UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

18|19

Trabajo fin de grado

Framework orientado a algoritmos de recomendación basados en similitudes

Alberto García Redero

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

excelencia Campus Internacional UAM CSIC

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Framework orientado a algoritmos de recomendación basados en similitudes

Autor: Alberto García Redero
Tutor: Alejandro Bellogín Kouki
Ponente: Fernando Díez Rubio

junio 2019

Alberto García Redero
Framework orientado a algoritmos de recomendación basados en similitudes

Alberto García Redero
C\ Marqués de la Valdavia Nº 109

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi familia y amigos

We can only see a short distance ahead,
but we can see plenty there that needs to be done.

Alan Turing

Prefacio

El trabajo de fin de grado expuesto a continuación con título "Framework orientado a algoritmos de recomendación basados en similitudes", tiene como finalidad el estudio e investigación de las metodologías de recomendación existentes utilizando vecinos próximos (KNN), al igual que el diseño e implementación de un framework a partir del cual podamos comparar las recomendaciones realizadas por el algoritmo de vecinos con nuevas variantes tal como la combinación de distintas redes neuronales con vecinos próximos. Para hacer esto posible, el framework permite obtener los factores latentes que la red extrae de los usuarios y los items, para así poder comprobar si el algoritmo de vecinos mejora los resultados al tener por entrada unos datos preprocesados de los cuales hemos obtenido ya previamente sus características.

Este proyecto ha sido realizado con la intención de poder añadir fácilmente nuevos conjuntos de datos y nuevas redes que permitan realizar distintas pruebas en el futuro, así como la mayor sencillez posible a la hora de utilizarlo. Permitiendo que, por ejemplo, el proceso de extracción completo de los vecinos de cierta red se pueda completar en dos pasos y sin que el usuario deba realizar en ningún momento un tratamiento de los datos por su parte.

Alberto García Redero

Agradecimientos

En primer lugar querría agradecer a Alejandro Bellogín la oportunidad de poder realizar con él este trabajo, así como su ayuda y guía siempre que la he necesitado.

También me gustaría agradecerles a mis padres por haber sido los pilares en los que me he podido apoyar en todo momento, gracias por enseñarme que si quieres hacer algo lo único que importa es tu actitud.

Gracias a mi hermano Rodrigo por siempre haber estado ahí.

Finalmente me gustaría agradecer a mis amigos por alegrarme siempre y ayudarme a ver las cosas con perspectiva.

Resumen

Internet ha crecido exponencialmente en los últimos años. Es casi imposible encontrar un lugar en el que no puedas de una manera u otra acceder a la red, pero tampoco podemos imaginarnos lo que sería internet sin los algoritmos de recomendación. Nos ayudan a la hora de buscar información, de encontrar nuevos amigos, de descubrir nuevas series o de comprar artículos entre otras cosas. Y es que internet gracias a estos, es en gran medida lo que ha llegado a ser. La mayoría gastamos horas y horas únicamente vagando por la red, descubriendo algo nuevo gracias a vídeos o escuchando nueva música. Por esto cada día se da una mayor importancia al rendimiento de estos algoritmos, es importante conseguir el mejor acierto en menor tiempo para lograr atraer al usuario.

En este Trabajo de Fin de Grado se realiza una investigación sobre el actual uso del algoritmo de vecinos próximos y de sus distintas variantes ya sea bien orientado a usuarios u orientado a items y las distintas métricas utilizadas. De la misma manera se profundiza en el estudio de nuevas variantes, especialmente se hace foco en el uso de los factores latentes y cómo se pueden extraer gracias a las redes neuronales para así poder ser utilizados en combinación con vecinos próximos.

Por otro lado se realiza una investigación de cómo puede afectar al resultado de vecinos la extracción de los factores latentes por un tipo de red o por otro. También se procederá a comprobar distintos parámetros, con el fin de dar con los de mayor importancia y que puedan asegurarnos unos mejores resultados.

Palabras clave

Sistema de recomendación, algoritmo, framework, redes neuronales, vecinos próximos (KNN)

Abstract

Internet has grown exponentially in the last years. It is almost impossible to find a place where you can not access the Internet, but neither we can not imagine what internet would be without the recommendation algorithms. They help us when looking for information, to find new friends, to discover new series or to buy items among other things. As a result, internet has become what it is nowadays. Most of us spend hours and hours just surfing the internet, discovering something new thanks to videos or listening to new music. This is why every day greater importance is given to the performance of these algorithms, it is important to achieve the best success in less time to attract the user.

In this bachelor thesis, we will explore the current use of the algorithm of nearest neighbours, such as its different variants (whether it is user or item oriented) and the most common metrics. In the same way we go further into the study of new variants, specially the use of embeddings and how it can be extracted thanks to the neural networks in order to combine them with nearest neighbours.

On the other hand, an investigation of how the embeddings extracted by different neural networks can affect the result shall take place. Also it will be verified the importance of different parameters, in order to find the most importants which can lead us to better results.

Keywords

Recommendation system, algorithm, framework, neural networks, nearest neighbours (KNN)

Índice

Listas

Lista de códigos

Lista de figuras

1

Introducción

Con el avance de internet muchas grandes compañías han surgido o se han expandido aprovechando este, además, es indiscutible el poder que otorga un correcto uso de los datos que las grandes compañías recolectan diariamente. Uno de los mayores campos en los que estos se utilizan son las recomendaciones al usuario, aportando mayores beneficios a la entidad y mejorando la experiencia del cliente al utilizarla. Gracias a esto empresas de la talla de Amazon o Google han llegado a ser lo que hoy son. Por ponernos en un contexto, en el año 2013, un 35 % de las compras en Amazon y un 75 % de los vídeos vistos en Netflix fueron gracias a recomendaciones realizadas a los usuarios [1]. Por todo esto cada vez se le da una mayor importancia a la experiencia del usuario y por tanto a las recomendaciones que el sistema le presenta, causando que la exploración hacia nuevas y mejores técnicas sea de vital importancia.

