****

**Escuela de Ingenierías**

**Industrial, Informática y Aeroespacial**

Sistema híbrido de recomendación de películas empleando grafos de conocimiento

Sistemas de Información de Gestión y Business Intelligence

**GRADO EN INGENIERÍA INFORMÁTICA**

(Diciembre, 2020)

Autor: D. Álvaro González Jiménez

Tutor: D. Enrique López González

**ÍNDICE**

[1. Descripción del problema 3](#_Toc57583394)

[2. Herramientas utilizadas 6](#_Toc57583395)

[1. Base de datos: 6](#_Toc57583396)

[2. Interfaz de usuario (Frontend): 7](#_Toc57583397)

[3. Servidor (Backend): 7](#_Toc57583398)

[3. Base de datos utilizada 10](#_Toc57583399)

[1. Selección de los datos 10](#_Toc57583400)

[2. Conversión de los datos 11](#_Toc57583401)

[3. Importación de los datos 12](#_Toc57583402)

[4. Procesamiento de los datos 13](#_Toc57583403)

[5. Estructura de la base de datos 14](#_Toc57583404)

[4. Descripción de la aplicación 17](#_Toc57583405)

[5. Explicación del algoritmo 18](#_Toc57583406)

[a. Algoritmo basado en contenido 18](#_Toc57583407)

[b. Algoritmo de filtrado colaborativo 23](#_Toc57583408)

[c. Algoritmo híbrido 28](#_Toc57583409)

[6. Análisis de resultados 33](#_Toc57583410)

[7. Dafo 34](#_Toc57583411)

[8. Líneas de futuro 35](#_Toc57583412)

[9. Lecciones aprendidas 36](#_Toc57583413)

[10. Bibliografía 37](#_Toc57583414)

1. Descripción del problema

Tal vez si pensamos en un sistema de recomendación, el primero que se nos viene a la mente es un sistema de recomendación de películas. Todas y cada una de las plataformas de cine en streaming emplean un complejo sistema de recomendación encargado de analizar gigantescos volúmenes de datos y realizar sugerencias a sus usuarios en tiempo record. Aunque mucha gente lo desconozca, dichos sistemas no sólo tienen en cuenta los gustos del usuario para efectuar una recomendación, sino que analizan una gran cantidad de variables tales como el género, la edad, la raza, el dispositivo en el que sueles ver contenido, las horas del día en las que acostumbras a visualizar contenido o incluso el lugar de residencia con el fin de obtener recomendaciones lo más certeras posibles.

Estos sistemas, que se encuentran en constante desarrollo y evolución, son mucho más que simples recomendadores de películas. Si bien es posible que aunque utilices dichas plataformas con asiduidad no hayas reparado en ello, aunque estos sistemas recomienden la misma película o serie a dos personas diferentes, la portada mostrada no tiene por qué coincidir. Esto tiene una explicación muy sencilla, esos dos usuarios no son iguales, o al menos no lo son a ojos del sistema. Según la asunción que el sistema tenga del usuario, la portada que se le muestra difiere. Aunque desde luego este tipo de comportamientos no se pueden generalizar para toda la población, en la práctica ocurre que es infinitamente más probable que acabemos visualizando un contenido si la portada (que no deja de ser nuestra primera impresión sobre el contenido) es de nuestro agrado. Debido a esto, estas plataformas intentan que dicha primera impresión sea lo más fructífera posible, mostrando al usuario una portada que se asemeje al estilo de contenido que al usuario le guste visualizar, así como mostrando personajes que concuerden en raza y género con la asunción que el sistema tenga de esas dos características del usuario.

Tras esta breve descripción del estado del arte, supongo que lo más lógico es pensar ¿por qué iba a querer alguien desarrollar un sistema de recomendación de películas dado el extremo desarrollo y complejidad que caracteriza a los sistemas existentes en el mercado? Pues, aunque parezca increíble, tiene su respuesta. Vayamos por partes:

1. **Extensión del catálogo:** Como es evidente, los sistemas de recomendación de las grandes plataformas de streaming única y exclusivamente trabajan sobre el conjunto de películas que, en ese momento, están disponibles en la plataforma. Esto supone una gran diferencia con mi aplicación, que consta con una base de datos con información y valoraciones de algo más de 9.000 películas, mientras que por ejemplo Netflix apenas cuenta con unas 2.500, Prime Video con unas 4.400 y HBO con unas 900.
2. **Antigüedad de las películas:** En las plataformas de streaming, las películas que más abundan son aquellas más modernas, que por lo general tienden a gustarle mucho más al gran grueso de la población. Sin embargo, precisamente por la necesidad de tener en el catálogo películas que agraden a la mayoría, quedan en el olvido un sinfín de clásicos como *Con la muerte en los talones* de Alfred Hitchcock, auténticos clásicos del séptimo arte discriminados únicamente por su antigüedad. En mi sistema, esta discriminación no existe. Pueden encontrarse películas desde el año 1903 hasta la actualidad, recomendándose todas ellas por igual, todo en base a ciertos criterios y reglas matemáticas que serán perfectamente descritas en capítulos posteriores de este mismo documento.
3. **Ausencia de intereses externos:** Con el paso de los años, las plataformas de streaming han dejado de ser simples intermediarias entre productoras cinematográficas y usuarios, pasando a convertirse cada vez con más frecuencia en productoras del propio contenido que ofrecen. Aquí se plantea un dilema, ¿acaso alguien considera que los sistemas de recomendación no van a valorar especialmente bien las películas que sus propias compañías han producido? Esas películas no se pueden ver en una sala de cine, tampoco se pueden adquirir en un centro comercial. Que tengan éxito o no depende solamente de que una gran cantidad de usuarios las visualicen, y ahí, los sistemas de recomendación tienen la misma importancia que una buena campaña de marketing. Mi sistema, por el contrario, no se ve influenciado por ningún tipo de vicio ni interés externo. Es, si se quiere ver así, totalmente puro e imparcial.

Habiendo ya explicado en qué se diferencia de los sistemas de recomendación convencionales, queda por explicar exactamente cómo funciona el sistema, qué problemas resuelve y, en definitiva, cuál es su razón de ser.

En su función más elemental, es un sistema de recomendación híbrido capaz de recomendar películas en base a dos aspectos totalmente diferentes que son aunados finalmente para efectuar una recomendación final. Estos dos aspectos son los siguientes:

* **Algoritmo de recomendación basado en contenido:** En base a las características que definen a las películas valoradas por el usuario, se efectúan recomendaciones de películas similares a las valoradas.
* **Algoritmo de recomendación de filtrado colaborativo:** En base a las valoraciones asignadas a las películas, se recomiendan al usuario las películas mejor valoradas por otros usuarios, en este caso aquellos que más se parecen a él.

Además de como sistema de recomendación, la aplicación también puede funcionar como una agenda en la que apuntar aquellas películas que hemos visto y la puntuación que les hemos asignado, aunque claramente se trata de una función secundaria.

1. Herramientas utilizadas

En lo relativo a las tecnologías utilizadas, se pueden dividir en 3 grupos perfectamente diferenciados: base de datos, interfaz de usuario y servidor.

1. Base de datos: Entre las diversas tecnologías de bases de datos que pueden encontrarse en el mercado, hay dos que se llevan toda la atención: las bases de datos relacionales (MySQL, Oracle, MSSQL) y las bases de datos orientadas a documentos (MongoDB, CouchDB). Aunque sus usos son extremadamente variados (no en vano copan casi todo el mercado) para el problema que deseaba resolver no eran, ni en el mejor de los casos, lo más mínimamente viables.

El motivo es sencillo, no son capaces de computar cálculos sobre el más de millón de relaciones de las que dispone la base de datos del sistema en un periodo de tiempo aceptable. Por ello, me decidí a utilizar una base de datos orientada a grafos de conocimiento, Neo4j para ser más exactos. Su estructura y funcionamiento, en el que las relaciones son el aspecto más importante, las convierten en la base de datos más apropiada para resolver este tipo de problemas, así como otros problemas relacionados con detección de fraudes o incluso investigaciones genéticas.

