente manera:

$$\overline{\Sigma} = \begin{bmatrix} \sqrt{\lambda_1} & 0 & \ldots & 0 \\ 0 & \sqrt{\lambda_2} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sqrt{\lambda_r} \end{bmatrix}$$

Es importante tener en cuenta que los autovalores deben estar ordenados de mayor a menor antes de calcular la raíz cuadrada y construir la matriz Σ.

Finalmente para obtener la matriz Σ basta con completar $\overline{\Sigma}$ con filas y columnas de ceros hasta obtener una matriz pertenciente a $\mathbb{C}^{m\times n}$.

3.3.3. Matriz U

De forma similar al cálculo de V, para la obtencion de U construimos una matriz $M \in \mathbb{C}^{m\times m} M = AA^*$ de forma que empleando el propio teorema SVD 3.12 obtenemos la siguiente igualdad:

$$M = AA^* = (U\Sigma V^*) \cdot (U\Sigma V^*)^* = U\Sigma V^* V \Sigma^* U^*$$

Al ser V unitaria, $V^*V = Id$, y como Σ es diagonal, trasposición no la modifica. Por lo tanto se tiene:

$$M = U\Sigma\Sigma U^* = U\Sigma^2 U^* = UDU^*$$

Con lo que se observa que U es la matriz que forma junto con D la diagonalización de M. Por lo que para obtener U bastaría con obtener los autovectores de M^* de manera similar a la ya explicada con V.

3.3.4. Propiedades

En el contexto de la recomendación, las propiedades de cada una de las matrices en la descomposición SVD tienen las siguientes implicaciones:

- **Matriz U:** La matriz U captura las relaciones y similitudes entre los usuarios. Cada columna de U representa un usuario y sus valores correspondientes en esa columna reflejan la importancia o relevancia de los componentes principales para ese usuario. Al utilizar la matriz U en la recomendación, podemos identificar usuarios similares en función de sus vectores singulares izquierdos y utilizar esa información para generar recomendaciones.
- **Matriz Σ:** La matriz Σ representa la importancia relativa de los componentes principales en los datos originales. Los valores singulares en la diagonal principal de Σ están ordenados de mayor a menor importancia. Al utilizar la matriz Σ en la recomendación, podemos seleccionar los valores singulares más altos para capturar la variabilidad más significativa de los datos y generar recomendaciones en función de esos componentes principales.
- **Matriz V^*:** La matriz V^* captura las relaciones y similitudes entre los elementos o ítems en el sistema de recomendación. Cada columna de V^* representa un elemento y sus valores correspondientes en esa columna reflejan la importancia o relevancia de los componentes principales para ese elemento. Al utilizar la matriz V^* en la recomendación, podemos identificar elementos similares en función de sus vectores singulares derechos y utilizar esa información para generar recomendaciones.

En resumen, las matrices U, Σ y V^* en la descomposición SVD permiten capturar y representar las relaciones entre usuarios, la importancia de los componentes principales y las similitudes entre elementos. Estas propiedades se pueden combinar con filtros colaborativos para generar recomendaciones precisas y personalizadas.

3.4. Ejemplo de descomposición SVD

En esta sección, se llevará a cabo la implementación práctica del método de recomendación SVD con el objetivo de ofrecer una explicación concreta de los conceptos presentados en este capítulo.

3.4.1. Definición de los conjuntos

En este ejemplo, consideraremos 2 usuarios y 3 elementos. La interacción que se tomará en cuenta para los datos será la descarga de un elemento. Los conjuntos se definirán de la siguiente manera:

$$U = \{u : 1 \leq u \leq 3\}$$

$$I = \{i : 1 \leq i \leq 5\}$$

La matriz de interacciones A se representará de la siguiente manera:

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

En esta matriz, cada fila representa a un usuario y cada columna representa a un elemento. Los valores en la matriz indican la interacción de descarga del usuario con el elemento. Un valor de 1 significa que el usuario ha descargado el elemento, mientras que un valor de 0 indica que no lo ha descargado.

Con esta configuración de usuarios, elementos y matriz de interacciones, podremos aplicar el método de recomendación SVD para obtener puntuaciones de predicción y generar recomendaciones personalizadas.

3.4.2. Cálculo de V

Para el cálculo de V tenemos que obtener la matriz $M = A^*A$:

$$M = A^*A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Para obtener los autovalores de la matriz M, debemos resolver la ecuación $|M - \lambda Id| = 0$, donde λ es el autovalor y Id es la matriz identidad. Sustituyendo los valores de la matriz M, tenemos:

$$|M - \lambda Id| = \begin{vmatrix} 3-\lambda & 1 & 1 & 1 & 1 \\ 1 & 1-\lambda & 0 & 0 & 1 \\ 1 & 0 & 1-\lambda & 0 & 0 \\ 1 & 0 & 0 & 1-\lambda & 0 \\ 1 & 1 & 0 & 0 & 1-\lambda \end{vmatrix}$$

Expandiendo el determinante, tenemos:

$$-\lambda^5 + 7\lambda^4 - 13\lambda^3 + 7\lambda^2 = 0 \Rightarrow \lambda^2(-\lambda^3 + 7\lambda^2 - 13\lambda^1 + 7) = 0 \Rightarrow \lambda^2(\lambda - 1)(\lambda^2 - 6\lambda + 7) = 0$$

Resolviendo esta ecuación, encontramos los autovalores:

$$\lambda_1 = 3 + \sqrt{2}, \quad \lambda_2 = 3 - \sqrt{2}, \quad \lambda_3 = 1, \quad \lambda_4 = 0, \quad \lambda_5 = 0$$

Para obtener los autovectores de la matriz M, debemos resolver la ecuación característica $M\vec{v} = \lambda\vec{v}$, donde $\vec{v}$ es el autovector y λ es el autovalor correspondiente.

La ecuación característica se puede escribir como:

$$\begin{pmatrix} 3 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{pmatrix} = \lambda \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{pmatrix}$$

Esto se puede reescribir como un sistema de ecuaciones lineales:

$$\begin{aligned} 3v_1 + v_2 + v_3 + v_4 + v_5 &= \lambda v_1 \\ v_1 + v_2 + v_5 &= \lambda v_2 \\ v_1 + v_3 &= \lambda v_3 \\ v_1 + v_4 &= \lambda v_4 \\ v_1 + v_2 + v_5 &= \lambda v_5 \end{aligned}$$

Resolviendo este sistema de ecuaciones, encontramos los autovalores y autovectores correspondientes. Al resolver el sistema, se obtendrán los siguientes resultados:

Autovalor 1 ($\lambda_1 = 3 + \sqrt{2}$):

$$\begin{aligned} v_1 &= 1 + \sqrt{2} \\ v_2 &= 1 \\ v_3 &= \frac{\sqrt{2}}{2} \\ v_4 &= \frac{\sqrt{2}}{2} \\ v_5 &= 1 \end{aligned}$$

Autovalor 2 ($\lambda_2 = 3 - \sqrt{2}$):

$$\begin{aligned} v_1 &= 1 - \sqrt{2} \\ v_2 &= 1 \\ v_3 &= \frac{-\sqrt{2}}{2} \\ v_4 &= \frac{-\sqrt{2}}{2} \\ v_5 &= 1 \end{aligned}$$

Autovalor 3 ($\lambda_3 = 1$):

$$\begin{aligned} v_1 &= 0 \\ v_2 &= 0 \\ v_3 &= -1 \\ v_4 &= 1 \\ v_5 &= 0 \end{aligned}$$

Autovalor 4 ($\lambda_4 = 0$):

$$\begin{aligned} v_1 &= 0 \\ v_2 &= -1 \\ v_3 &= 0 \\ v_4 &= 0 \\ v_5 &= 1 \end{aligned}$$

Autovalor 5 ($\lambda_5 = 0$):

$$\begin{aligned} v_1 &= -1 \\ v_2 &= 1/2 \\ v_3 &= 1 \\ v_4 &= 1 \\ v_5 &= 1/2 \end{aligned}$$

Se observa que hay un autovalor nulo con multiplicidad 2, lo que indica que la matriz M no es de rango completo y tiene una dependencia lineal entre sus columnas.