1.1. Motivación del proyecto

Los sitios web de mayor importancia ofrecen millones de productos diariamente a los usuarios y elegir entre estos supone un gran esfuerzo. Para solventar este problema surgieron los sistemas de recomendación, ofreciendo a cada cliente los productos que mejor se adapten a sus gustos. Aunque con el paso de los años, el enorme crecimiento de estos ha planteado algunos desafíos clave para los sistemas de recomendación. Estos son: producir recomendaciones de alta calidad y realizar muchas recomendaciones por segundo para millones de clientes y productos. Con el fin de cumplir estos objetivos se creó el algoritmo de vecinos próximos, el cual se continua utilizando hoy en día dada su gran eficacia.

Por otro lado, vivimos en una época en la cual dado el gran rendimiento de las redes neuronales, se está explorando el uso de estas en casi todos los campos. Desde reconocimiento de imágenes al análisis de datos [2], no hay campo en el que no se haya experimentado con estas y las recomendaciones no son una excepción. Una de las mejores cualidades de las redes neuronales es su capacidad de caracterizar elementos y esto puede ser explotado conjuntamente por otro algoritmo, como vecinos próximos.

Con el fin de entender y explorar los mejores métodos de recomendación y los más novedosos, en este trabajo de fin de grado se ha realizado un framework el cual permite combinar el algoritmo de vecinos próximos con las redes neuronales, permitiéndonos de una manera sencilla obtener los vecinos de cierto item o usuario y así pudiendo comparar su rendimiento con vecinos sin ayuda externa, para poder saber si ciertamente la combinación de ambos métodos aporta una mejora en los resultados de la recomendación.

1.2. Objetivos

Como hemos introducido en el apartado anterior el objetivo de este trabajo es la comparación del algoritmo de vecinos próximos sobre los datos sin modificar contra el rendimiento de este si previamente a su uso hemos extraído usando redes neuronales los factores latentes de los usuarios y los ítems que conforman el espacio inicial.

Para permitirnos lograr este objetivo debemos cumplir ciertos requisitos:

- Estudio de la actual situación de vecinos próximos y las posibles variantes de este.
- Estudio de las redes neuronales y cómo pueden utilizarse en recomendación.
- Estudio de los posibles tipos de combinación de las redes neuronales y vecinos próximos.
- Capacidad de utilización de diversos conjuntos de datos, sin especializarnos en uno solo.
- Prueba de diversas redes neuronales con el fin de comprobar si las redes en general mejoran o empeoran el rendimiento, o cuales son mejores para este trabajo.
- Capacidad de generar recomendaciones a partir de los factores latentes obtenidos por la red.
- Propuesta de nuevos métodos de recomendación utilizando redes neuronales y vecinos.

1.3. Estructura del trabajo

El presente documento está dividido en cinco partes, estas son:

- **Capítulo 1. Introducción:** Descripción de la situación actual de los sistemas de recomendación, motivaciones a la hora de realizar este proyecto y objetivos del mismo.
- **Capítulo 2. Estado del Arte:** Introducción a los sistemas de recomendación, vecinos próximos y a las redes neuronales. Expone las variantes de combinación de ambos algoritmos para la generación de recomendaciones.
- **Capítulo 3. Diseño e Implementación:** Explicación del proyecto realizado partiendo de la base teórica introducida previamente, así cómo el por qué de las decisiones tomadas y los diagramas explicativos del framework.
- **Capítulo 4. Pruebas y Resultados:** Análisis y comparación de los resultados obtenidos con cada una de las variantes experimentadas.
- **Capítulo 5. Conclusiones y Trabajo Futuro:** Argumentación de las conclusiones obtenidas tras la realización del proyecto y tareas a realizar en el futuro con el fin de continuar y ampliar el proyecto.

2

Estado del arte

En este apartado se explicará toda la base teórica en la que el trabajo se ha basado, introduciendo los sistemas de recomendación y profundizando en mayor medida en el algoritmo de vecinos próximos. También se hará una breve introducción a las redes neuronales y cómo estas pueden ser utilizadas en recomendación. Finalmente se explicarán métodos de combinación de ambos algoritmos con el fin de mejorar los resultados obtenidos por vecinos próximos.

2.1. Los sistemas de recomendación

Todos los usuarios de internet han utilizado un sistema de recomendación de una manera u otra. Por ejemplo, imagina que un amigo te recomienda un libro. Para poder comprarlo lo buscas en una tienda online y tras haber encontrado el ejemplar que necesitabas puedes observar, con casi total seguridad, que en una parte de la pantalla aparece un apartado con libros que otros usuarios que compraron el que buscabas también han comprado. Si sueles comprar libros, seguramente la tienda te muestre un listado de libros que puedan interesarte nada más entrar de nuevo. Esto es realizado gracias a los sistemas de recomendación.

En este ejemplo podemos ver algunas de las características de los sistemas de recomendación. En un principio es fácil notar que el sistema nos aporta libros que puedan interesarte ofreciéndote así una recomendación personalizada. Por otro lado, normalmente se puede mostrar un listado con los libros más comprados, que puede no ser tan personal pero puede dar grandes resultados al ser elementos de interés general.

La recomendación personalizada es una rama de estudio en pleno auge y con constantes innovaciones. Desde el principio de los años 90, se mantiene mejorando utilizando cada vez algoritmos más sofisticados. En este Trabajo de Fin de Grado vamos a centrarnos en los algoritmos con recomendaciones personalizadas, de los cuales los principales son: basados en contenido y de filtrado colaborativo.

2.1.1. Algoritmos Basados en Contenido

Los algoritmos basados en contenido intentan recomendar al usuario ítems similares a aquellos que le han gustado a este en el pasado. Estos para buscar cuánto se parecen dos ítems, comprueban y comparan las características de ambos, basándose en la premisa de que si a un usuario le ha gustado por ejemplo una película de acción sienta mayor afinidad a ver otra del mismo género a, por ejemplo, ver una de romance. Otra manera de obtener información sobre el contenido puede ser por ejemplo un documento de texto, donde gracias a TF-IDF (Term Frequency-Inverse Document Frequency) podamos comprender el contenido del mismo.