Habiendo decidido el tipo de base de datos a utilizar, tan sólo quedaba una cosa, aprender a interactuar con ella. Neo4j implementa un lenguaje propio de consultas basado en SQL denominado Cypher, lo que hace que, si bien no sea necesario aprender la lógica que opera detrás de las consultas, sí que resulte extremadamente necesario aprender la sintaxis que emplea Cypher. Para ello, decidí completar el curso **Introduction to Neo4j 4.0** [1]**,** un curso de unas 16 horas de duración en el que pude aprender todas las funcionalidades de Cypher que posteriormente tuve que utilizar en el desarrollo del sistema, es decir, funcionamiento de una base de datos orientada a grafos de conocimiento, creación de nodos y relaciones, queries, indexación, importación de datos…

Sobre la base de datos hay muchos más aspectos que comentar, de modo que he decidido crear un capítulo que hable en exclusiva del origen del base de datos, procesamiento, importación, estructura, etc.

1. Interfaz de usuario (Frontend): A la hora de diseñar la interfaz de usuario, que generalmente se denomina frontend, decidí utilizar las tecnologías que ya conocía previamente. Así, al no tener que aprender desde 0 ninguna tecnología, puede realizar el desarrollo de una forma mucho más eficiente, obteniendo además un mejor resultado. Las tecnologías en cuestión son las siguientes:
2. **Javascript** [2]**:** Lenguaje de programación que aporta interactividad dinámica a la página web.
3. **Vue** [3]**:** Framework de Javascript que permite la construcción de interfaces de usuario y aplicaciones de una única página.
4. **Vuetify** [4]: Framework de Vue que permite la creación de interfaces de usuario basándose en componentes prediseñados.
5. **Vue router** [5]: Librería encargada de gestionar los movimientos entre vistas en una aplicación creada con Vue.
6. **Vuex** [6]: Librería encargada de gestionar los estados de una aplicación creada con Vue. Además, sirve como repositorio central de datos de la aplicación.
7. **Vuex persistedstate** [7]: Librería encargada de persistir la información de Vuex al refrescar una página.
8. **Axios** [8]: Cliente HTTP basado en promesas que permite efectuar peticiones API a un servidor.
9. Servidor (Backend): A la hora de diseñar el servidor de la aplicación, generalmente denominado Backend, se me presentaron dos alternativas entre las que tuve que decidir. Por un lado, tenía la opción de desarrollarlo en nodejs [9], un entorno en tiempo de ejecución basado en Javascript [2] que permite el desarrollo de servidores. Por otro lado, dado que es en el servidor donde se ejecutan los algoritmos de recomendación, estaba la opción de desarrollarlo en Flask [10], un framework de Python [11] que permite el desarrollo de servidores de forma sencilla.

Tras valorar pros y contras de ambas implementaciones me decidí por la segunda opción, especialmente porque Python iba a ser muchísimo más eficiente que Javascript a la hora de realizar operaciones de análisis y procesamiento de datos.

Como no había desarrollado jamás una aplicación con Flask, me vi en la necesidad de aprender desde 0 su funcionamiento, el cual aprendí mediante dos vías diferentes:

1. **Visualización de un vídeo en Youtube** [12]: En este vídeo se explica a la perfección el funcionamiento del framework. Peticiones, respuestas, gestión de rutas, argumentos, validación de entradas… Tan solo visualizando los 50 primeros minutos de este vídeo ya pude ponerme sin ningún problema a desarrollar el backend del proyecto.
2. **Observación del siguiente proyecto en Github** [13]: En este repositorio, dentro de la carpeta flask-api puede encontrarse un ejemplo de desarrollo de una aplicación Flask con acceso a base de datos neo4j que me sirvió de ejemplo para observar cómo se hacen las consultas a la base de datos desde Python.

Tras aprender el funcionamiento del framework, proceso que me llevó aproximadamente un día, pude empezar a desarrollar el backend del proyecto sin ningún tipo de problema. Las tecnologías que empleé, además de por supuesto Python y Flask, fueron las siguientes:

1. **Pandas** [14]**:** Módulo de Python que sirve para la manipulación y análisis de datos.
2. **Numpy** [15]**:** Módulo de Python que da soporte a la creación de vectores y matrices, así como a la realización de una gran cantidad de operaciones matemáticas complejas.
3. **Neo4j** [16]**:** Módulo de Python que permite la comunicación con bases de datos Neo4j.
4. **Sklearn** [17]**:** Módulo de Python que aporta una gran cantidad de funcionalidades relacionadas con Aprendizaje automático.
5. **Scipy** [18]**:** Módulo de Python que aporta una gran cantidad de herramientas y algoritmos matemáticos.

1. Base de datos utilizada

A la hora de desarrollar un buen sistema de recomendación, resulta estrictamente necesario que la base de datos sobre la que se trabaje sea lo más completa posible. Por muy robusto y efectivo que pueda ser el sistema, si la calidad de los datos no es la adecuada, las recomendaciones en la práctica no serán casi nunca certeras. Debido a esto, he considerado oportuno crear un capítulo dedicado única y exclusivamente a la propia base de datos en la que se traten diversos temas como la selección, procesamiento y carga de los datos, así como de la estructura de la propia base.

1. Selección de los datos

Para seleccionar un conjunto de datos sobre el que trabajar, decidí buscar en la conocida comunidad de científicos de datos Kaggle [19]. Allí, pueden encontrarse una gran cantidad de dataset, cursos y competiciones de temáticas más que variadas. Entre los dataset de películas que encontré, debo destacar los siguientes:

1. **TMDB 5000 Movie Dataset** [20]: Aunque se trata de un dataset bastante completo con numeras películas y valoraciones, faltaban algunos campos en la base de datos que, en mi opinión, eran muy importantes, como por ejemplo el reparto de la película.
2. **The movies dataset** [21]**:** Si bien se trata de un dataset extremadamente completo que contiene una gran cantidad de información sobre cada película, valoraciones, etc. A la hora de explorar el conjunto de datos en profundidad me di cuenta de que todo estaba almacenado en formato json dentro de un archivo csv. En base a esto, y aunque el dataset en principio tenía bastante buena pinta, decidí seguir buscando.
3. **Movielens** [22]**:** El dataset de películas por excelencia sobre el que desarrollar un sistema de recomendación, que incluye millones de valoraciones de usuarios a decenas de miles de películas. El único problema que encontré al dataset es que no almacena información sobre el casting de las películas, lo que hace que, a la hora de mostrar información en mi aplicación sobre las películas, no pueda ser esta muy completa.
4. **Extensión de Movielens** [23]**:** Cuando ya había decidido utilizar Movielens, encontré por casualidad este dataset, una ampliación de Movielens realizada por algunos investigadores de la Universidad Autónoma de Madrid. En este dataset, además de toda la información existente en el dataset de Movielens, podemos encontrar otra información adicional como el país de origen de la película, el reparto, director, etc. Por si esto fuera poco, todo el dataset ha pasado previamente un proceso de limpieza, de modo que no hay que realizar un proceso de limpieza de datos muy inferior a los demás.

Tras analizar las ventajas e inconvenientes de seleccionar cada uno de los diversos dataset, decidí utilizar el último de todos (la extensión de Movielens) dado que contiene una gran cantidad de películas y valoraciones y, además, no es necesario realizar un proceso de limpieza demasiado exhaustivo.

1. Conversión de los datos

Para importar datos a una base de datos Neo4j desde un archivo, es necesario que éste se encuentre en formato CSV. En mi caso, el conjunto de datos elegido se encontraba en varios archivos en formato DAT, en el que las diferentes columnas se encuentran separadas por tabulaciones en lugar de por comas. Debido a esto, me vi en la obligación de desarrollar un pequeño script en Python que se encargara de transformar los archivos al formato deseado.

Dentro de este pequeño script, además de cambiar las tabulaciones por comas, también se realizan otro tipo de transformaciones como, por ejemplo, reemplazar todos los caracteres especiales por sus equivalentes en el lenguaje inglés, eliminando por tanto las tildes a las vocales, sustituyendo la letra ñ por la n, etc.

Aunque no lo había comentado hasta el momento, todos los archivos que forman la base de datos, un total de 6, deben ser transformados mediante el script en Python procesar\_csv.py que puede encontrarse en el repositorio de Github en la carpeta Scripts.