Por el Lema 3.7 los autovectores con autovalores distintos son ortogonales de manera automática. En el caso de los dos autovectores v_4 y v_5, los dos con autovalor 0, nos podían haber salido no ortogonales. Pero como forman un subespacio vectorial siempre los podemos elegir ortogonales entre sí. De hecho, primero nos salió $v_5 = (-1, 0, 1, 1, 1)$ pero le restamos $\frac{1}{2}v_4$ para que fuera ortogonal a v_4.

Finalmente, normalizamos los autovectores para obtener la matriz V. Para normalizar los vectores dados, simplemente dividimos cada componente del vector por su norma. La norma de un vector de dimensión 5, $\vec{v}$, se calcula como se dijo en la Sección 3.1:

$$\|\vec{v}\| = \sqrt{v_1^2 + v_2^2 + v_3^2 + v_4^2 + v_5^2}$$

Luego, podemos normalizar los vectores como sigue:

Autovalor 1 ($\lambda_1 = 3 + \sqrt{2}$):
$\|\vec{v}\| = \sqrt{6 + 2\sqrt{2}}$

$$\begin{aligned} v_1 &= 0{,}81 \\ v_2 &= 0{,}34 \\ v_3 &= 0{,}24 \\ v_4 &= 0{,}24 \\ v_5 &= 0{,}34 \end{aligned}$$

Autovalor 2 ($\lambda_2 = 3 - \sqrt{2}$):

$\|\vec{v}\| = \sqrt{6 - 2\sqrt{2}}$

$$\begin{aligned} v_1 &= -0{,}23 \\ v_2 &= 0{,}56 \\ v_3 &= -0{,}4 \\ v_4 &= -0{,}4 \\ v_5 &= 0{,}56 \end{aligned}$$

Autovalor 3 ($\lambda_3 = 1$):
$\|\vec{v}\| = \sqrt{2}$

$$\begin{aligned} v_1 &= 0 \\ v_2 &= 0 \\ v_3 &= -0{,}71 \\ v_4 &= 0{,}71 \\ v_5 &= 0 \end{aligned}$$

Autovalor 4 ($\lambda_4 = 0$):
$\|\vec{v}\| = \sqrt{2}$

$$\begin{aligned} v_1 &= 0 \\ v_2 &= -0{,}71 \\ v_3 &= 0 \\ v_4 &= 0 \\ v_5 &= 0{,}71 \end{aligned}$$

Autovalor 5 ($\lambda_5 = 0$):
$\|\vec{v}\| = \sqrt{\frac{7}{2}}$

$$\begin{aligned} v_1 &= -0{,}53 \\ v_2 &= 0{,}56 \\ v_3 &= 0{,}53 \\ v_4 &= 0{,}53 \\ v_5 &= 0{,}56 \end{aligned}$$

Colocando los vectores como filas obtenemos V^*:

$$V^* = \begin{pmatrix} 0{,}81 & 0{,}34 & 0{,}24 & 0{,}24 & -0{,}53 \\ -0{,}23 & 0{,}56 & -0{,}4 & -0{,}4 & 0{,}56 \\ 0 & 0 & -0{,}71 & 0{,}71 & 0{,}53 \\ 0 & -0{,}71 & 0 & 0 & 0{,}53 \\ -0{,}5 & 0{,}5 & 0{,}5 & 0{,}5 & 0{,}56 \end{pmatrix}$$

3.4.3. Cálculo de U

El cálculo de U es muy similar al cálculo de V en cuanto a operaciones. Lo primero que hay que hacer es la matriz $M = AA^*$:

$$M = AA^* = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \end{pmatrix}$$

Para obtener los autovalores de la matriz M, debemos resolver la ecuación $|M - \lambda Id| = 0$, donde λ es el autovalor y Id es la matriz identidad. Sustituyendo los valores de la matriz M, tenemos:

$$|M - \lambda Id| = \begin{vmatrix} 2-\lambda & 1 & 1 \\ 1 & 2-\lambda & 1 \\ 1 & 1-\lambda & 3 \end{vmatrix}$$

Expandiendo el determinante, tenemos:

$$-\lambda^3 + 7\lambda^2 - 13\lambda + 7 = 0 \Rightarrow (\lambda - 1)(\lambda^2 - 6\lambda + 7) = 0$$

Resolviendo esta ecuación, encontramos los autovalores:

$$\lambda_1 = 3 + \sqrt{2}, \quad \lambda_2 = 3 - \sqrt{2}, \quad \lambda_3 = 1$$

Para obtener los autovectores de la matriz M, debemos resolver la ecuación característica $M\vec{v} = \lambda\vec{v}$, donde $\vec{v}$ es el autovector y λ es el autovalor correspondiente.

La ecuación característica se puede escribir como:

$$\begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \lambda \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

Esto se puede reescribir como un sistema de ecuaciones lineales:

$$\begin{aligned} 2v_1 + v_2 + v_3 &= \lambda v_1 \\ v_1 + 2v_2 + v_3 &= \lambda v_2 \\ v_1 + v_2 + 3v_3 &= \lambda v_3 \end{aligned}$$

Resolviendo este sistema de ecuaciones, encontramos los autovalores y autovectores correspondientes. Al resolver el sistema, se obtendrán los siguientes resultados:

Autovalor 1 ($\lambda_1 = 3 + \sqrt{2}$):

$$\begin{aligned} v_1 &= \frac{\sqrt{2}}{2} \\ v_2 &= \frac{\sqrt{2}}{2} \\ v_3 &= 1 \end{aligned}$$

Autovalor 2 ($\lambda_2 = 3 - \sqrt{2}$):

$$v_1 = -\frac{\sqrt{2}}{2}$$
$$v_2 = -\frac{\sqrt{2}}{2}$$
$$v_3 = 1$$

Autovalor 3 ($\lambda_3 = 1$):

$$v_1 = -1$$
$$v_2 = 1$$
$$v_3 = 0$$

Se observa que no hay autovalores nulos, lo que indica que la matriz M es de rango completo y no tiene una dependencia lineal entre sus columnas. Por otro lado, como los tres autovalores son distintos los vectores son ortogonales automáticamene.

Finalmente, normalizamos los autovectores para obtener la matriz U:

Luego, podemos normalizar los vectores como sigue:

Autovalor 1 ($\lambda_1 = 3 + \sqrt{2}$):
$\|\vec{v}\| = \sqrt{2}$

$$v_1 = 0{,}5$$
$$v_2 = 0{,}5$$
$$v_3 = 0{,}71$$

Autovalor 2 ($\lambda_2 = 3 - \sqrt{2}$):
$\|\vec{v}\| = \sqrt{2}$

$$v_1 = -0{,}5$$
$$v_2 = -0{,}5$$
$$v_3 = 0{,}71$$

Autovalor 3 ($\lambda_3 = 1$):
$\|\vec{v}\| = \sqrt{2}$

$$v_1 = -0{,}71$$
$$v_2 = 0{,}71$$
$$v_3 = 0$$

Colocando los vectores como columnas obtenemos U:

$$U = \begin{pmatrix} 0{,}5 & -0{,}5 & -0{,}71 \\ 0{,}5 & -0{,}5 & 0{,}71 \\ 0{,}71 & 0{,}71 & 0 \end{pmatrix}$$

3.4.4. Cálculo de Σ

El cálculo de Σ es bastante directo una vez que conocemos los autovalores, ya que podemos construir la matriz de la siguiente manera:

$$\Sigma = \begin{pmatrix} \sqrt{\lambda_1} & 0 & 0 & 0 & 0 \\ 0 & \sqrt{\lambda_2} & 0 & 0 & 0 \\ 0 & 0 & \sqrt{\lambda_3} & 0 & 0 \end{pmatrix} = \begin{pmatrix} \sqrt{3+\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & \sqrt{3-\sqrt{2}} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

3.5. Realizacion de una recomendación

Con la intención de realizar una recomendación precisa se puede aplicar un filtro colaborativo sobre la descomposición SVD.