2.1.2. Algoritmos de Filtrado Colaborativo

2.1.2.1. Filtrado colaborativo basado en modelos

El principal problema de las estrategias basadas en vecinos es que la complejidad del cálculo es enorme. Por ello surgieron las estrategias basadas en modelos, que construyen modelos estadísticos de patrones de valoración de usuarios y productos para poder realizar las predicciones. Entre los métodos que hacen uso de esta estrategia podemos encontrar las redes neuronales, los modelos probabilísticos o, los más destacados: los modelos de factores latentes. Este método es el más utilizado por la mejora de precisión que ha proporcionado gracias a la factorización de matrices.

Factorización de Matrices

Supongamos tener una matriz de IDs de usuarios y sus interacciones con los productos. Las filas corresponderán a los usuarios y las columnas a los ítems. En la matriz, cada entrada contiene la calificación dada por un usuario a un elemento determinado, ya sea bien explícitamente o implícitamente.

	i_1	i_2	i_3	i_4	...	i_n
u_1	1		5			
u_2			4			2
u_3				3		
u_4		1		5		
...						
u_m			2			

Figura 2.1: Ejemplo de matriz de rátings. [3]

Definición del método de factorización de matrices

La matriz de calificaciones **R**, está compuesta por entradas r_{ij}, las cuales son las calificaciones del usuario *i* al ítem *j*. En la mayoría de los casos estas matrices son grandes, con millones de datos, y dispersas, lo que quiere decir que cada usuario únicamente ha interactuado con una minúscula parte del conjunto de ítems. Siendo en muchos ejemplos vacía en un 99 %.

La factorización de matrices pretende extraer el conjunto de atributos comunes existentes para todos los elementos, marcando las diferencias entre estos por el grado que expresan. Así, el rating otorgado por un usuario a un ítem se puede aproximar con la suma de la atracción del usuario a cada atributo en función del grado por el que cada ítem tiene expresado dicho atributo, estos son los **factores latentes**. Un ejemplo en música serían los géneros y los autores. Se puede ver intuitivamente que cierto usuario puede tener mayor interés por un género que por otro.

Otra de las grandes características de la factorización de matrices reside en el hecho de que no necesita saber el número de factores latentes que permitan comprender los gustos de un usuario, ya que supone que existen un número arbitrario.

Transformación de la matriz para obtener los factores latentes

Para un conjunto de usuarios de tamaño *u* y de ítems de tamaño *i*, se eligen un número arbitrario *d* de factores latentes y se factoriza la matriz en dos de menores dimensiones **U** y **V**

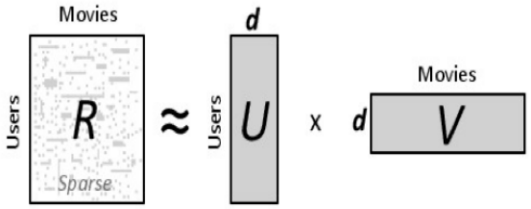


Figura 2.2: Transformación de la matriz. [3]

Esto produce una compresión de la información dispersa en **R** en espacios de menores dimensiones *d*x*u* y *d*x*i*. Cada usuario es representado con un vector de este nuevo espacio dimensional, sea x_u en **U** el vector con los factores que representan el interés del usuario por estos. De igual manera, y_i en **V** corresponde a cuanto expresa el ítem a estos factores.

Para poder calcular la calificación del usuario *u* al ítem *i*, debemos realizar el producto escalar de ambos vectores:

$$r_{ui} = x_u^T * y_i \tag{2.1}$$

Métodos utilizados para la Factorización de Matrices

A continuación vamos a enumerar algunos de los métodos de Factorización de Matrices más comunes según [4]:

- **Factorización LU**
- **Factorización de Cholesky**
- **Factorización LDL^T**
- **Factorización QR**
- **Descomposición en valores singulares**

2.1.2.2. Filtrado colaborativo basado en memoria

También llamadas estrategias basadas en vecinos. Son las que realizan las recomendaciones en función de las valoraciones dadas por los usuarios. Estos sistemas son los más utilizados a la hora de recomendar. Se basan en aprovechar el interés de un usuario hacia cierto ítem, ya bien sea implícitamente dada por una puntuación del usuario o por cierta información que podamos obtener sobre sus gustos tales como el tiempo de reproducción de un vídeo, o los sitios a los que accede. Esto es explotado ya que si ciertos usuarios u y v son similares, se le recomendarán a u ítems que le hayan gustado a v. Este método tiene la gran ventaja de evitar realizar recomendaciones encasilladas en un único tipo, ya que permite adaptar las recomendaciones a los gustos de la comunidad. Pero se complica el hecho de recomendar a usuarios con gustos diferentes a la mayoría si el conjunto de datos no es muy amplio [5] [6].

Las ventajas que aportan los mecanismos basados en memoria son:

- Son fáciles de implementar.
- Permite que los usuarios y los ítems tengan cualquier estructura y característica.
- Obtiene mayoritariamente buenos resultados.

Por otro lado, los inconvenientes que tienen son:

- Requieren un gran número de puntuaciones y poco dispersas.
- No tienen en cuenta el contexto, sólo las puntuaciones.
- En sistemas de millones de elementos buscar en la tabla de puntuaciones puede ser poco escalable implicando un coste considerable.

Basado en usuario

Tal como hemos explicado antes se basan en que si queremos predecir la reacción de cierto usuario ante un nuevo ítem, esta se asemejará a la que tuvieron los usuarios parecidos a este.

$$r(u,i) = c * \sum_{v \in N_k(u); r(v,i) \neq 0} sim(u,v) * r(v,i) \tag{2.2}$$

Siendo c, la cual es recomendable al producir mejores resultados, la normalización de la fórmula:

$$c = \frac{1}{\sum_{v \in N_k(u)} sim(u,v)} \tag{2.3}$$

Basado en ítem

Similar al basado en usuario, con la diferencia de que la importancia que le puede dar el usuario al nuevo ítem será similar a la importancia que le dio a ítems parecidos a este nuevo.

$$r(u,i) = c * \sum_{v \in N_k(i); r(u,j) \neq 0} sim(i,j) * r(u,j) \tag{2.4}$$

Siendo c:

$$c = \frac{1}{\sum_{v \in N_k(i)} sim(i,j)} \tag{2.5}$$

Este rating predicho necesita una función de similitud entre los usuarios o los ítems, la cual puede ser calculada de distintas maneras. A continuación explicaremos las de mayor importancia:

Funciones de similitud

- **Coseno**

  Función más común de todas, cada elemento es representado por un vector. Así pues, la similitud entre dos elementos es la distancia coseno entre los dos vectores que se busca comparar. Estos vectores pueden ser por ejemplo en el caso de los usuarios, los rátings que han dado a cada uno de los ítems. En caso contrario el vector de cada ítem estará formado por los rátings que los usuarios les han otorgado.