1. Importación de los datos

Con el fin de importar todos los datos en la base de datos, y aunque es posible hacerlo en un único script, decidí crear un script por cada archivo, de modo que fuera más sencillo tanto depurar errores y, además, el tiempo de carga fuera menor.

Aunque la estructura de dichos scripts difiere en base a cuál sea el archivo que se quiere cargar, aquí puede encontrarse un ejemplo de la carga de todas las películas que contiene la base de datos:

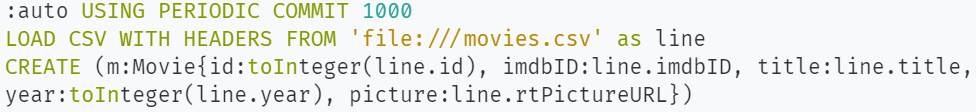


Figura 3.1.- Importación datos películas

Al efectuar la carga, decidí importar el contenido de los archivos en su totalidad, de modo que la carga de las valoraciones puede llegar a demorarse unas cuantas horas. En el caso de querer realizar esta carga en un periodo de tiempo mucho menor, bastaría con añadir una limitación en el script de carga, indicando que solo se quieren cargar las x primeras líneas.

Los scripts de carga pueden encontrarse dentro del repositorio de Github en la carpeta Scripts. Al ejecutarlos, es importante incluirlos dentro de la carpeta Import en Neo4j, de no ser así, el gestor no sería capaz de encontrarlos y, por tanto, no se podría efectuar la carga de manera satisfactoria.

Muy relacionados con la carga de los datos está la indexación. En vistas a que el sistema de búsqueda implementado en la interfaz de usuario resulte más efectivo, siempre es conveniente añadir una serie de índices que, bien implementados, son capaces de reducir los tiempos de carga de una manera increíble.

En mi caso, el índice principal y más importante es el siguiente, que permite indexar las películas en base a su título:



Figura 3.2.- Indexación películas en base a título

Al igual que todos los scripts, los archivos encargados de la creación de índices pueden encontrarse dentro del repositorio Github en la carpeta Scripts.

1. Procesamiento de los datos

Habiendo importado todos los datos, llega el momento de efectuar una serie de transformaciones en los datos que nos ayuden a eliminar duplicados y a facilitar el trabajo a los algoritmos de recomendación que se ejecutarán en el backend.

Dentro de este procesamiento, destacan 3 tipos totalmente diferenciados:

* 1. **Eliminación nodos repetidos**: En vistas a una mayor robustez del sistema de recomendación, resulta necesario eliminar aquellos nodos que se encuentran repetidos en la base de datos. Aunque el dataset se encontraba ya bastante limpio en un principio, sí que se podían encontrar algunas inconsistencias que debían ser solventadas.

Entre los directores, el id asignado para su identificación se encontraba en algunas ocasiones repetido, de modo que existían dos o más nodos para un mismo director. Esto también ocurría con algunos actores y películas, que también tenían el mismo identificador o la misma portada en el caso de las películas.

Para eliminar estos nodos repetidos, decidí implementar un sencillo Script en Cypher. El siguiente ejemplo muestra como eliminar todas las películas con la misma portada a excepción de 1. Sin embargo, el script es perfectamente extrapolable a cualquier otro tipo de nodo

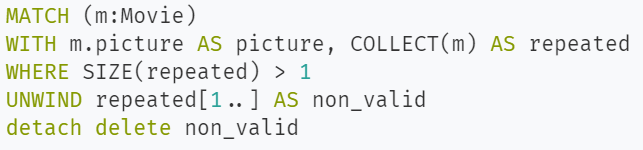


Figura 3.3.- Eliminar películas repetidas

* 1. **Eliminación nodos incompletos**: Al igual que con los nodos repetidos, los nodos incompletos también podían suponer un problema al sistema. Debido a esto, decidí eliminar todos aquellos nodos que carecieran de alguna propiedad, ya sirviera esta de identificación o fuera secundaria.
  2. **Procesamiento de texto**: Aunque en un principio no había reparado en ello, al ir desarrollando el algoritmo de recomendación basado en contenido reparé en que, por la forma de separar tokens en Python, no podían existir guiones ni espacios en los nombres de las características que fueran a ser utilizados como criterios de recomendación. Debido a esto, decidí desarrollar un script en Cypher que lo solucionara. El siguiente ejemplo muestra como sustituir los guiones y espacios contenidos en el texto por barras bajas:

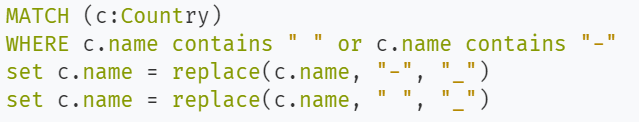


Figura 4.- Sustitución espacios y guiones por barra baja

1. Estructura de la base de datos

Para terminar el capítulo, considero oportuno explicar de forma detallada cuál es la estructura de la base de datos, es decir, por qué tipo de nodos y relaciones está formada.

Al haberse realizado una carga de la totalidad de los datos, el tamaño de la base es considerable, unos 110.000 nodos y 1.000.000 de relaciones. Dado que la gran mayoría de relaciones son valoraciones de los usuarios a las películas, y como ya comenté anteriormente, en el caso de querer que los tiempos de respuesta de la base de datos sean menores y el tiempo de importación de los datos sea también menor, tan sólo es necesario establecer una limitación a la hora de cargar las valoraciones en las que se indique el número de éstas que se quieran cargar.

En total, la base de datos está formada por 6 nodos diferentes y 5 relaciones entre ellos, siendo el nodo central de la base el que contiene el título de las películas. El esquema de visualización de la base de datos sería el siguiente:

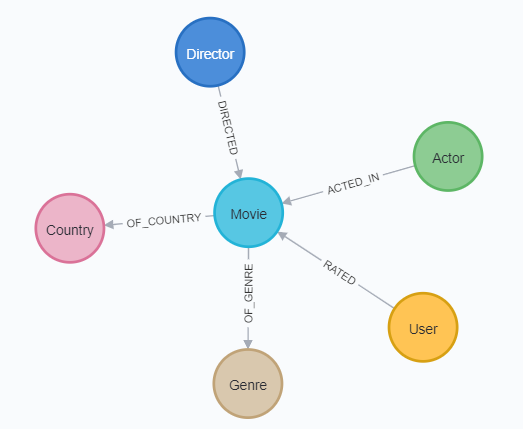


Figura 5.- Esquema visualización BBDD

A su vez, se podría definir cada uno de los nodos de la siguiente manera:

* **Movie**: Nodo que contiene la siguiente información sobre una película: identificador único, identificador de la película en imdb, título, año, link a la fotografía de la portada. En la base hay un total de 9.060 películas diferentes.
* **Genre**: Género al que puede pertenecer una película. En la base se pueden encontrar un total de 20 géneros diferentes.
* **Country**: País de origen de una película. En la base hay un total de 71 países diferentes.
* **Actor**: Nodo que contiene la información de cada actor en la base, identificador único y nombre del actor. Existen un total de 95320 actores diferentes.
* **Director:** Nodo que contiene la información de cada director en la base, identificador único y nombre del director. Existen un total de 4054.
* **User:** Nodo que contiene la información de cada usuario. Para los usuarios generados a partir del csv, la única información disponible será el identificador. Sin embargo, para los usuarios generados a partir de la aplicación web mediante el sistema de registro, también se encontrará almacenado su nombre, username y su contraseña debidamente cifrada.

Una vez descritos de forma correcta los distintos nodos existentes en la base de datos, creo necesario definir de igual modo las relaciones existentes entre ellos.

* **OF\_GENRE**: Relación entre Movie y Genre. Cada película puede pertenecer a un género o a varios. Existen un total de 18.366 relaciones de este tipo.
* **OF\_COUNTRY**: Relación entre Movie y Country. Cada película solo puede estar relacionada con un único país de origen. Existen un total de 9.060 relaciones de este tipo.
* **ACTED\_IN**: Relación entre Movie y Actor. Cada actor puede actuar en tantas películas como se desee. Existen un total de 207.985 relaciones de este tipo.
* **DIRECTED**: Relación entre Director y Movie. Cada director puede dirigir tantas películas como sea necesario y, además, cada película puede estar dirigida por más de un director. Existen un total de 9.060 relaciones de este tipo.
* **RATED:** Relación entre User y Movie. Cada User puede valorar tantas películas como quiera y cada película puede estar valorada por más de un User. La relación tiene un atributo que contiene la valoración que el usuario ha asignado a dicha película, un número entre 0 y 5. Existen un total de 758.836 relaciones de este tipo.