En esta sección se pretende mediante la combinación de ambos se logrará una sugerencia que se ajuste mejor a las preferencias individuales de cada usuario utilizando la descomposicion SVD del ejemplo anterior.

Pretendemos recomendar un item a un usuario, por lo tanto nos centraremos en la matriz V^*:

$$V^* = \begin{pmatrix} 0{,}81 & 0{,}34 & 0{,}24 & 0{,}24 & -0{,}53 \\ -0{,}23 & 0{,}56 & -0{,}4 & -0{,}4 & 0{,}56 \\ 0 & 0 & -0{,}71 & 0{,}71 & 0{,}53 \\ 0 & -0{,}71 & 0 & 0 & 0{,}53 \\ -0{,}5 & 0{,}5 & 0{,}5 & 0{,}5 & 0{,}56 \end{pmatrix}$$

Procedemos con la aplicación de un filtro colaborativo

3.5.1. Búsqueda de una métrica

Con la función de similitud pretendemos determinar lo parecidos que son dos elementos. Para ello se selecciona una métrica, en este caso la métrica L_1 que viene dada por:

$$sim(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Donde x e y son vectores de n dimensiones.

Para demostrar que la función dada es una métrica, debemos verificar si cumple con las propiedades requeridas:

- **Positividad definida:** La positividad definida de una métrica viene dada por la siguiente expresión:

$$sim(x, y) \geq 0, \forall x, y \in \mathbb{R}^n$$

La función dada suma los valores absolutos de las diferencias de los elementos correspondientes de los vectores x e y. Los valores absolutos aseguran que el resultado sea no negativo. Por lo tanto, se cumple la positividad definida.

- **Simetría:**
  Para verificar que se cumple la simetría hay que ver:

  $$sim(x, y) = sim(y, x)$$

  Desarrollando la función obtenemos:

  $$sim(x, y) = \sum_{i=1}^{n} |x_i - y_i| = \sum_{i=1}^{n} |-1| \cdot |y_i - x_i| = \sum_{i=1}^{n} |y_i - x_i| = sim(y, x)$$

  Por lo tanto se verifica la simetría.

- **Axioma de coincidencia:** Para probar el axioma de coincidencia es necesario que se cumpla:

  $$sim(x, y) = 0 \Leftrightarrow x = y$$

  Supongamos que tenemos dos vectores x e y, donde $x = (x_1, x_2, ..., x_n)$ e $y = (y_1, y_2, ..., y_n)$. Si $x = y$, esto implica que cada componente correspondiente de x e y es idéntico, es decir, $x_i = y_i$ para todo i.
  Entonces, para cada término en la suma de la función $sim(x, y)$, tenemos:

  $$|x_i - y_i| = |x_i - x_i| = |0| = 0$$

  Por lo tanto, $sim(x, y) = \sum_{i=1}^{n} |x_i - y_i| = \sum_{i=1}^{n} 0 = 0$

  Esto demuestra que si $x = y$, entonces $sim(x, y) = 0$.

  Por otro lado, supongamos que $sim(x, y) = 0$. Esto significa que cada término individual en la suma de la función es igual a cero, es decir, $|x_i - y_i| = 0, \forall i, 1 \leq i \leq n$.

  La única forma en que el valor absoluto de una diferencia sea igual a cero es cuando los dos valores son iguales. Por lo tanto, $x_i = y_i$ para todo i.

  Esto implica que todos los componentes de x son iguales a los componentes correspondientes de y, es decir, $x = y$.

  En conclusión, hemos demostrado que la función $sim(x, y) = \sum_{i=1}^{n} |x_i - y_i|$ cumple con el axioma de coincidencia. La métrica es igual a cero si y solo si los dos elementos son idénticos.

- **Desigualdad triangular:**
  La desigualdad triangular de una métrica consiste en lo siguiente:

  $$sim(x, z) \leq sim(x, y) + sim(y, z), \forall x, y, z \in \mathbb{R}^n$$

  Consideremos tres vectores $x, y, z \in \mathbb{R}^n$.

  Las funciones dadas son:

  $$sim(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

  $$sim(y, z) = \sum_{i=1}^{n} |y_i - z_i|$$

  $$sim(x, z) = \sum_{i=1}^{n} |x_i - z_i|$$

  Para demostrar la desigualdad triangular, necesitamos probar que:

  $$sim(x, z) \leq sim(x, y) + sim(y, z)$$

  Usando la desigualdad triangular para el valor absoluto, sabemos que:

  $$|a + b| \leq |a| + |b|$$

  Aplicando esta desigualdad a cada término de la suma, obtenemos:

  $$|(x_i - y_i) + (y_i - z_i)| \leq |x_i - y_i| + |y_i - z_i| \Rightarrow |x_i - z_i| \leq |x_i - y_i| + |y_i - z_i|$$

  Sumando ambos lados de la desigualdad para $i = 1$ a n, obtenemos:

  $$\sum_{i=1}^{n} |x_i - z_i| \leq \sum_{i=1}^{n} |x_i - y_i| + \sum_{i=1}^{n} |y_i - z_i|$$

  Lo cual es equivalente a:

  $$sim(x, z) \leq sim(x, y) + sim(y, z)$$

  Por lo tanto, se cumple la desigualdad triangular.

En resumen, hemos demostrado que la función $sim(x, y)$ cumple con las propiedades requeridas para ser considerada como una métrica.

3.5.2. Búsqueda de vecinos

Para obtener las relaciones entre ítems, calculamos la similitud entre pares mediante la métrica anterior y elaboramos la siguiente tabla:

$sim(i_x, i_y)$	i_1	i_2	i_3	i_4	i_5
i_1	0	2.74	2.78	2.78	2.69
i_2	2.74	0	2.82	2.82	1.76
i_3	2.78	2.82	0	1.41	2.57
i_4	2.78	2.82	1.41	0	2.57
i_5	2.69	1.76	2.57	2.57	0

Para cada elemento, buscamos su vecino más cercano, es decir, el elemento distinto de sí mismo con el valor más bajo. Esto da lugar a las siguientes relaciones:

Usuario	i_1	i_2	i_3	i_4	i_5
Vecino	i_5	i_5	i_4	i_3	i_2

Una vez hemos calculado las relaciones podemos proceder a realizar una recomendación.

3.5.3. Recomendación

Utilizando la tabla de relaciones anterior, podemos hacer recomendaciones a cada usuario en función de los elementos con los que han interactuado. Es decir, si un usuario ha interactuado con un cierto elemento, se le recomendará el elemento que sea más similar a este último.

Por ejemplo, si un usuario interactúa con el item i_3, se le recomendará el item i_4. Sin embargo, puede ocurrir que el usuario ya haya interactuado con el item en el pasado. En este caso, no tendría sentido recomendar un elemento con el que ya ha interactuado previamente. Por lo tanto, se le recomendará el siguiente item con la puntuación más baja.

Capítulo 4

Implementación en el portal

Los filtros colaborativos son una herramienta altamente poderosa y valiosa en la actualidad. Por lo tanto, resulta muy interesante implementarlos en un portal que será utilizado posteriormente por una empresa.

4.1. Conceptos iniciales

En esta sección, exploraremos algunos conceptos que están menos relacionados con los filtros colaborativos. El objetivo es obtener una comprensión más completa del trabajo realizado y la plataforma en la que se trabajará.