  - **Coseno orientado a usuarios:**

$$sim(u,v) = cos(u,v) = \frac{\sum_{i \in I_{uv}} r_{ui} * rvi}{\sqrt{\sum_{i \in I_u} r_{ui}^2 * \sum_{j \in I_v} r_{vj}^2}} \tag{2.6}$$

  - **Coseno orientado a ítems:**

$$sim(i,j) = cos(i,j) = \frac{\sum_{u \in U_{ij}} r_{ui} * ruj}{\sqrt{\sum_{u \in U_i} r_{ui}^2 * \sum_{u \in U_j} r_{uj}^2}} \tag{2.7}$$

- **Pearson** Función parecida al coseno, pero que incluye una manera de normalizar los datos. No todos los usuarios puntúan los ítems con el mismo criterio, cierto usuario puede tender a poner notas más altas que otro aunque

a ambos les haya gustado por igual el ítem. Por ello se suelen normalizar los datos con la media de los rátings dados por el usuario, obteniéndose la correlación de Pearson entre los datos de cada usuario o item.

- **Pearson orientado a usuarios:**

$$sim(u,v) = Pearson(u,v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \overline{r}_u) * (r_{vi} - \overline{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \overline{r}_u)^2 * \sum_{i \in I_{uv}} (r_{vi} - \overline{r}_v)^2}} \tag{2.8}$$

- **Pearson orientado a ítems:**

$$sim(i,j) = Pearson(i,j) = \frac{\sum_{u \in U_{ij}} (r_{ui} - \overline{r}_i) * (r_{uj} - \overline{r}_j)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \overline{r}_i)^2 * \sum_{u \in U_{ij}} (r_{uj} - \overline{r}_j)^2}} \tag{2.9}$$

- **Jaccard** Función que opera entre conjuntos, mide la similitud entre las características de dos elementos. Siendo 0 en caso de no tener nada en común y 1 en caso de ser iguales.

$$sim(A,B) = Jaccard(A,B) = \frac{|A \cap B]}{|A \cup B]} = \frac{|A \cap B]}{|A| + |B| - |A \cap B]} \tag{2.10}$$

2.1.2.3. Sistemas Híbridos

Los algoritmos explicados anteriormente tienen ciertas limitaciones, como por ejemplo el exceso de información por cada elemento o en su defecto la falta, que pueden provocar que el sistema no trabaje correctamente haciéndole demasiado costoso o incapaz de tomar ciertas decisiones. Problemas con los nuevos usuarios o ítems, etc.

Para resolver este problema se utilizan los sistemas híbridos, los cuales combinan varios de estos para así lograr mejorar los problemas de uno de estos. Estos se pueden dividir en [6]:

- **Ponderado:** Combina los resultados de los distintos sistemas de recomendación dando un peso distinto a cada uno.
- **Mixto:** Muestra los resultados de varios sistemas de recomendación a la vez.
- **Intercambio:** El sistema elige uno de los algoritmos de recomendación a partir de ciertos criterios o consecuencias.
- **Cascada:** Utiliza los resultados de un sistema de recomendación como entrada de otro con el fin de mejorar los resultados de este.
- **Combinación de características:** Aprovecha las mejores características de los sistemas que los conforman para crear un único sistema de recomendación.

En este Trabajo de Fin de Grado vamos a intentar mejorar el rendimiento de vecinos próximos, por lo que nos centraremos en utilizar el método de cascada. Utilizaremos la red neuronal para preprocesar los datos y así obtener los factores latentes de cada elemento, que serán utilizados por vecinos para realizar sus predicciones. Para hacer esto posible vamos a estudiar una manera de obtener los factores latentes, la factorización de matrices.

2.1.3. Métricas de evaluación

Tras la creación de un sistema de recomendación siempre hay que comprobar cuan bueno es. Dependiendo de lo que se busque comprobar unas métricas serán más útiles que otras. A continuación hablaremos de las más importantes, profundizando en las más utilizadas.

- **Error cuadrático medio (RMSE):** Medida de precisión, mide la media de los errores al cuadrado. Comparando errores de predicción de distintos modelos para un conjunto de datos en particular.
- **Error absoluto medio (MAE):** Muy parecido al RMSE con la diferencia de calcular la distancia entre ambos valores de rating, real y predicho, con el valor absoluto y no con la raíz de los cuadrados. Por esto penaliza menos los errores grandes que RMSE
- **Precisión, at K (P@K):** Para explicar esta métrica debemos comenzar por su variante más sencilla: Precisión. Esta calcula la división entre los elementos devueltos relevantes, verdaderos positivos (TP) entre el número total de elementos recomendados, siendo estos los elementos relevantes devueltos y los no relevantes, falsos positivos (FP). Su variante, Precisión at K, comprueba la precisión retornada hasta cierto valor K de elementos. Esto nos permite observar el comportamiento más en detalle de los elementos devueltos, permitiéndonos jugar con K y comprobar si nuestro sistema devuelve los elementos correctos en las primeras posiciones o cómo estos están distribuidos.

$$Precision@k = \frac{TP}{TP + FP} = \frac{TP}{N} \tag{2.11}$$

- **Recall, at K (P@K):** Similar a Precisión at K, con la salvedad de que divide entre el total de los elementos relevantes del conjunto para un usuario, que son los devueltos por el sistema, verdaderos positivos (TP) y los no devueltos, falsos negativos (FN).

$$Recall@k = \frac{TP}{TP + FN} \tag{2.12}$$

- **Normalized Discounted Cumulative Gain:** Utiliza la relevancia de los elementos retornados, comprueba que los elementos de mayor importancia o grado, g(d_k) hayan sido devueltos en los primeros lugares. Para ello, normaliza el valor de DCG que otorga un valor mayor mientras más ordenado esté, gracias a IDCG. Este último calcula el valor DCG de la situación ideal para así asegurarnos un valor nDCG en un rango entre 0.0 y 1.0.

$$nDCG = \frac{DCG}{IDCG} \tag{2.13}$$

$$DCG = \sum_{k} \frac{g(d_k)}{log_2(k+1)} \tag{2.14}$$

$$IDCG = max_{R \in \varsigma(D)} DCG(R, q) \tag{2.15}$$

- **Aggregate diversity (AD):** Devuelve un valor proporcional al número total de ítems que el sistema recomienda.
- **EPC (Expected Popularity Complement):** Número esperado de ítems relevantes recomendados no vistos previamente [7].