4. Descripción de la aplicación

La aplicación es una aplicación web que, por tanto, opera bajo el paradigma cliente-servidor. Este tipo de aplicaciones están formadas por 3 partes esenciales, base de datos, interfaz de usuario y servidor.

1. Explicación del algoritmo

El algoritmo, como ya comenté al principio de esta memoria, es un algoritmo de recomendación híbrido compuesto de 2 algoritmos de diferentes, uno basado en contenido y otro de filtrado colaborativo.

Para mejor comprensión del algoritmo, que en la práctica no es un algoritmo sino tres, considero que es adecuado describir cada uno de esos tres algoritmos por separado: el basado en contenido, el de filtrado colaborativo y la suma de ambos.

Para la descripción de estos algoritmos, en lugar de basarme en el código de la aplicación, he considerado oportuno realizar una explicación teórica y, posteriormente, un ejemplo práctico muy sencillo que clarifique el algoritmo. De explicarlo sobre el código, sería más complejo de entender debido a que sería necesario tener cierto conocimiento sobre algunos módulos específicos de Python.

Cabe destacar que para el diseño e implementación de estos algoritmos me basé en 2 libros de texto:

* **Graph Powered Machine Learning** [24]: Libro en el que se habla de todo tipo de algoritmos de aprendizaje automático aplicados a bases de datos orientadas a grafos de conocimiento.
* **Recommender Systems. The textbook** [25]**:** La biblia de los sistemas de recomendación. Libro en el que se explica de forma detallada el funcionamiento de los diversos sistemas de recomendación existentes, profundizando en la lógica matemática sobre la que operan, planteando diversos escenarios, etc.

1. Algoritmo basado en contenido

Este tipo de algoritmos suelen dividirse por norma general en tres partes perfectamente diferenciadas, de modo que yo, para explicar el diseño, voy a dividir también la explicación en 3 partes.

* **Preprocesamiento y extracción de características**.

En este punto, se extraen las características de las diversas películas presentes en la base de datos. Tras un proceso de análisis exhaustivo del problema, decidí que las características de la película que se utilizarán serían género, director y país de origen.

Para cada película se generará un vector que tendrá tantas columnas como características diferentes existan en la base de datos. Es decir, si existen 10 géneros, 5 directores, y 3 países el tamaño del vector será de 10 + 5 + 3 = 18 características. Además, aquellas películas que hayan sido valoradas por el usuario tendrán una columna más en la que se refleje la puntuación asignada a dicha película.

En el vector, aquellas características que no estén presentes en la película estarán representadas por un número 0. A su vez, para las características que sí estén presente se pueden dar dos supuestos diferentes:

* + Si la característica es un género o un país, se encontrará un 1 en la columna correspondiente.
  + Si la característica es un director, se encontrará un 2 en la columna correspondiente.

Esto se debe a que, por norma general, el director de una película influye mucho más en la atracción que dicha película genera en el espectador que los géneros o el país de origen de la misma.

Este proceso de extracción de características se realizará para todas las películas existentes en la base de datos, sin importar si han sido valoradas por el usuario o no.

* **Creación de un perfil de usuario basado en contenido**

Una vez extraídas todas las características de cada película, es necesario modelar un perfil que represente los gustos del usuario. Para ello, tan solo se tendrán en cuenta aquellas películas que han sido valoradas por el usuario, apartando el resto de películas por el momento.

Para modelar el perfil, es necesario multiplicar cada vector de características por la valoración que el usuario ha realizado para cada película de modo que, si por ejemplo el usuario ha valorado la película con un 4, todas las características que no están presentes en la película continuarán siendo 0 mientras que las que sí están presentes pasarán a ser 4. Los directores, serán multiplicados además por 2, ya que como he comentado anteriormente tendrán el doble de peso en la recomendación que el resto de características.

A continuación, se debe calcular la media de cada una de las características, sin tener en cuenta los valores nulos (0). Para aquellas características que aparezcan en un porcentaje inferior al 30% de las películas valoradas, y en vistas a que los resultados del algoritmo no se vean alterados por características que aparecen en un número reducido de ocasiones, se les aplica una reducción multiplicándolas por el doble de su frecuencia. De este modo, no ocurre que una característica excepcionalmente valorada en una única ocasión altere los resultados del sistema, pues no se asume una particularidad como si de una generalidad se tratase.

Por último, es necesario aplicar una normalización al vector, de modo que los géneros y los países pasen a estar del rango [0,5] al rango [0,1] y los directores pasen a estar del rango [0,10] al rango [0,2]. Este paso es realmente necesario para calcular la similitud coseno entre el perfil y el resto de películas de la base de datos, pues las características de estas películas están definidas en esos rangos.

* **Filtrado y recomendación**

Para calcular las predicciones de puntuaciones para cada una de las películas de la base de datos no valoradas, emplearé la función de similitud coseno, expresada por la siguiente fórmula:



Figura 5.1.- Fórmula similitud coseno

Al calcular la similitud coseno entre el perfil generado y todas las películas no valoradas por el usuario obtendremos, para cada película, un valor entre 0 y 1. Cuanto mayor sea esta similitud, mayor será también la probabilidad de que esta película le guste el usuario.

Una vez definidas las tres partes que forman el algoritmo y cómo se va a implementar cada una de ellas, es el momento de poner un ejemplo práctico que ayude a la comprensión del funcionamiento del mismo. Para ello, se partirá de un ejemplo sencillo con 4 películas, 2 valoradas por el usuario y 2 sin valorar.

* **Preprocesamiento y extracción de características**.

Se partirá de un dataset compuesto por las siguientes 4 películas



Figura 5.2.- Conjunto películas

El primer paso consiste en obtener todas las características diferentes presentes en el dataset. Dichas características son:



Figura 5.3.- Características totales

A continuación, generamos para cada película un vector de características. En los géneros y países presente en la película se pondrá un 1, mientras que en el director se pondrá un 2. En las características que no estén presentes, se pondrá un 0. En las películas valoradas, además, se añadirá una columna en la que se indicará la valoración efectuada.

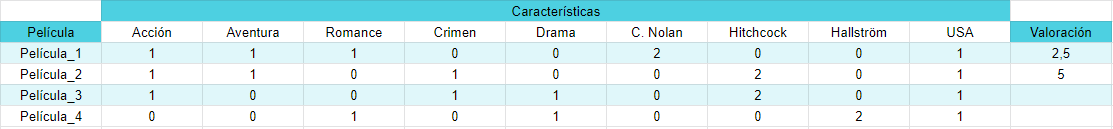


Figura 5.4.- Vectores de características generados

* **Creación de un perfil de usuario basado en contenido**

En este paso tan solo son necesarias las dos primeras filas de la matriz, en las que se encuentran las dos películas valoradas por el usuario.

Para comenzar, se multiplica cada uno de los valores del vector por la valoración asignada, obteniendo el siguiente resultado:

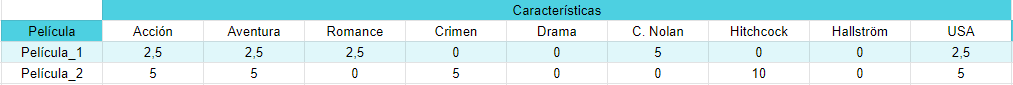


Figura 5.5.- Vector características tras multiplicación

A continuación, se calcula la media aritmética de las características, sin tener en cuenta los 0. El perfil resultante obtenido es el siguiente:

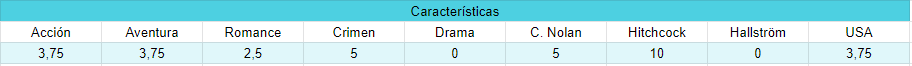


Figura 5.6.- Perfil generado sin normalizar

Por último, es necesario normalizar las características para que se encuentren en el mismo rango que los vectores de características definidos en el paso 1, pudiendo así calcular la similitud coseno entre ellos. Para ello, tan solo hay que dividir cada valor entre 5.