4.1.1. Qué es un CMS

Un CMS o *Content Management System* es un sistema online de gestión de contenidos con el que crear un repositorio web. Consiste en un *software* que ayuda a gestionar de forma rápida y sencilla contenidos estáticos como usuarios o datos.

Una de las funcionalidades principales de los CMS es la facilidad para crear contenidos en la web, ya que se encarga de todo el proceso de subida de la información en la página. De esta manera se consigue que el usuario sólo tenga que publicar el contenido y aparecerá automáticamente en la web.

4.1.2. Qué es Drupal

Drupal es un CMS basado en LAMP (acrónimo de Linux, Apache, MySQL y PHP, herramientas que esta infraestructura emplea), que permite la instalación de funcionalidades mediante módulos y cuya apariencia puede ser modificada mediante temas. El lenguaje empleado en este sistema es php y es combinable con MySQL.

4.2. Entidades creadas

Una entidad de Drupal es una unidad de información estructurada en campos. Cada entidad tiene sus propias propiedades y gestiona tipos disferentes de información, por

ejemplo, existe la entidad *User* que se encarga de la información de los usuarios y cada vez que un usuario se registra Drupal almacena su información en forma de entidad *User*.

Las principales entidades usadas en este proyecto son las siguientes: contenidos, multimedia y taxonomías.

4.2.1. Contenidos

Este tipo de entidad suele ser utilizado para gestionar la información de los contenidos de la página como por ejemplo, artículos o en nuestro caso las APIs. Existe una gran libertad a la hora de definir nuevos tipos de contenido en Drupal, lo que hace una tarea sencilla la creación de una plantilla para las APIs.

Los campos son utilizados para estructurar las información y cada uno puede tener un tipo de valor distinto. Dicho valor se establece al crear el nuevo tipo de contenido. Así una API en el portal tendría la siguiente forma:

- **title:** Campo del tipo texto sin formato con el título de la API.
- **description:** Texto que puede contener o no formato html, es una descripción de las funcionalidades de la API.
- **category:** Campo del tipo término de taxonomía del vocabulario *categorias_api* que identifica la API con esta categoría.
- **subcategory:** Campo del tipo término de taxonomía del vocabulario *categorias_api* que identifica la API con esta subcategoría.
- **yaml:** Campo del tipo archivo. Este es el archivo que ha subido el usuario al crear la API; este campo solo admite archivos de tipo *yaml* e *yml*.
- **downloads:** Campo del tipo lista de texto. En él se almacenan los ids de los usuarios que han descargado l especificación.

4.2.2. Taxonomías.

Drupal proporciona otro tipo de entidad, las taxonomías. Estas son utilizadas para organizar los contenidos por palabras clave respecto a un vocabulario. Por ello, se ha creado un vocabulario llamado *categorias_api* para la categorización de las APIs.

En este vocabulario se establece una jerarquía arbólea de padres e hijos entre categoría y subcategoría, evitando así que una API termine siendo catalogada en una categoría que no le corresponde. Por ejemplo, se tienen dos APIs, una con *categoría1* y *subcategoría1* y otra con categoría *categoría2* y subcategoría *subcategoría1*. En este caso, si no existiera una jerarquía de padres e hijos en el vocabulario, Drupal detectaría que ambas APIs pertenecen a la misma subcategoría, lo que no tendría sentido porque al pertenecer a distintas categorías no deberían pertenecer a la misma subcategoría.

Con las jerarquías se consigue que esto no ocurra ya que se podrán diferenciar dos subcategorías con el mismo nombre en función de sus padres.

4.3. Métodos para crear APIs

Con el objetivo de convertir un archivo YAML o JSON en un tipo de contenido de Drupal, se han creado algunos métodos que gestionan la información y crean el contenido. A continuación, se presentan algunos de ellos.

4.3.1. Clase ApiProcessor

Para llevar a cabo el procesamiento de los archivos YAML de las APIs, se ha creado la clase *ApiProcessor*. La idea es que los procesos encargados de la creación y edición de APIs utilicen los siguientes métodos: *newApi*, *editApi* y *taxonomyComplete*.

4.3.2. Método newApi

Este método recibe como parámetro el archivo YAML que contiene la información de la API que se pretende crear.

Para realizar esta acción, el contenido del archivo se convierte en un array utilizando el método *Yaml::parseFile*. A continuación, se crean los términos de taxonomía para la categorización mediante el método *taxonomyComplete*, que analizaremos más adelante. Luego, se verifica si existe una definición de la API en el YAML y, en caso contrario, se cambia el valor por "API sin descripción".

Una vez se dispone de toda la información necesaria para crear una API, se construye un array con dicha información y se utiliza el gestor de nodos que proporciona Drupal para crear la API y guardarla, *\Drupal::EntityTypeManager*.

Finalmente, se devuelve la entidad creada. Si se produce algún error durante la ejecución del código, se elimina el archivo de la memoria y se muestra un mensaje de error por pantalla.

4.3.3. Método taxonomyComplete

En este método se busca crear un nuevo término de taxonomía si este aún no existe. Para ello, se compara el valor del nombre y el valor del padre del término. Los parámetros de este método son una cadena de texto que indica el vocabulario al que pertenece el término, otra cadena de texto con el valor del término y un entero con el ID del padre del término. Si el valor del término es nulo, se asume el valor .ºtra"para dicho término, ya que todas las APIs deberían ser categorizadas.

Posteriormente, se cargan todos los hijos del término padre. Si entre ellos hay algún término cuyo valor coincida con el término que se está buscando, se retorna ese término. En caso contrario, se crea uno nuevo y se retorna.

4.4. Creación de una API desde la web

Con el propósito de proporcionar al usuario un bloque en el que pueda subir una especificación y generar automáticamente el contenido de una API, se ha creado el módulo Upload Yaml. Este módulo ofrece al usuario un formulario en el que puede arrastrar o seleccionar un archivo de su PC local para crear una API. El esquema de archivos y directorios del módulo es el siguiente:

```
upload_yaml
├── src
│   ├── Form
│   │   └── UploadYamlForm.php
│   └── Plugin
│       └── Block
│           └── UploadYamlBlock.php
└── upload_yaml.info.yml
```

Figura 4.1: Esquema del módulo upload_yaml.

4.4.1. Uploadyaml.info.yml

Este archivo proporciona a Drupal la información sobre el módulo, como el título, la descripción de su funcionalidad y los requerimientos necesarios para su funcionamiento.

4.4.2. UploadYamlForm.php

En este código se describe el comportamiento del formulario y cómo gestiona los datos recibidos. El formulario solo necesita recibir un archivo, por lo que en el método *buildForm* se establece que el valor recibido debe ser de tipo *managed file*, además de aplicar filtros de extensión de archivo (*.yml* y *.yaml*).

A continuación, se crea el botón que el usuario utilizará para indicar que desea procesar el archivo una vez completado el formulario.

Para indicar qué hacer con la información del formulario, se utiliza la clase *submitForm*, que se ejecuta cuando el usuario presiona el botón del formulario. En este método, se realiza una llamada al método *newApi* para crear la nueva API y se recibe la entidad creada, de la cual se obtiene su ID para redirigir al usuario al nodo creado.

4.4.3. UploadYamlBlock.php

En este código se crea el bloque que contendrá el formulario de subida de la especificación. Para mostrar el formulario en la página, se crea un bloque y se establece la página, la región y los usuarios para los que debe ser mostrado el formulario. Este formulario queda así:

Figura 4.2: Formulario para subir un archivo

En resumen, el módulo Üpload Yaml"proporciona al usuario un formulario que le permite subir un archivo de especificación y generar automáticamente el contenido de una API.