- **Gini:** Tiene en cuenta no solo si los ítems han sido recomendados a alguien, sino a cuántas personas y cómo se encuentran distribuidos (de manera uniforme o desigual), donde un valor más alto indica una recomendación más equilibrada y diversa [7].

2.2. Redes Neuronales

Son una herramienta matemática que pretende emular de manera simplificada el funcionamiento de las neuronas en el cerebro. Esto es, la conversión de cierta información codificada en números en cierta nueva información codificada como nuevos números.

En los últimos años han ganado notoria importancia gracias a los grandes logros que se están llevando a cabos con estas. Desde modelos capaces de atravesar captchas hasta imágenes de rostros de personas completamente generadas por la red y con una similitud a una foto de una persona real que hacen que sean imposibles de distinguir. La simpleza del sistema, el cual está compuesto únicamente por neuronas y las conexiones entre estas, es la gran virtud que las permite realizar tareas tan dispares y complejas como las que son capaces de realizar hoy en día. Por esto, actualmente nos encontramos en una exploración constante de hasta dónde pueden llegar y en qué nuevos campos pueden ser utilizadas.

2.2.1. Ejemplos de Redes Neuronales

Redes Neuronales Profundas

Son un tipo de redes neuronales compuesto por múltiples capas en las que se extraen características de la entrada proporcionada al sistema. Un ejemplo típico sería la figura 2.3.

Redes recurrentes

Al contrario que las anteriores no tienen estructura de capas, permitiendo que todas las neuronas estén en contacto entre sí, llegando a producir bucles. Permiten introducir la temporalidad y que la red tenga memoria, ya que los valores introducidos en cierto momento son tratados y se mantienen transitando aunque la entrada cambie por otra nueva. Un ejemplo típico sería la figura 2.4.

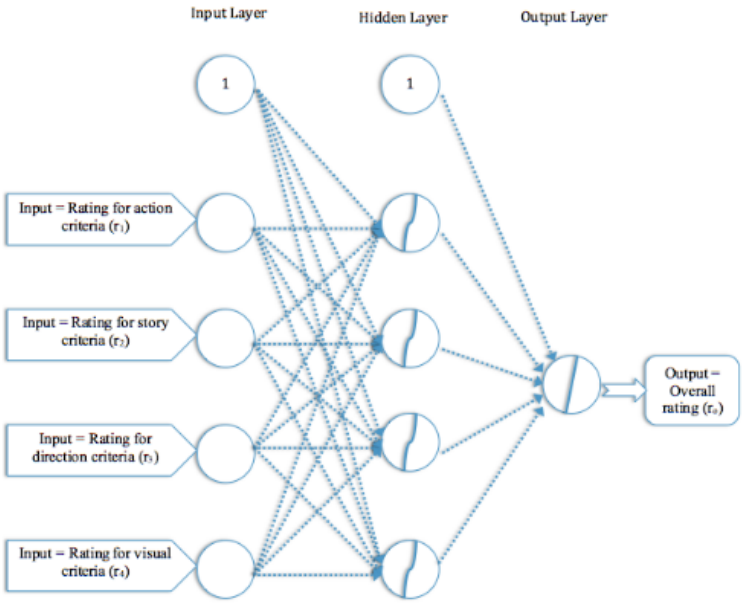


Figura 2.3: Estructura típica de una red profunda. Destacan las capas ocultas y las bias, con valor constante igual a 1 [8].

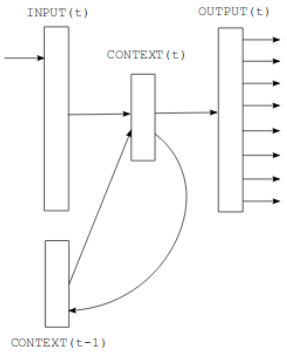


Figura 2.4: Ejemplo sencillo de red recurrente [9].

Redes convolucionales

Muy similares a las redes profundas, en muchos lugares las consideran un subtipo de estas, mantienen la idea de una estructura en capas, pero en la que las neuronas de una capa no reciben conexiones desde todas las neuronas de la capa anterior, sólo de algunas. Esto permite otorgar a las neuronas la característica de la especialización, haciendo que ciertas neuronas busquen ciertas características de la entrada. También tienen la ventaja de que al disminuir conexiones el número de operaciones disminuye mejorando los tiempos de ejecución y los recursos necesarios.Un ejemplo típico sería la figura 2.5.

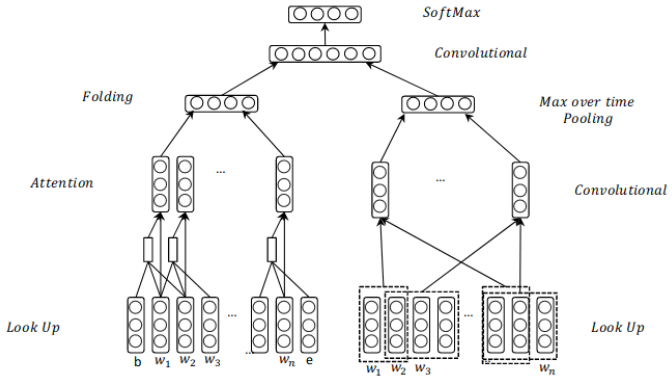


Figura 2.5: Ejemplo de red convolucional. [10]

2.2.2. Conceptos básicos de las Redes Neuronales

Como hemos comentado anteriormente las redes neuronales están compuestas por neuronas y la conexión entre estas. Para lograr obtener los resultados deseados, al igual que en nuestro cerebro [11], debemos jugar con la activación de unas neuronas u otras según sea necesario. La función de activación controla el encendido o apagado de una neurona en función de unas condiciones matemáticas dadas por el contexto. Por otro lado, la red neuronal necesita una manera de evaluar y de tomar el siguiente paso para lograr mejorar. La evaluación se realiza gracias a la función de pérdida que calcula cuanto dista el resultado final del deseado. Para minimizar esta, la red ajusta sus neuronas en función del gradiente, que es una medida que permite a la red tener una idea de qué decisiones debe de tomar para encontrar la mejor situación posible.