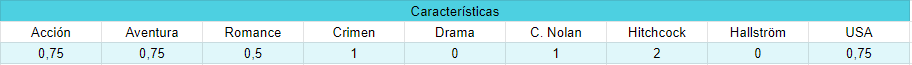


Figura 5.7.- Perfil normalizado

* **Filtrado y recomendación**

En este paso, partimos de los siguientes datos:

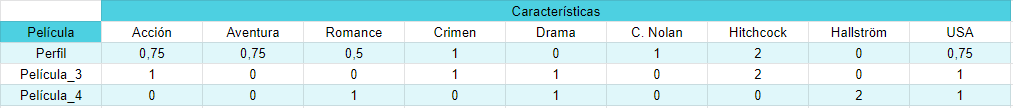


Figura 5.8.- Películas a comparar

En base a estos datos, calculamos la similitud coseno entre el Perfil y las dos películas que no han sido valoradas por el usuario, utilizando la siguiente fórmula:



Figura 5.9.- Fórmula similitud coseno

Para la película 3, el cálculo de la similitud coseno sería el siguiente:

A su vez, el cálculo de la similitud coseno para la película 4 sería el siguiente:

Como se puede comprobar, en base a las similitudes coseno calculadas, es altamente probable que la película 3 resulte del agrado del usuario, mientras que es bastante poco probable que lo resulte la película 4. En base a esta puntuación calculada, será la película 3 la que el sistema recomiende al usuario.

1. Algoritmo de filtrado colaborativo

Al igual que los algoritmos de recomendación basados en contenido, los algoritmos de recomendación de filtrado colaborativo también constan de 3 partes perfectamente diferenciadas.

* **Construcción matriz de valoraciones**

Para construir la matriz, lo primero que hay que hacer es solicitar a la base que nos devuelva la información relativa a las valoraciones efectuadas por los usuarios. Como el tamaño de la base de datos es gigantesco, he decidido tras efectuar un análisis de la situación solicitar tan solo la información relativa a los 100 usuarios que comparten más valoraciones con el usuario al que se quiere recomendar. De esta manera, se reduce de forma significativa la carga computacional del sistema.

A continuación, tras haber obtenido la información necesaria de la base de datos, ya se puede construir la matriz de valoraciones, que tendrá tantas filas como usuarios vayan a ser comparados y tantas columnas como películas diferentes hayan valorado entre todos los usuarios. Por tanto, si quisiéramos comparar 3 usuarios que hubieran valorado entre todos 15 películas, la dimensión de la matriz de valoraciones resultante sería de 3x15.

Los valores colocados en cada posición de la matriz serán las valoraciones que cada usuario haya asignado a cada película y, en caso de no haber valorado una película, un 0. De este modo, en la posición i,j de la matriz podrá encontrarse la valoración que el usuario i asignó a la película j y, en caso de no existir una valoración para la película, un 0.

Como es lógico, la matriz de valoraciones será por norma general una matriz dispersa, pues lo más común es que un usuario haya valorado solo unas pocas películas de entre todas las existentes en la base de datos.

* **Cálculo similitud entre usuarios**

Tras haber construido la matriz de valoraciones, es necesario calcular la similitud entre el usuario para el que se quiere efectuar una recomendación y el resto de usuarios.

Aunque dicha similitud se puede calcular mediante diversas técnicas, considero que la técnica que mejor puede solventar la necesidad que se me presenta es Pearson correlation. El motivo es sencillo, es una técnica que tiene en cuenta la media de las valoraciones al realizar la comparación, algo que es especialmente importante en este caso dado que podríamos encontrarnos usuarios que valoran todas sus películas en un rango de [3.5,5] y otros que usuarios que valoran todas sus películas con puntuaciones más bajas.

Aunque existen diversas técnicas para calcular la correlación de Pearson, considero que en este caso en particular resulta mucho más interesante calcular la similitud únicamente entre las películas que ambos usuarios han valorado, pues de esta manera es posible medir con mucha más precisión cómo de similares son ambos perfiles. La fórmula para calcular dicha similitud es la siguiente:



Figura 5.10.- Cálculo Pearson Correlation

* **Predicción valoraciones**

Tras haber calculado la similitud entre todos los usuarios y el usuario al que queremos recomendar, llega el momento de predecir la puntuación que dicho usuario podría asignar a todas las películas valoradas por los usuarios más parecidos a él.

Con el fin de reducir de forma significativa el coste computacional de las operaciones matemáticas que es necesario realizar, he tomado la decisión de tener en cuenta única y exclusivamente los 10 usuarios más parecidos.

Para calcular esa predicción de las valoraciones, he decidido utilizar la siguiente fórmula, que tiene en cuenta la valoración media del usuario y del resto de usuarios a la hora de realizar la predicción.



Figura 5.11.- Función de predicción

Aunque la fórmula no lo refleje, las películas que hayan sido valoradas por un porcentaje reducido de los usuarios más parecidos no serán tenidas en cuentas. He decidido anteponer la robustez del algoritmo con respecto a las particularidades.

Una vez definidas las 3 partes que forman el algoritmo y su implementación, llega el momento de poner un ejemplo práctico que facilite la comprensión del mismo. Para simplificarlo lo máximo posible, tan solo se tendrán en cuenta 3 usuarios que hayan valorado 5 películas entre todos y que, además, tengan 3 valoraciones en común.

* **Construcción matriz de valoraciones**

Supongamos que el usuario al que queremos efectuar una recomendación es el usuario 1, obtenemos la información de los usuarios con más valoraciones en común. Acto seguido, construimos la siguiente matriz de valoraciones, colocando un 0 en las columnas de las películas no valoradas por el usuario.

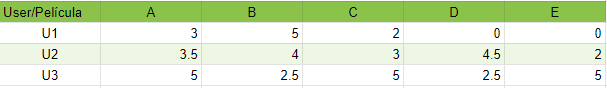


Figura 5.12.- Matriz valoraciones generada

* **Cálculo similitud entre usuarios**

Para calcular la similitud entre el usuario 1 y el resto de usuarios que se pueden encontrar en la matriz de valoraciones se utiliza la siguiente implementación de Pearson Correlation.



Figura 5.13.- Cálculo Pearson Correlation

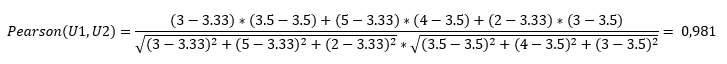
Un ejemplo del cálculo de la similitud entre el usuario 1 y el usuario 2 es el siguiente.  


Figura 5.14.- Calculo similitud U1 y U2

Tras realizar el cálculo de las dos correlaciones de Pearson, la tabla resultante es la siguiente.

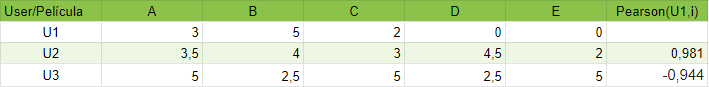


Figura 5.15.- Pearson Correlation calculada

Cuanto mayor sea el valor de Pearson, mayor es la similitud entre los usuarios. Como ya podíamos observar a simple vista, el usuario 2 es muy parecido al usuario 1, mientras que el usuario 3 es muy diferente.

* **Predicción valoraciones**

Ahora que ya hemos calculado la similitud de Pearson para cada usuario, llega el momento de realizar la predicción de la puntuación de las películas no valoradas.

Antes de realizar la predicción, es necesario mencionar que solo se tendrán en cuenta las valoraciones de los usuarios más parecidos al usuario. Como este ejemplo es muy sencillo, tan sólo tendré en cuenta las valoraciones del usuario 2. Sin embargo, en la aplicación existen un número mucho mayor de usuarios, lo que hace que, a la hora de efectuar predicciones, solo haya que tener en cuenta las valoraciones de aquellos 10 usuarios más parecidos al que se desea recomendar.

Como recordatorio, la fórmula que se utiliza para realizar la predicción es la siguiente:



Figura 5.16.- Función de predicción

Un ejemplo de aplicación de la fórmula a la predicción de la puntuación para la película D es la siguiente:

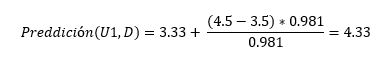


Figura 5.17.- Predicción puntuación D

Por otra parte, la aplicación de la fórmula para la predicción de la puntuación para la película E es:

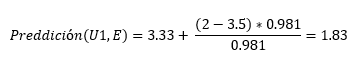


Figura 5.18.- Predicción puntuación E

Como podemos observar, en base a los resultados obtenidos, es altamente probable que la película D sea del agrado del usuario 1. En lo relativo a la película E, las probabilidades se reducen muchísimo.