4.5. Crear una API mediante instrucciones HTTP

Dadas las ventajas de utilizar APIs, resulta interesante implementar el propio portal como una API, lo que permite a los usuarios interactuar con él mediante solicitudes HTTP, incluyendo la capacidad de crear contenido en el portal. Para lograr esto, se crea un endpoint específico en el portal que acepta solicitudes POST. Cuando un usuario envía un archivo en el cuerpo de la solicitud, este archivo se carga en el portal y se utiliza el método ApiProcessor para crear contenido basado en el mismo. Esto significa que el usuario puede enviar archivos al portal y generar contenido automáticamente a través de la API, lo que mejora la capacidad de personalización y contribución de los usuarios al portal.

4.6. Implementación de filtro colaborativo con SVD

Con la intención de ofrecer a cada usuario una experiencia personalizada se implementará en el portal un filtro colaborativo combinado con la técnica SVD (Singular Value Decomposition) y así poder mostrar a cada usuario un mensaje con las APIs relacionadas al descargar una especificación.

Los pasos para generar la recomendación son los siguientes:

1. **Preparación de los datos:** Se crea una matriz en la que las filas representan usuarios y las columnas representan APIs. Para acortar el tiempo de ejecución y reducir la complejidad temporal, se tomará únicamente una muestra acotada de datos seleccionados en funcion de su id dentro del portal, es decir, una cantidad fija de usuarios y APIs.

2. **Calcular descomposición SVD:** Hay que descomponer la matriz de datos con el método SVD. Así obtendremos las tres matrices de la descomposición. En nuestro caso, nos centraremos en V, ya que esta contiene la información sobre las APIs.

3. **Búsqueda de vecinos:** Se ordenarán las APIs en función de su similitud respecto a la especificación descargada. Esta similitud vendrá dada por una métrica.

4. **Recomendación final:** Se mostrará un mensaje referenciando al usuario las dos APIs más similares a la especificación.

A continuación, procederemos a la explicación de la implementación de estos pasos en el portal.

4.6.1. Preparación de los datos

La implementación de un filtro colaborativo en un portal de Drupal implica la preparación de los datos para su análisis y utilización en el proceso de recomendación. En este caso, para poder aplicar el filtro colaborativo, se construirá una matriz que organice los datos del portal. Esta matriz tendrá una estructura en la que cada columna representará una API y cada fila representará un usuario.

Cada celda de la matriz contendrá información sobre la interacción de un usuario con una API en particular. Por ejemplo, si un usuario ha utilizado una API específica, se marcará esa celda con un valor que indique la interacción o el nivel de preferencia del usuario hacia esa API.

La construcción de esta matriz implica recopilar y procesar los datos del portal de Drupal. Esto puede incluir el análisis de registros de actividad de usuarios, historiales de navegación, acciones realizadas en el portal (como búsquedas, descargas o valoraciones), entre otros datos relevantes.

En nuestro caso, representaremos la interacción realizada por el usuario como la descarga de una especificación de API. Cada descarga se representará con un valor de 1 en la matriz, indicando que el usuario ha descargado esa especificación. Por otro lado, si el usuario no ha descargado una especificación, se registrará un valor de 0 en la matriz para esa entrada correspondiente.

Cabe destacar que en este contexto no tendría sentido recomendar al usuario una API que ya se ha descargado, por lo tanto, se descartarán las APIs con las que ya se ha interactuado

Una vez que hemos construido la matriz, podemos aplicar la descomposición SVD para obtener sus valores más relevantes.

4.6.2. Calcular descomposición SVD

La descomposición SVD nos permite descomponer la matriz en tres componentes principales: la matriz de vectores singulares izquierdos, la matriz de valores singulares y la matriz de vectores singulares derechos. Estos componentes nos proporcionan información valiosa sobre la estructura y las propiedades de la matriz original.

La descomposición de esta matriz puede realizarse utilizando el paquete de PHP llamado "math-php", el cual ofrece una manera sencilla de calcular la descomposición SVD de una matriz dada. Este paquete proporciona funciones específicas que permiten realizar operaciones matemáticas, incluyendo la descomposición SVD.

Una vez calculada la descomposición SVD de la matriz, es posible buscar los dos vecinos más similares a la API específica que ha sido descargada.

4.6.3. Búsqueda de los vecinos

Para determinar la similitud entre pares de APIs es necesario definir una métrica. En este caso la métrica viene dada por la siguiente relación:

$$sim(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Donde x e y son vectores de n dimensiones.

Como se demostró en la sección 3.5.1, la función $sim(x, y)$ es una métrica, lo que significa que puede utilizarse para determinar la similitud entre APIs. En este contexto, cuanto menor sea el valor de $sim(x, y)$, es más probable que las APIs sean más similares.

Nuestro objetivo es realizar recomendaciones basadas en la API descargada, por lo tanto, no es necesario buscar los vecinos de todas las APIs. En su lugar, podemos enfocarnos en encontrar los vecinos más cercanos a la API descargada. Para esto, utilizamos la función de similitud mencionada anteriormente y calculamos el valor de similitud entre la API descargada y cada una de las demás APIs. Luego, seleccionamos los dos vecinos con los valores de similitud más altos, ya que son los más cercanos a la API descargada. Estos dos vecinos representan las APIs más similares y pueden ser considerados como recomendaciones relevantes para el usuario en función de su interacción previa.

Una vez se han encontrado las APIs más similares se puede realizar la recomendación final.

4.6.4. Recomendación final

La recomendación final se lleva a cabo mediante la presentación de sugerencias de nuevas APIs al usuario, basadas en la similitud con la API previamente descargada.

Para lograr esto, se muestra un mensaje en el portal que incluye los nombres de las dos APIs más relevantes, los cuales están enlazados con sus respectivas páginas de contenido. De esta manera, el usuario puede acceder rápidamente a las APIs recomendadas y explorar más detalles sobre su funcionalidad y uso. Esta estrategia de presentación de la recomendación final a través del portal facilita la interacción del usuario con las APIs sugeridas y fomenta su exploración y utilización de las mismas.

El mensaje de recomendación final sería:

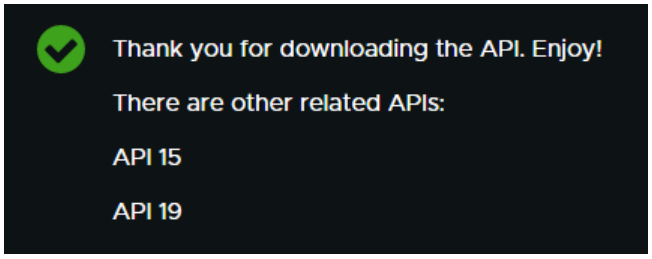

Figura 4.3: Mensaje de recomendación para el usuario.

Capítulo 5

Conclusiones y lineas futuras

5.1. Sistemas de recomendación

La implementación de un filtro colaborativo en un marketplace de APIs ofrece numerosas ventajas para brindar a los usuarios una experiencia personalizada y mejorar la relevancia de las recomendaciones. Al combinar esta técnica con la descomposición SVD (Singular Value Decomposition), se pueden obtener resultados aún más precisos y efectivos.

Hemos podido observar que los filtros colaborativos son de gran utilidad en la actualidad. Por ello algunas de sus principales ventajas son:

- **Personalización:** Los filtros colaborativos permiten recomendar APIs de acuerdo con las preferencias y comportamientos de cada usuario, lo que proporciona una experiencia única y adaptada a sus necesidades específicas.
- **Descubrimiento de APIs relevantes:** Al analizar las descargas y patrones de comportamiento de los usuarios, los filtros colaborativos pueden ayudar a descubrir APIs relevantes que de otro modo podrían pasar desapercibidas.
- **No se requiere información detallada de los usuarios:** A diferencia de otros enfoques, los filtros colaborativos no necesitan información detallada sobre los usuarios. Se basan en patrones de comportamiento colectivo y no en datos personales sensibles.