Función de Activación

Los nodos aportan un valor a sus salidas. Este valor se propaga por la red gracias a conexiones en una única dirección hacia otros nodos de la red. A su vez asociada a cada conexión hay un peso sináptico llamado w_{ij}, que determina el efecto del nodo j al nodo i.

- **Función Softmax:** Rango de valores entre 0 y 1. Tiene varias características:

- Diferenciable, podemos calcular la pendiente de la curva en cualquier par de puntos.
- Monótona.
- Por su lógica, puede causar que la red se quede atascada en entrenamiento.

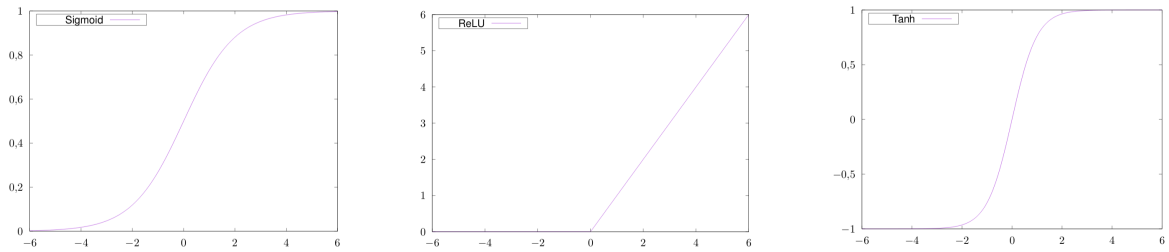
Ejemplo disponible en la figura 2.6(a).

- **ReLU:** Rango de valores entre 0 e infinito. Tiene varias características:
  - Monótona.
  - Buena propagación del gradiente.
  - Eficiente computacionalmente.
  - La más utilizada y recomendada.

Ejemplo disponible en la figura 2.6(b).

- **Función Tangente hiperbólica:** Rango de valores entre -1 y 1. Tiene varias características:
  - Acentúa de mayor manera los valores negativos y los cercanos al 0.
  - Diferenciable.
  - Monótona.

Ejemplo disponible en la figura 2.6(c).



(a) Función Sigmoidal (b) Función ReLU (c) Función Tangente Hiperbólica

Figura 2.6: Funciones de Activación.

Gradiente y pérdida

En las redes neuronales normalmente la función de error está compuesta por dos términos, uno evalúa la salida y el otro que evita el sobreaprendizaje. Como explicamos anteriormente el peso de las conexiones entre las neuronas, denominado w, tiene un valor crucial en la minimización de la **pérdida**. Así pues podemos escribir f(w) para indicar que el valor de error cometido por la red depende de los pesos en esta. Por ello, el objetivo es encontrar un valor w' para el cual obtengamos un mínimo global de la función de error.

El vector **gradiente** se define como las derivadas primera y segunda de la función f. Esto es debido ya que gracias a la derivada de una función en un punto dado podemos obtener la pendiente en dicho punto. Además sabemos que el punto óptimo es aquel en el que el gradiente sea un vector nulo.

Método del Descenso del Gradiente

Es el algoritmo de entrenamiento más simple y utilizado. Consiste en dar pasos en la dirección en que el gradiente desciende más hasta encontrar el punto de convergencia. La distancia en el espacio que nos desplazamos por cada paso es la **tasa de aprendizaje** y es elegida por el usuario. Una tasa con un valor pequeño haría que nos quedásemos estancados en un mínimo local. Y una tasa con un valor muy grande pasará del mínimo y no lo podrá encontrar. Normalmente el valor correcto se encuentra entre 1×10^{-3} y 1×10^{-5}, pero no existe ninguna manera de dar con este más que probar hasta acercarnos al correcto.

2.2.3. Aplicación de las Redes Neuronales a recomendación

Como ya hemos comentado antes, uno de lo usos de las Redes Neuronales más importantes es la recomendación. Para realizar esta tarea se pueden utilizar, entre otras variantes, los tres tipos de redes que hemos comentado previamente: profundas, convolucionales o recurrentes [12]. A continuación vamos a hablar de algunas de las redes más famosas e importantes.

Redes Neuronales en el desafío de Netflix

Fue una competición abierta en la que el creador del mejor algoritmo de filtrado colaborativo, que sería evaluado a partir del RMSE obtenido [13]. El último año que fue realizado el equipo BellKor's Pragmatic Chaos logró un RMSE de 0.8567. La solución propuesta por este equipo fue una mezcla de las Redes Neuronales que había realizado previamente y mejores resultados les había dado. Estaban conformadas por distintas variantes de Redes Profundas y su funcionamiento era el siguiente: k redes aleatorias eran mezcladas en una nueva pequeña red. Esto daba N predicciones que volvían a ser mezcladas ciegamente. Este proceso les permitió mejorar en un 10.06 % el acierto obtenido el año anterior y llevarse el premio [14].

Redes Neuronales en Youtube

El algoritmo de recomendación de Youtube lleva en el foco de mira muchos años y es que debido al crecimiento de la plataforma, los creadores de contenido en la susodicha llevan intentando descubrir cual es la clave para ser los más recomendados. En los últimos años el algoritmo ha cambiado, pero la base siguen siendo las Redes Neuronales. Según [15], la estructura del sistema de recomendaciones estaba compuesta por dos redes neuronales profundas: una para generar los candidatos a recomendar y otra para formar el ránking. La elección de los candidatos se basaba en los gustos del usuario, gracias al filtrado colaborativo la red buscaba usuarios similares a partir de características cómo los IDs de los vídeos vistos, tokens usados en las búsquedas y posición demográfica. Por otro lado la red de ránking otorgaba una puntuación a cada vídeo en función de las características que describían al vídeo y al

usuario.