En base a esto, el sistema recomendaría al usuario 1 la película D, que es la que ha obtenido una predicción de valoración mayor.

Por último, cabe destacar que existe la posibilidad de que alguna de las valoraciones que se realicen se encuentren en un rango diferente al [0,5], rango en el que se encuentran todas las valoraciones existentes en la base de datos. Si eso pasara, su valor se cambiaría a 0 o a 5 dependiendo de si han excedido la cota superior o inferior.

Como último paso ya del algoritmo, en vistas a una mejor integración con el algoritmo híbrido, se dividen las predicciones realizadas para cada película entre 5, logrando así que dicha predicción se encuentre en el rango [0,1], mismo rango que las predicciones efectuadas por el algoritmo basado en contenido.

1. Algoritmo híbrido

Aunque existen diversos algoritmos de recomendación híbridos, cada uno con sus pros y sus contras, para un sistema de recomendación de películas el que mejor funciona con diferencia es el algoritmo de recomendación híbrido paralelo ponderado, que tiene el siguiente esquema:

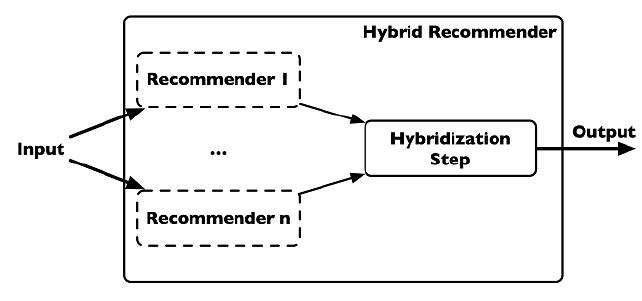


Figura 5.19.- Esquema algoritmo paralelo ponderado

De forma general, y al igual que los 2 algoritmos anteriores, este algoritmo también puede dividirse en 3 etapas perfectamente diferenciadas.

* **Unión de las salidas de ambos algoritmos**

Como la salida de ambos algoritmos es una lista que contiene los ids de las películas y la valoración estimada, es necesario convertir este par de listas en una matriz en la que se encuentre, para cada película, la puntuación estimada por cada uno de los algoritmos de recomendación.

En caso de que alguna película no esté presente en uno de los dos dataframes que forman la salida de los algoritmos, se pondrá un 0 en su lugar.

Con esta técnica conseguiremos una matriz que tendrá tantas filas como películas diferentes existan entre ambas salidas y dos columnas, pudiéndose encontrar en la primera la puntuación estimada por el algoritmo basado en contenido y en la segunda la puntuación estimada por el algoritmo de filtrado colaborativo.

* **Cálculo de la suma ponderada de puntuaciones**

Habiendo fusionado ya las salidas de ambos algoritmos, llega el momento de calcular la puntuación final de cada película, lo cual se realiza con la siguiente fórmula:

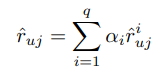


Figura 5.10.- Cálculo hibridación

A la hora de establecer las ponderaciones para cada uno de los algoritmos de recomendación, creo que lo más justo dada la más que probada efectividad de ambos es darle algo más de peso al algoritmo de CF con respecto al algoritmo CB. Por ello, he decidido establecer αCB como 0,4 y αCF como 0,6.

* **Ordenación en base a la puntuación estimada**

Tras haber calculado la puntuación para cada una de las películas, aunque pueda parecer algo tribial, es necesario ordenar las películas de modo que a la hora de devolver las n primeras, se devuelvan aquellas películas con una mayor puntuación.

Tras haber definido las 3 partes que forman el algoritmo, llega el momento de poner un ejemplo concreto en el que se facilite la comprensión del mismo. Para que el ejemplo sea lo más sencillo posible, se partirá de dos salidas de 5 y 3 películas cada una.

* **Unión de las salidas de ambos algoritmos**

Partimos de la siguiente salida de los dos algoritmos:

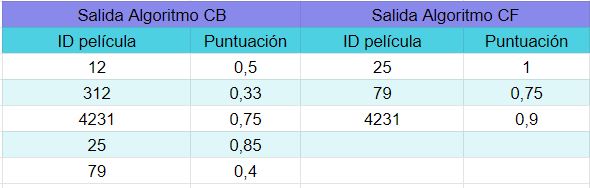


Figura 5.21.- Salida algoritmos

Como se puede apreciar, algunas de las películas existentes en la salida del algoritmo CB no están presentes en las salidas del algoritmo CF, lo que hace que a la hora de fusionar ambas salidas sea necesario poner un 0 en aquellas puntuaciones inexistentes. El dataframe resultante de la unión de ambas salidas es el siguiente:



Figura 5.22.- Matriz resultante

* **Cálculo de la suma ponderada de puntuaciones**

Tras haber generado la matriz resultante de la unión de ambas salidas, llega el momento de calcular la suma ponderada de ambas puntuaciones lo cual, como recordatorio, se calcula de la siguiente manera:

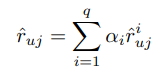


Figura 5.23.- Cálculo hibridación

Los cálculos que se efectuarían serían, por tanto, los siguientes:

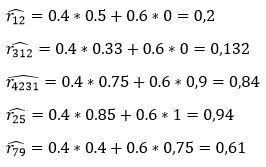


Figura 5.24.- Cálculos efectuados hibridación

Siendo la matriz resultante la siguiente:

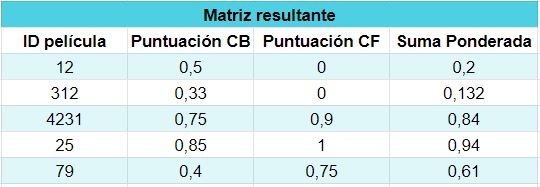


Figura 5.25.- Matriz ponderada resultante

* **Ordenación en base a la puntuación estimada**

Habiendo calculado ya la suma ponderada, solo nos queda ordenar la matriz en base al valor de dicha columna en orden descendente, de modo que las películas con mayor suma ponderada sean las que se encuentren en las primeras filas de la matriz. En este ejemplo concreto, la matriz final es la siguiente:

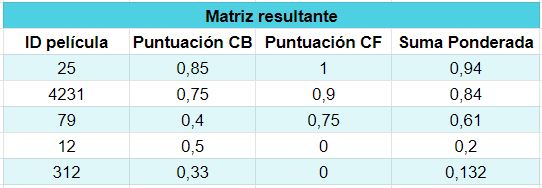


Figura 5.26.- Matriz ordenada

De esta matriz se seleccionarían las n primeras películas las cuales, tras consultar a la base la información necesaria que se debe mostrar en la aplicación, serían devueltas al frontend.

Con esto doy por terminada la explicación de los tres algoritmos de recomendación. Como ya comenté en el apartado anterior, el usuario puede seleccionar qué algoritmo de recomendación quiere que se utilice. Sin embargo, como se demostrará en el siguiente capítulo, el algoritmo que presenta una mayor robustez y fiabilidad es el híbrido, con bastante diferencia con respecto a los demás.

1. Análisis de resultados
2. Análisis crítico

Como en casi cualquier proyecto realizado, es prácticamente obligatorio realizar un análisis crítico exhaustivo que permita observar de manera global cuál es la situación del proyecto. De este modo, las estrategias de marketing que se planteen tendrán una efectividad mucho mayor.

Aunque existen diversas técnicas que permiten realizar un análisis crítico para una determinada empresa o proyecto, uno de los más simples y seguramente el más visual de todos es el análisis DAFO.

Dicho análisis, cuyas siglas corresponden a las 4 áreas diferentes a analizar (Debilidades, Amenazas, Fortalezas y Oportunidades), es una herramienta de estudio que permite estudiar la situación de una determinada empresa o proyecto analizando sus características internas (Debilidades y Fortalezas) y sus características externas (Amenazas y Oportunidades) y presentándolas en forma de matriz cuadrada.