Sin embargo hay ciertos escenarios en los que un filtro colaborativo puede dar problemas o incluso tener dificultades algunas a la hora de recomendar. Algunas de sus desventajas principales serían:

- **Problema de arranque en frío:** Los filtros colaborativos pueden enfrentar dificultades al recomendar para nuevos usuarios o nuevas APIs, ya que no hay suficiente información disponible para establecer relaciones de similitud.
- **Sesgo de popularidad:** Existe el riesgo de que las recomendaciones se vean sesgadas hacia APIs populares o descargadas con mayor frecuencia, lo que puede limitar la diversidad de las recomendaciones.

La combinación de filtros colaborativos con la descomposición en valores singulares (SVD) ofrece numerosas ventajas y puede proporcionar resultados significativos en diversas aplicaciones. A continuación, se enumeran algunas de las ventajas clave de esta combinación:

- **Mejora de la precisión:** La descomposición SVD permite capturar patrones subyacentes en los datos de las descargas y mejorar la precisión de las recomendaciones.
- **Manejo de la esparcidad de los datos:** La descomposición SVD ayuda a lidiar con la esparcidad de los datos al reducir la dimensionalidad y encontrar representaciones más compactas de las preferencias de los usuarios.
- **Escalabilidad:** Al utilizar la descomposición SVD, es posible realizar cálculos eficientes incluso en conjuntos de datos grandes y en crecimiento constante.

Sin embargo, también es importante tener en cuenta las posibles desventajas de la combinación de filtros colaborativos con la descomposición en valores singulares (SVD). A continuación, se mencionan algunas de las posibles limitaciones:

- **Requerimientos de almacenamiento y procesamiento:** La descomposición SVD puede requerir recursos significativos de almacenamiento y procesamiento, especialmente para conjuntos de datos grandes.
- **Sensibilidad a cambios en los datos:** La descomposición SVD puede no ser adecuada si los patrones de descarga cambian con frecuencia, ya que puede llevar tiempo y recursos volver a calcular la descomposición.

En conclusión, implementar un filtro colaborativo en un marketplace de APIs brinda la oportunidad de ofrecer una experiencia personalizada y relevante a los usuarios. La combinación con la descomposición SVD puede mejorar la precisión de las recomendaciones y lidiar con la esparcidad de los datos. Sin embargo, se deben considerar las posibles limitaciones, como el problema de arranque en frío y el sesgo de popularidad, así como los requisitos de almacenamiento y la sensibilidad a los cambios en los datos al utilizar la descomposición SVD.

5.2. Implementación en el portal

La gestión de la información en la nube es crucial en la actualidad, lo que ha llevado a un aumento en la creación de marketplaces de APIs por parte de las empresas. Estos marketplaces desempeñan un papel fundamental al permitir a los desarrolladores de software trabajar con datos y establecer una comunicación rápida, segura y sencilla entre aplicaciones. El marketplace de APIs desarrollado para la empresa Axpe Consulting se convierte en un entorno de gran utilidad, ya que se espera que se utilice en el futuro y, por lo tanto, el portal se seguirá actualizando y mejorando.

En este contexto, Drupal ha demostrado ser un CMS potente para crear portales de gestión de información. Ofrece flexibilidad en el diseño de páginas y en la definición

de funcionalidades, y su sistema de módulos facilita la creación de un entorno web que se ajuste a las necesidades específicas. La facilidad para desarrollar módulos personalizados o personalizar módulos existentes en Drupal garantiza que el sitio web cumpla exactamente con los requisitos buscados, sin complicaciones significativas.

Además, es de suma importancia implementar sistemas de recomendación en el portal del marketplace de APIs. Estos sistemas aportan una experiencia personalizada a los usuarios, ayudándoles a descubrir y seleccionar las APIs más relevantes para sus necesidades.

La combinación de filtros colaborativos y SVD en los sistemas de recomendación ofrece varias ventajas. Por un lado, permite capturar patrones ocultos en los datos y generar recomendaciones más precisas y relevantes para cada usuario. Al analizar la similitud entre usuarios y utilizar la descomposición SVD para extraer características latentes, se obtienen recomendaciones más personalizadas y acertadas.

En conclusión, la implementación de un marketplace de APIs y la combinación de filtros colaborativos con la descomposición SVD en los sistemas de recomendación brindan una solución potente y efectiva. Permite a los usuarios descubrir APIs en función de sus posibles gustos, al tiempo que proporciona a la empresa una plataforma versátil para la gestión de información. Si bien existen desafíos técnicos y de datos, los beneficios de ofrecer una experiencia personalizada y mejorar la utilización de las APIs hacen que esta combinación sea valiosa para los marketplaces de APIs.

Bibliografía

[1] Kenneth Hoffman y Ray Kunze, Álgebra lineal, Prentice-Hall Hispanoamericana, 1973

[2] Antonio Sala, Modelado, Análisis, Identificación y Control de Sistemas Multivariables. Universitat Politècnica de València, 2023. Disponible en `http://personales.upv.es/asala/DocenciaOnline/Cursos/Apuntes.html`

[3] Denis Serre, Matrices: Theory and Applications, Springer-Verlag, New York, 2002.

[4] "Curso JSON:API", Escuela Drupal. URL: `https://escueladrupal.com/curso/jsonapi`

[5] "¿Qué es una API REST?" IBM. URL: `https://www.ibm.com/es-es/cloud/learn/rest-apis`

[6] "Taxonomía: Categorización de contenidos con Drupal 7". Tutorial Drupal, 2015. URL: `https://tutorial-drupal.com/taxonomia-categorizacion-de-contenidos-con-drupal-7`

[7] "What is a rest api". RedHat, 2020. URL: `https://www.redhat.com/es/topics/api/what-is-a-rest-api#rest`

[8] "Undestanding Drupal". Drupal. URL: `https://www.drupal.org/es/docs/user_guide/es/understanding-drupal.html`

Apéndices

Apéndice A

Módulo Upload Yaml

A.1. Upload_yaml.info.yml

```
name: Bloque Formulario Para Subir Yaml
description: 'Bloque Formulario Subir Yaml'
package: custom

type: module
core_version_requirement: ^8.8.0 || ^9.0
```

A.2. UploadYamlBlock.php

```
<?php

namespace Drupal\upload_yaml\Plugin\Block;

use Drupal\Core\Block\BlockBase;

/**
 * @Block(
 *   id = "Drupal_block",
 *   admin_label = "Drupal Block",
 * )
 */

class UploadYamlBlock extends BlockBase{

  /**
   * {@inheritdoc }
   */
  public function build(){

    return [
```

```
22        \Drupal::formBuilder()->getForm('\Drupal\upload_yaml\Form\
             UploadYamlForm'),
23        '#cache' => [
24          'max-age' => 0,
25        ],
26
27      ];
28
29    }
30  }
```

A.3. UploadYamlForm.php

```
<?php

namespace Drupal\upload_yaml\Form;

use Drupal\Core\Form\FormBase;
use Drupal\Core\Form\FormStateInterface;
use Drupal\taxonomy\Entity\Term;
use Symfony\Component\HttpFoundation\RedirectResponse;

/**
 * @file
 * Formulario Yaml Form
 */
class UploadYamlForm extends FormBase {
  /**
   * {@inheritdoc }
   */
  public function getFormId() {
    return 'upload_yaml_form';
  }

  public function buildForm(array $form, FormStateInterface
      $form_state) {
    $form['upload']['yaml'] = [
      '#type'                 => 'managed_file',
      '#upload_location'      => 'public://',
      '#multiple'             => false,
      '#description'          => t('Extensiones permitidas: yaml
         yml'),
      '#upload_validators'    => [
        'file_validate_extensions'    => array('yaml yml')
      ],
      '#title'                => t('Subir un archivo yaml para
         crear una nueva API')
      ];

    $form['actions']['#type'] = 'actions';
    $form['actions']['submit'] = [
      '#type' => 'submit',
      '#button_type' => 'primary',
      '#default_value' => $this->t('Save') ,
    ];
    return $form;
  }

  /**
   * {@inheritdoc }
   */
```

```
46    public function submitForm(array &$form, FormStateInterface
         $form_state) {
47      $file_data = $form_state->getValue(['upload' => 'yaml']);
48      $file = \Drupal\file\Entity\File::load( $file_data[0] );
49      $file_name = $file->getFilename();
50      $file->setTemporary();
51      $file->save();
52      $api = $this->newApi($file);
53      isim($api){
54        (new RedirectResponse('/node/' . ($api->id())))->send();
55        exit();
56      }
57    }
58  }
```