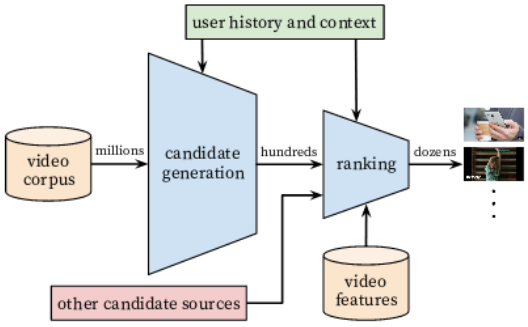


Figura 2.7: Sistema de Recomendaciones de Youtube. Constaba de dos redes profundas: una para la generación de candidatos y otra para la generación del ránking,

2.2.4. Embeddings en las Redes Neuronales

Las Redes Neuronales como hemos comentado anteriormente tienen una gran variedad de usos, uno de estos son los embeddings. Un embedding es un mapeo de una variable discreta en un vector de números continuos. Los embeddings son muy útiles ya que permiten disminuir la dimensionalidad de las variables y representar de una manera significativa estas en un nuevo espacio. El uso de los embeddings tiene tres propósitos [16]:

- Búsqueda de vecinos en el nuevo espacio generado. Puede ser utilizado para realizar recomendaciones utilizando en los intereses de los usuarios o para categorizarlos un clusters.
- Para utilizarlos como entrada en otro modelo.
- Para visualizar conceptos y relaciones entre categorías.

Los embeddings son generados por las Redes Neuronales a partir de los pesos de la red, que como dijimos anteriormente, son ajustados para minimizar la función de pérdida. Los vectores resultantes son representaciones de las categorías, donde las similares tiene valores más próximos entre sí. Esto puede recordar a los *factores latentes* explicados anteriormente en la Factorización de Matrices.

3

Diseño e Implementación

En este Trabajo de Fin de Grado se ha realizado un framework con la finalidad de permitir la construcción de un sistema híbrido formado por una Red Neuronal, que obtiene los embeddings, y vecinos próximos. Gracias al software desarrollado se puede realizar casi automáticamente (en dos pasos) este proceso para, a continuación, proceder a la comparación de este método con el método clásico de vecinos próximos o cualquier otro que se desee o que exista en otra librería de recomendación.

3.1. Requisitos

En esta sección vamos a listar los distintos requisitos funcionales y no funcionales de la aplicación desarrollada.

3.1.1. Requisitos funcionales

RF-1.– El lenguaje debe ser Python 2.9 o 3.0.

RF-2.– El sistema debe contar con un modo automático en el que el usuario pueda obtener en dos pasos el resultado del sistema.

RF-3.– El sistema debe permitir probar más de dos redes neuronales distintas.

RF-4.– El sistema debe ser capaz de realizar recomendaciones dado un ID de usuario.

RF-5.– El sistema debe tener la opción de obtener ítems similares al seleccionado por el usuario.

RF-6.– El sistema debe poder obtener los ítems de mayor relevancia del conjunto de datos.

RF-7.– El sistema debe permitir la utilización de distintas métricas (como el coseno, la distancia euclídea o la distancia manhattanl) a la hora de utilizar vecinos para obtener los elementos similares.

RF-8.– El sistema no necesita tener una interfaz gráfica.

RF-9.– Los resultados finales serán volcados a disco con el fin de poder ser utilizados posteriormente.

RF-10.– El sistema debe permitir aceptar datasets bien conocidos como el de MovieLens 100K [1] y 20M [2] sin realizar modificaciones en estos.

RF-11.– Las Redes Neuronales deben guardar su estado en intervalos de tiempo menor de cinco minutos, con el

[1] files.grouplens.org/datasets/movielens/ml-latest-small.zip

[2] files.grouplens.org/datasets/movielens/ml-20m.zip

fin de prevenir errores en ejecuciones de gran coste (como por ejemplo, veinte mil iteraciones en el dataset de 20M).

RF-12.– El sistema debe en todo momento informar al usuario del estado de la ejecución, así como de los ficheros que falten o de las posibles opciones existentes en caso de poder tomar distintas decisiones (como de qué red extraer los vecinos a partir de los embeddings), con el fin de entender su funcionamiento.

RF-13.– El sistema ayudará al usuario a la hora de hacer las ejecuciones, por ejemplo mostrando las distintas opciones posibles dada la situación en la que se encuentre o recomendando el siguiente paso a dar.

RF-14.– La salida del sistema debe estar integrada con otras librerías de recomendación, como por ejemplo, RankSys [3].

3.1.2. Requisitos no funcionales

RNF-1.– El sistema debe poder ser accedido desde un repositorio Git.

RNF-2.– El sistema debe poder ejecutarse sin limitaciones de memoria RAM, en cualquier ordenador.

RNF-3.– El sistema debe contar con un manual de uso.