En el caso concreto de este proyecto, la matriz DAFO obtenida es la siguiente:

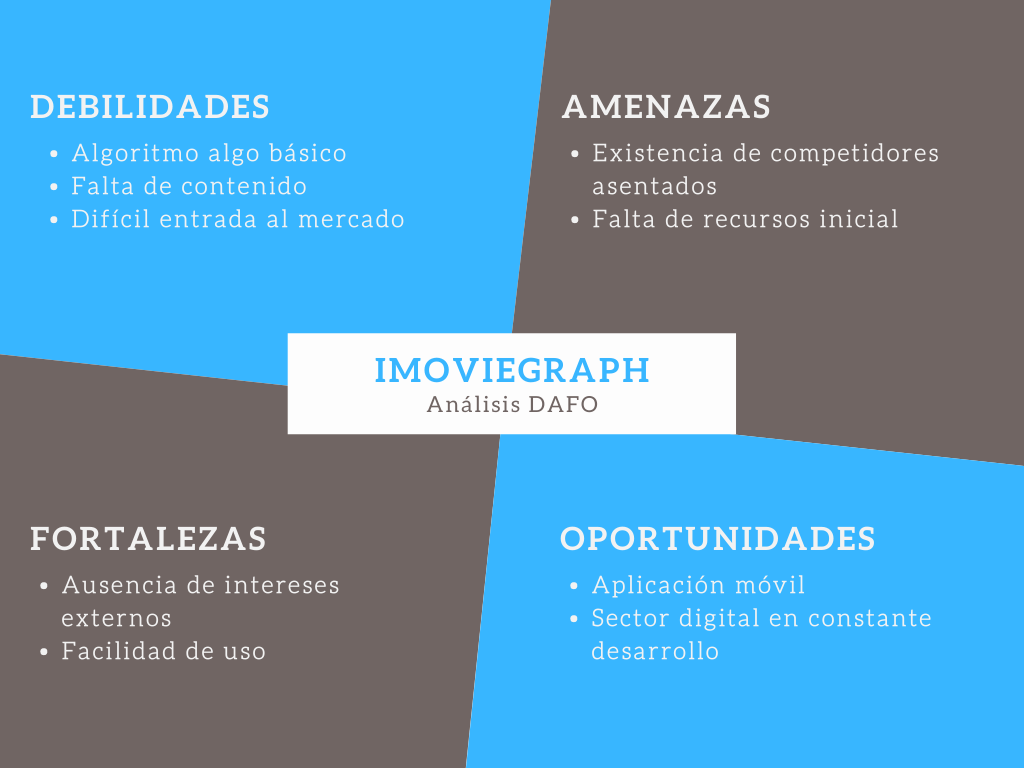


Figura 7.1.- Análisis DAFO

Aunque en la propia matriz se pueden comprender de forma rápida los puntos reseñables de las 4 áreas, considero apropiado explicar cada uno de ellos de forma un poco más extendida, aclarando aquellos conceptos que difícilmente pueden ser desarrollados en una matriz.

* 1. Debilidades
  + **Algoritmo algo básico**: Aunque ya se haya comprobado que el algoritmo tiene un comportamiento bastante robusto, hay una gran cantidad de aspectos que podrían añadirse y que mejorarían enormemente su rendimiento, tales como: Incorporación de algoritmo basado en contexto, incorporación de características como sinopsis, actores o año al algoritmo basado en contenido.
  + **Falta de contenido**: Si bien puede parecer que 9.000 películas es un número más que aceptable, se evidencia claramente la falta de series en el catálogo de la aplicación. Hoy en día, este tipo de contenido es aún más consumido que las películas, de modo que carecer de él supone una gran debilidad
  + **Difícil entrada al mercado**: Aunque en la era en la que vivimos pueda parecer que resulta muy sencillo hacerse un lugar en Internet, lo cierto es que sin una buena campaña de marketing esto puede resultar un obstáculo gigantesco para cualquier proyecto de nueva creación.
  1. Amenazas
  + **Existencia de competidores asentados**: En el mercado se pueden encontrar empresas como IMDb, que tienen una base de datos de películas y valoraciones gigantesca sobre la que han implementado un sistema de recomendación.
  + **Falta de recursos inicial**: Como en casi todos los negocios, el comienzo puede resultar bastante duro, pues es necesario alquilar servidores con buena potencia para que sean capaces de realizar todos los cálculos asociados a las recomendaciones en un periodo de tiempo corto. Y esto, sin dinero, es muy complicado conseguirlo.
  1. Fortalezas
  + **Ausencia de intereses externos**: Como es lógico, los sistemas de recomendación que utilizan las plataformas de streaming están influenciados, lo que hace que por norma general recomienden en mayor medida el contenido que esas propias plataformas han producido. Esta aplicación, por el contrario, es totalmente transparente y no se encuentra influenciada por ningún factor externo.
  + **Facilidad de uso:** A la hora de diseñar la interfaz de la aplicación, se ha tenido mucho en cuenta la accesibilidad de los usuarios, de modo que cualquier persona independientemente de sus conocimientos informáticos pueda utilizarla sin ningún tipo de problema.
  1. Oportunidades
  + **Aplicación móvil**: Aunque la aplicación esté diseñada para ser utilizada en un ordenador, también puede utilizarse sin ningún tipo de problema en un dispositivo portable, lo que hace que un usuario pueda valorar películas y obtener recomendaciones con un simple click en su teléfono móvil.
  + **Sector digital en constante desarrollo**: En la actualidad, cada vez son más las personas que utilizan dispositivos electrónicos para realizar todo tipo de acciones. Por ello, el número de potenciales consumidores es gigantesco

1. Líneas de futuro

Debido al reducido periodo de tiempo en el que he tenido que desarrollar la aplicación, no he sido capaz de abarcar todos los campos que en un principio me habría gustado. En un principio, siendo un completo principiante en la materia, se me pasaron un montón de cosas por alto, pero tras todo el tiempo que le he dedicado al proyecto, un montón de mejoras y de oportunidades de negocio pasan ahora mismo por mi mente.

Para empezar, es necesario desarrollar una aplicación móvil. Aunque una aplicación web esté diseñada para ser utilizada tanto en dispositivos móviles como en ordenadores, siempre resulta mucho más cómodo utilizar una aplicación nativa. En la práctica, las aplicaciones web visualizadas desde Smartphone acaban teniendo una apariencia y una usabilidad bastante inferior a la misma aplicación visualizada desde una pantalla de ordenador.

El sector móvil es un sector en constante desarrollo y evolución. No es ni mucho menos una casualidad que casi todas las empresas por diversos que puedan ser los negocios a los que se dediquen estén totalmente empeñadas en que te descargues su aplicación. Desde bancos hasta tiendas, todas estas empresas han detectado que parte de su futuro se encuentra estrechamente ligado a las aplicaciones móviles y, como es lógico, han actuado en consecuencia.

Por estas razones considero que es totalmente prioritario desarrollar una aplicación móvil a la par que la aplicación web. Si quieres entrar en el mercado en condiciones, sin duda has de hacerlo con estas dos aplicaciones de la mano.

Otra línea de futuro que sin duda tendría que ser explorada es el aumento de tamaño de la base de datos. Al partir de una base de datos generada por un tercero, siempre echaremos en falta un montón de información. En mi caso, sin ir más lejos, echo en falta que la base de datos no tenga información alguna sobre series. Al igual que también me gustaría que contuviera películas más novedosas, más información sobre el reparto, una pequeña descripción de la película, keywords…

Desde luego, parece demasiado optimista pensar que alguien va a aparecer por arte de magia en Kaggle y publicar una base de datos de estas características. Por ello, lo más efectivo siempre será crear nuestra propia base de datos. ¿Y cómo podemos hacer eso? La respuesta es sencilla, scrapeando.

Mediante minería de datos (comúnmente denominada web scraping) somos capaces de explorar una página web como IMDb de forma automática y descargar toda esa información que podamos necesitar: películas, géneros, directores, reparto, palabras clave, descripciones, valoraciones… Absolutamente toda la información que podamos necesitar está disponible en internet a día de hoy. Si quieres construir una buena base de datos, tan solo tienes que aprender web scraping, el resto es pan comido.