Apéndice B

Métodos de recomendación

B.1. Metodo main de recomendación (Crear conjunto de datos)

```
1     /**
2      * Performs the Singular Value Decomposition (SVD) on the node.
3      *
4      * @param integer $nodeID
5      *   The ID of the node.
6      */
7     public function performSVD($nodeID)
8     {
9       // Load APIs
10      $apiIDs = [];
11      $apiQuery = \Drupal::entityQuery('node')
12        ->condition('type', 'api')
13        ->accessCheck(FALSE);
14      $apisAux = $apiQuery->execute();
15
16      $storageHandler = \Drupal::entityTypeManager()->getStorage('
          node');
17      $apis = $storageHandler->loadMultiple($apisAux);
18
19      // Load Users
20      $entityTypeManager = \Drupal::entityTypeManager();
21      $users = $entityTypeManager->getStorage('user')->loadMultiple
          ();
22
23      // Create the data matrix
24      $matrix = [];
25      $i = 0;
26      $j = 0;
27      $k = 12;
28      foreach ($users as $user) {
29        if ($i != 0) {
30          foreach ($apiIDs as $id) {
```

```
31            if ($j < $k && $id != $nodeID) {
32              $api = $apis[$id];
33              if (in_array($user->uid->value, $this->getArray($api
                  ->field_downloads))) {
34                $matrix[$i - 1][$j] = 1;
35              } else {
36                $matrix[$i - 1][$j] = 0;
37              }
38              $j++;
39            }
40          }
41        } else {
42          foreach ($apis as $api) {
43            $apiIDs[] = $api->nid->value;
44          }
45        }
46        $i++;
47        $j = 0;
48      }
49      // Add the downloaded API to the matrix
50      $i = 0;
51      foreach ($users as $user) {
52        if ($i != 0) {
53          $api = $apis[$nodeID];
54          if (in_array($user->uid->value, $this->getArray($api->
              field_downloads))) {
55            $k = count($matrix[$i - 1]);
56            $matrix[$i - 1][$k] = 1;
57          } else {
58            $k = count($matrix[$i - 1]);
59            $matrix[$i - 1][$k] = 0;
60          }
61        }
62        $i++;
63      }
64      $apiIDs[] = $nodeID;
65      $downloadedIndex = $k; // Index of the downloaded API
66
67      // Obtain the SVD decomposition of A
68      $A = MatrixFactory::create($matrix);
69      $SVD = $A->svd();
70      $V   = $SVD->V;
71
72      // Obtain a list of the most similar APIs
73      $result = $this->getNeighbors($downloadedIndex, $V);
74
75      // Array with the id in the keys and the distance in the
          value
76      $ids = [];
77      for ($i = 0; $i < count($result); $i++) {
78        $ids[$apiIDs[$i]] = $result[$i];
79      }
```

```
80
81      // Reverse order of the array of distances
82      arsort($ids);
83
84      // Total number of recommendations
85      $totalRecommends = 2;
86
87      // Recommend APIs to the user
88      $this->showRecomendations($ids, $totalRecommends);
89    }
```

B.2. Metodo de obtención de vecinos

```
1     /**
2      * Returns an array of distances from a given node to all other
          nodes in the matrix.
3      *
4      * @param integer $nodeID
5      *   The ID of the node.
6      * @param Matrix $matrix
7      *   The matrix.
8      *
9      * @return array
10     *   The array of distances.
11     */
12    public function getNeighbors($nodeID, $matrix)
13    {
14      $results = [];
15      for ($row = 0; $row < $matrix->getM(); $row++) {
16        $dist = 0;
17        for ($col = 0; $col < $matrix->getN(); $col++) {
18          $dist += abs($matrix[$row][$col] - $matrix[$nodeID][$col
              ]);
19        }
20        $results[] = $dist;
21      }
22      return $results;
23    }
```

B.3. Método para mostrar la recomendación

```
1     public function showRecomendations($ids, $totalRecommends)
2     {
3       \Drupal::messenger()->addMessage('Thank you for downloading
          the API. Enjoy!');
```

```
4       \Drupal::messenger()->addMessage('There are other related
            APIs:');
5
6       $i = 0;
7       foreach (array_keys($ids) as $id) {
8         if ($i < $totalRecommends) {
9           $node = \Drupal\node\Entity\Node::load($id);
10          $url = Url::fromRoute('entity.node.canonical', ['node' =>
                $id]);
11          $fullUrl = Link::fromTextAndUrl($node->getTitle(), $url)
               ->toString();
12          \Drupal::messenger()->addMessage($fullUrl);
13        }
14        $i++;
15      }
16    }
```

Apéndice C

Reglas personalizadas

C.1. Clase ApiProcessor

```
<?php

namespace Drupal\rules\Plugin\RulesDataProcessor;

use Drupal\taxonomy\Entity\Term;

class ApiProcessor{
  /**
   * Creates a new API with the data of the file.
   *
   * @param File yaml with the api data
   */
  public static function newApi($file){
    // Convertir el file a un array
    $data =  \Symfony\Component\Yaml\Yaml::parseFile($file ->
        getFileUri());
    if (array_key_exists('info', $data) && array_key_exists('
        title', $data['info'])) {
      try{
      // Crear terminos de taxonomia para las categorias
      $term_category = $data['x-catalogation']['category'];
      $term_subcategory = $data['x-catalogation']['subcategory'];
      $category = ApiProcessor::taxonomyComplete('categorias_api'
          ,$term_category,0);
      $subcategory = ApiProcessor::taxonomyComplete('
          categorias_api',$term_subcategory,$category -> tid ->
          value);

      // Comprobar si existe una descripcion
      $description;
      if (array_key_exists('description', $data['info'])) {
        $description = $data['info']['description'];
      }else{
```

```
29          $description = 'API sin descripcion';
30        }
31
32        // Datos de la api
33        $apiData = [
34          'type' => 'api',
35          'title' => $data['info']['title'],
36          'field_descripcion_api' => $description,
37          'field_categoria' => $category,
38          'field_subcategoria' => $subcategory,
39          'field_yaml' => $file
40        ];
41
42        $api = \Drupal::entityTypeManager()
43           -> getStorage('node')
44           -> create($apiData);
45        $api ->save();
46
47        return $api;
48      }catch(Exception $e){
49        \Drupal::messenger()->addError('El formato del yaml es
             incorrecto');
50        $file -> delete();
51      }
52      }
53      \Drupal::messenger()->addError('El formato del yaml no es
           correcto');
54      $file -> delete();
55
56      return null;
57    }
58
59      /**
60     * Edits the API with of ID.
61     *
62     * @param nodeID ID of the node
63     */
64    public static function editApi($nodeID){
65      $node = \Drupal\node\Entity\Node::load($nodeID);
66      $fid = $node -> field_yaml -> target_id;
67      $file = \Drupal\file\Entity\File::load($fid);
68      // Convertir el file a un array
69      $data =  \Symfony\Component\Yaml\Yaml::parseFile($file ->
           getFileUri());
70      if ($file && array_key_exists('info', $data) &&
           array_key_exists('title', $data['info'])) {
71        // Crear terminos de taxonomia para las categorias
72        $term_category = $data['x-catalogation']['category'];
73        $term_subcategory = $data['x-catalogation']['subcategory'];
74        $category = ApiProcessor::taxonomyComplete('categorias_api'
             ,$term_category,0);
```

```
75        $subcategory = ApiProcessor::taxonomyComplete('
             categorias_api',$term_subcategory,$category -> tid ->
             value);
76
77        // Comprobar si existe una descripcion
78        $description;
79        if (array_key_exists('description', $data['info'])) {
80          $description = $data['info']['description'];
81        }else{
82          $description = 'API sin descripcion';
83        }
84
85        // Datos de la api
86        $apiData = [
87          'type' => 'api',
88          'title' => $data['info']['title'],
89          'nid' => $nodeID,
90          'field_descripcion_api' => $description,
91          'field_categoria' => $category,
92          'field_subcategoria' => $subcategory,
93          'field_yaml' => $file
94        ];
95
96
97        \Drupal\node\Entity\Node::load($nodeID)->delete();//
             Eliminar el nodo antiguo
98
99        $api = \Drupal::entityTypeManager()
100          -> getStorage('node')
101          -> create($apiData);
102       $api ->save();
103
104       return $node;
105     }
106     \Drupal::messenger()->addError('El formato del yaml no es
           correcto');
107     //$file -> delete();
108
109     return null;
110   }
111
112
113   /**
114    * Creates a new taxonomy term if none exists.
115    *
116    * @param String $vocabulary
117    *   Vocabulary of the term.
118    * @param String $term
119    *   Taxonomy term.
120    * @param int $parentID
121    *   ID of the term parent.
122    *
```

```
123     */
124    public static function taxonomyComplete($vocabulary, $term,
          $parentID){
125      isim($term == null){
126        $term = "Otras";
127      }
128
129      $category = null;
130
131      $term_array = \Drupal::entityTypeManager()->getStorage('
           taxonomy_term')->loadTree($vocabulary, $parentID, 1, TRUE)
           ;
132
133      foreach($term_array as $termAux){
134        isim($termAux -> get('name') -> value == $term){
135          $category = $termAux;
136        }
137      }
138
139      isim(!$category){
140        $parent = Term::load($parentID);
141        isim($parent){
142          $category = Term::create([
143            'name' => $term,
144            'parent' => $parent,
145            'vid' => $vocabulary,
146          ]);
147        }else{
148          $category = Term::create([
149            'name' => $term,
150            'vid' => $vocabulary,
151          ]);
152        }
153        $category->save();
154      }
155      return $category;
156    }
157
158  }
```

C.2. createNewApi.php

```
<?php

namespace Drupal\rules\Plugin\RulesAction;


use Drupal\rules\Core\RulesActionBase;
use Drupal\rules\Plugin\RulesDataProcessor\ApiProcessor;

/**
 * Provides "Create new api" rules action.
 *
 * @RulesAction(
 *   id = "create_new_api",
 *   label = @Translation("Create new Api"),
 *   category = @Translation("API"),
 *   context_definitions = {
 *    "fid" = @ContextDefinition("integer",
 *       label = @Translation("File ID"),
 *       description = @Translation("The ID of the file with the
    info.")
 *    )
 *   }
 * )
 */
class CreateNewApi extends RulesActionBase {
  /**
   * Create a new api.
   *
   * @param integer $fid
   *   ID of the file
   */
  protected function doExecute($fid) {
    $file = \Drupal\file\Entity\File::load($fid);
    $api = ApiProcessor::newApi($file);
  }
}
```

C.3. DeleteUnusedTerm.php

```
<?php

namespace Drupal\rules\Plugin\RulesAction;

use Drupal\rules\Core\RulesActionBase;
use Drupal\Core\Plugin\ContainerFactoryPluginInterface;
use Drupal\Core\Entity\EntityTypeManagerInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

/**
 * Provides a 'Delete a term if it is unused' action.
 *
 * @RulesAction(
 *   id = "rules_delete_unused_term",
 *   label = @Translation("Delete a term if it is unused"),
 *   category = @Translation("Taxonomy"),
 *   context_definitions = {
 *     "tid" = @ContextDefinition("integer",
 *       label = @Translation("Identifier"),
 *       description = @Translation("The id of the term that
    should be deleted if it is unused.")
 *     ),
 *   }
 * )
 *
 */
class DeleteUnusedTerm extends RulesActionBase {
  /**
   * Comprueba si el archivo asociado al nodo queda desvinculado
      de cualquier otro nodo y si es asi
   * lo elimina.
   *
   * @param int $tid
   *   The term id.
   */
  protected function doExecute($tid) {
    $countnode = \Drupal::database()->query("SELECT COUNT(nid) AS
        count FROM {taxonomy_index} WHERE tid = :tid", array(':
        tid' => $tid))
                  ->fetchField();
    isim($countnode < 1){
        $term = \Drupal\taxonomy\Entity\Term::load($tid);
        $term->delete();
    }
  }
}
```

C.4. DeleteUnusedFile.php

```
<?php

namespace Drupal\rules\Plugin\RulesAction;

use Drupal\rules\Core\RulesActionBase;
use Drupal\Core\Plugin\ContainerFactoryPluginInterface;
use Drupal\Core\Entity\EntityTypeManagerInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

/**
 * Provides a 'Delete a file if it is unused' action.
 *
 * @RulesAction(
 *   id = "rules_delete_unused_file",
 *   label = @Translation("Delete a file if it is unused"),
 *   category = @Translation("File"),
 *   context_definitions = {
 *     "fid" = @ContextDefinition("integer",
 *       label = @Translation("Identifier"),
 *       description = @Translation("The id of the file that should be deleted if it is unused.")
 *     ),
 *   }
 * )
 *
 */
class DeleteUnusedFile extends RulesActionBase {
  /**
   * Comprueba si el archivo asociado al nodo queda desvinculado de cualquier otro nodo y si es asi
   * lo elimina.
   *
   * @param int $fid
   *   The file id.
   */
  protected function doExecute($fid) {
    //Get usage of a file
    $file = \Drupal\file\Entity\File::load($fid);
    if ($file) {
      $file_usage = \Drupal::service('file.usage')->listUsage($file);
      isim(count($file_usage) < 1){
        $file -> setTemporary();
        $file -> delete();
      }
    }
  }
}
```

C.5. UploadApi.php

```
<?php

namespace Drupal\rules\Plugin\RulesAction;

use Drupal\rules\Core\RulesActionBase;
use Drupal\rules\Plugin\RulesDataProcessor\ApiProcessor;

/**
 * Provides "Update api" rules action.
 *
 * @RulesAction(
 *   id = "update_api",
 *   label = @Translation("Update Api"),
 *   category = @Translation("API"),
 *   context_definitions = {
 *
 *    "nodeID" = @ContextDefinition("integer",
 *       label = @Translation("Node ID"),
 *       description = @Translation("The ID of the updated node
    .")
 *    ),
 *    "fileID" = @ContextDefinition("integer",
 *       label = @Translation("File ID"),
 *       description = @Translation("The ID of the old file in
    the node that whill be removed.")
 *    ),
 *   }
 * )
 */
class UploadApi extends RulesActionBase{
  /**
   * Upload api.
   *
   * @param integer $nodeID
   *   ID of the node
   * @param integer $fileID
   *   ID of the file
   */
  protected function doExecute($nodeID,$fileID) {
    $node = \Drupal\node\Entity\Node::load($nodeID);
    $api = ApiProcessor::editApi($nodeID);

    $file = \Drupal\file\Entity\File::load($fileID);
    isim($api){
      $file -> delete();
    }else{
      $node = \Drupal\node\Entity\Node::load($nodeID);
      $node -> field_yaml = $file;
    }
```

```
48      }
49  }
```