Por último, cabe destacar otra línea de futuro que, si bien no es tan relevante como las dos anteriores, puede ayudar a diferenciar la aplicación en el mercado con respecto a sus competidoras.

Hay que tener una cosa muy clara con respecto a la aplicación. No es ni será nunca una aplicación de visualización de contenido. En base a esto, se presenta el siguiente problema. ¿Por qué un determinado usuario iba a molestarse en valorar películas en la aplicación si luego no va a ser capaz de encontrar donde visualizar todo el contenido recomendado? Es algo que me preocupa mucho y que puede llegar a suponer un problema crítico a la aplicación.

Para solventarlo, he pensado en lo siguiente: mostrarles dónde pueden ver el contenido que se les recomienda. Me explico. Mediante web scraping, podemos obtener todo el contenido disponible de cada una de las plataformas de streaming existentes. Si almacenamos esa información en la base de datos, podemos mostrar al usuario en qué plataformas está disponible el contenido que se le recomienda.

De esta sencilla manera, somos capaces de encontrar algo que nos diferencia del resto de competidoras, algo que nos hace únicos. Además, también somos capaces de generar valor a las plataformas de streaming. No sería algo generalizado ni mucho menos, pero en la práctica podría acabar ocurriendo que un determinado usuario acabara suscribiéndose a una determinada plataforma de streaming para ver una determinada película o serie e incluso quién sabe si también quedándose para siempre. Eso ya dependería de la plataforma en cuestión, nosotros somos y siempre seremos meros intermediarios.

1. Lecciones aprendidas

He de reconocer que esta aplicación ha supuesto todo un reto para mí, pero es precisamente por eso por lo que he aprendido cosas que ni siquiera me imaginaba al matricularme en la asignatura.

Lo primero que aprendí fueron las bases de datos orientadas a grafos de conocimiento. He de reconocerlo, antes de empezar la asignatura desconocía por completo su existencia. Jamás había oído hablar de ellas. Sin embargo, al tener que aprender su funcionamiento para el desarrollo de la asignatura, descubrí un mundo super interesante que nos permite hacer cosas que, con bases de datos relacionales, nos supondrían un auténtico quebradero de cabeza. Recomendaciones en tiempo real, detección de fraude… Son sólo algunas de las múltiples aplicaciones que se pueden realizar con este tipo de bases de datos.

Acto seguido, tuve que aprender a interactuar con una base de datos orientada a grafos, Neo4j para ser más concreto. Al igual que las bases de datos relacionales o las bases de datos orientadas a documentos, este tipo de bases de datos también tienen su propio lenguaje mediante el que realizar consultas. El lenguaje de Neo4j, que se está convirtiendo en un Standard en el sector, se denomina Cypher. Este lenguaje, que se nota que está influido por SQL, es extremadamente sencillo de aprender si conoces las bases de SQL. Es cierto que la sintaxis es algo diferente, pero la lógica sobre la que opera es prácticamente idéntica a SQL.

Para aprender Cypher, decidí realizar un curso en la academia de Neo4j [1]. Un curso totalmente gratuito en el que se puede aprender de forma perfecta todo lo que un desarrollador pudiera necesitar para comenzar un proyecto en el que se utilicen bases de datos Neo4j.

Tras haber aprendido todo lo necesario sobre bases de datos orientadas a grafos y Cypher, llegó el momento de ponerse manos a la obra con los sistemas de recomendación. Cuando empecé el proyecto, tenía algo de idea sobre como funcionaban estos sistemas, pero al ponerme a desarrollar de verdad me di cuenta de que lo que realmente sabía y nada no estaban demasiado lejos.

El mundo de los sistemas de recomendación es un mundo extremadamente complejo, especialmente si, como yo, te gusta entender la lógica matemática que respalda el funcionamiento de las cosas.

En un principio, en vistas a ir observando qué tipos de sistemas de recomendación se podían desarrollar utilizando Neo4j, me leí los capítulos 4, 5 y 7 del libro *Graph-Powered Machine Learning* [24]. Este libro me sirvió como iniciación en los sistemas de recomendaciones, pudiendo aprender con ejemplos la lógica que deben seguir este tipo de sistemas. Sin embargo, a medida que fui profundizando un poco en el libro, me di cuenta de que los algoritmos desarrollados difícilmente iban a poder ser escalados a una base de datos de tamaño grande. El motivo es muy sencillo. Las bases de datos, y especialmente sus lenguajes de consulta, han sido diseñado para efectuar inserciones, borrados y búsquedas. Esto quiere decir que en ninguno de los casos han sido diseñados para realizar operaciones matemáticas por simples que sean.

Los algoritmos que en el libro se desarrollaban, podían funcionar con vectores de 10 características o con un número de relaciones insignificante. Sin embargo, ¿qué iba a pasar con 9.000 películas y 1.000.000 de relaciones? Lo más seguro es que los tiempos de espera fueran totalmente inaceptables, nadie quiere esperar 3 minutos para absolutamente nada cuando utiliza una aplicación.

Debido a esto, decidí buscar algún otro libro que explicara los sistemas de recomendación de forma teórica, permitiéndome así implementar mis algoritmos desde 0. De entre todos los libros que encontré, el que más me llamó la atención fue *Recommender Systems. The textbook* [25], que viene a ser algo así como La Biblia de los sistemas de recomendación. Leyendo este libro, que describe de forma perfecta el funcionamiento de todos los tipos de sistemas de recomendación existentes. Es un libro fundamentalmente matemático, en el que se pueden aprender todo tipo de conceptos acerca de este tipo de sistemas: cálculo de similitudes, coeficiente de Gini, tipos de hibridación,

1. Bibliografía

|  |  |
| --- | --- |
| [1] | «Neo4j GraphAcademy,» [En línea]. Available: https://neo4j.com/graphacademy/online-training/v4/00-intro-neo4j-about/. |
| [2] | «API Javascript,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript. |
| [3] | «API Vue,» [En línea]. Available: https://vuejs.org/. |
| [4] | «API Vuetify,» [En línea]. Available: https://vuetifyjs.com/en/. |
| [5] | «API Vue-router,» [En línea]. Available: https://router.vuejs.org/. |
| [6] | «API Vuex,» [En línea]. Available: https://vuex.vuejs.org/. |
| [7] | «API Vuex-persistedstate,» [En línea]. Available: https://www.npmjs.com/package/vuex-persistedstate. |
| [8] | «API axios,» [En línea]. Available: https://www.npmjs.com/package/axios. |
| [9] | «API nodejs,» [En línea]. Available: https://nodejs.org/es/. |
| [10] | «API Flask,» [En línea]. Available: https://palletsprojects.com/p/flask/. |
| [11] | «API Python,» [En línea]. Available: https://es.python.org/. |
| [12] | «Vídeo Flask,» [En línea]. Available: https://www.youtube.com/watch?v=GMppyAPbLYk. |
| [13] | «Código Flask,» [En línea]. Available: https://github.com/neo4j-examples/neo4j-movies-template/tree/master/flask-api. |
| [14] | «API Pandas,» [En línea]. Available: https://pandas.pydata.org/. |
| [15] | «API Numpy,» [En línea]. Available: https://numpy.org/. |
| [16] | «API Neo4j Python,» [En línea]. Available: https://neo4j.com/developer/python/. |
| [17] | «API Sklearn,» [En línea]. Available: https://scikit-learn.org/stable/. |
| [18] | «API Scipy,» [En línea]. Available: https://www.scipy.org/. |
| [19] | «Kaggle,» [En línea]. Available: https://www.kaggle.com/. |
| [20] | «TMDB 5000 Movie Dataset,» [En línea]. Available: https://www.kaggle.com/tmdb/tmdb-movie-metadata?select=tmdb\_5000\_movies.csv. |
| [21] | «The movies dataset,» [En línea]. Available: https://www.kaggle.com/rounakbanik/the-movies-dataset. |
| [22] | «Movielens,» [En línea]. Available: https://grouplens.org/datasets/movielens/. |
| [23] | «Extension Movielens,» [En línea]. Available: http://files.grouplens.org/datasets/hetrec2011/hetrec2011-movielens-readme.txt. |
| [24] | A. Negro, Graph-Powered Machine Learning, Manning. |
| [25] | C. C. Aggarwal, Recommender Systems. The textbook, Springer, 2016. |