3.2. Estructura

3.2.1. Estructura General

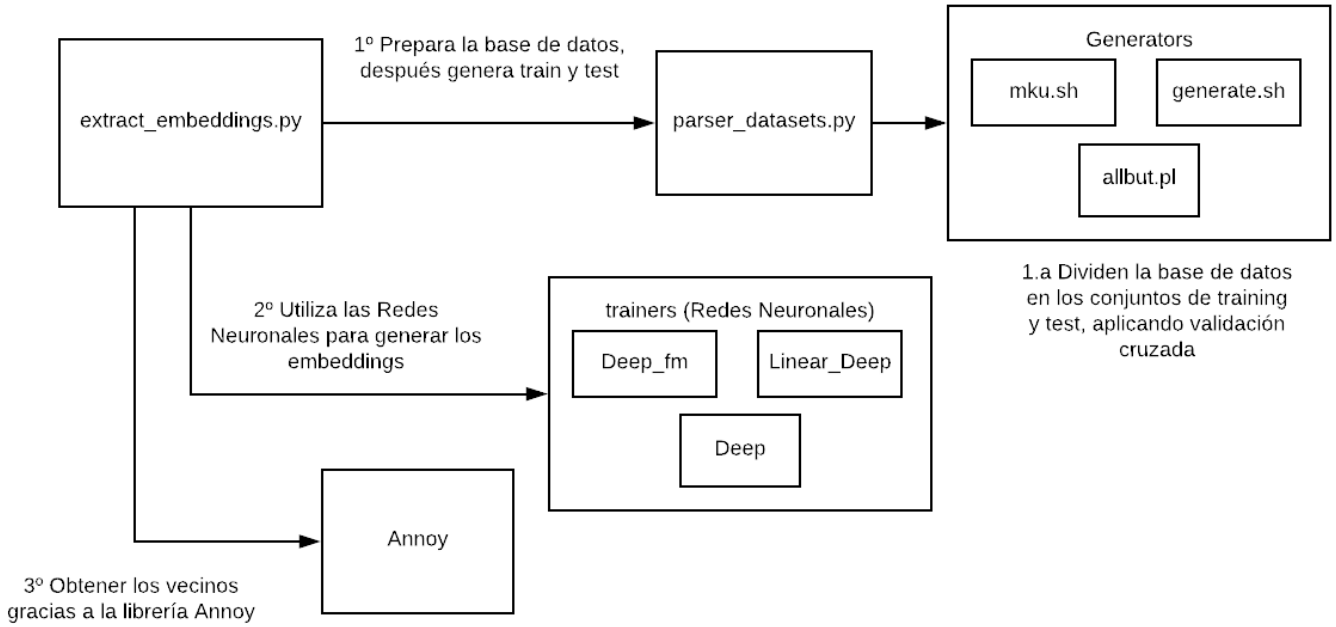


Figura 3.1: Diagrama de alto nivel del sistema.

[3] https://github.com/RankSys/RankSys

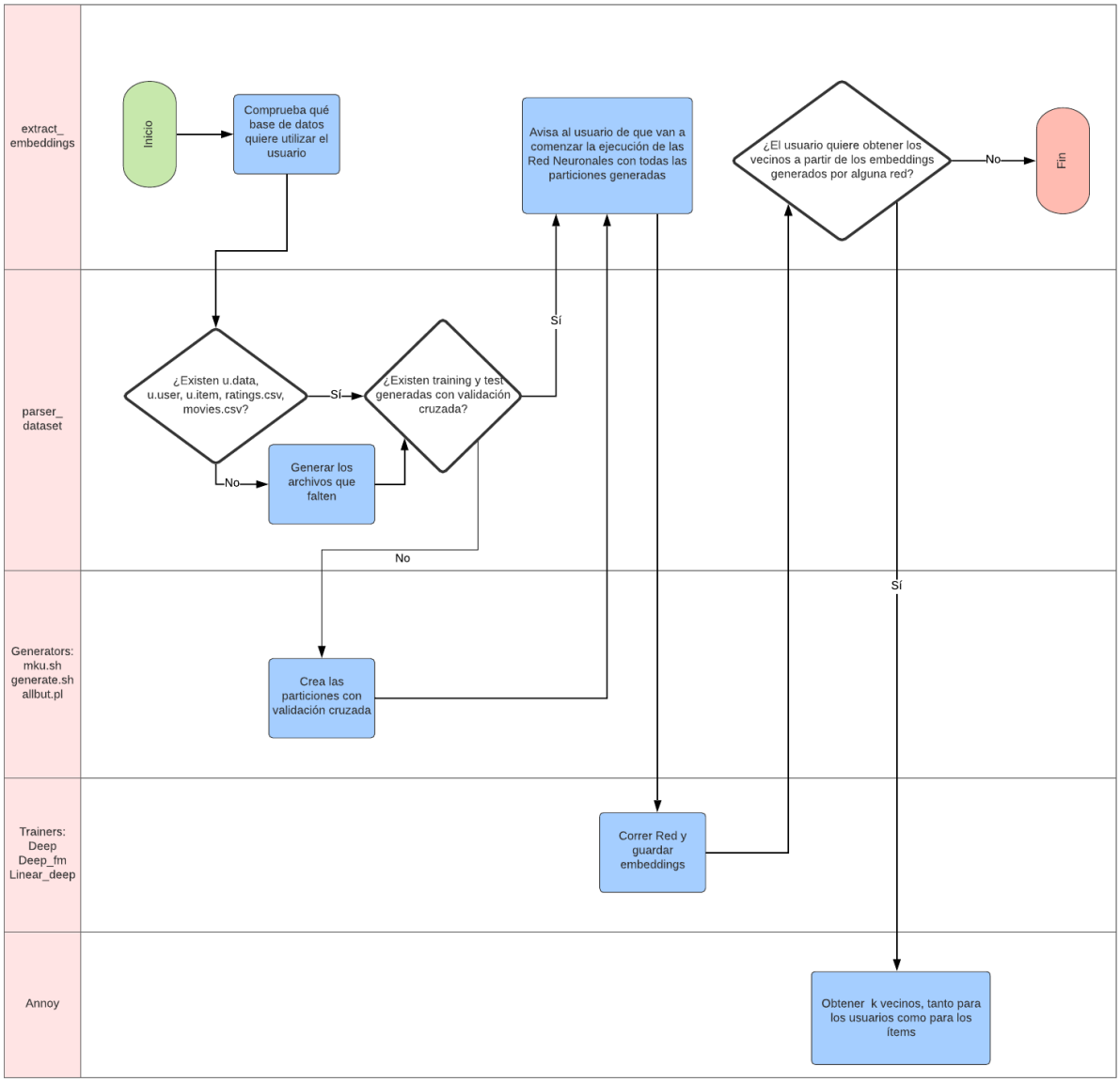


Figura 3.2: Diagrama de secuencia del proceso automático. Podemos observar tal como veíamos en la Figura 3.1 que el módulo extract_embeddings maneja el funcionamiento del proyecto apoyándose en los distintos subsistemas.

El sistema se compone de cinco ficheros escritos en Python; como podemos ver en el diagrama 3.1, el módulo extract_embeddings es el núcleo del framework. A partir de este se pueden realizar todas las acciones posibles para generar el sistema híbrido. Estas son en esencia: extraer los vecinos de los elementos y realizar recomendaciones. Para poder llevarlas a cabo se ha dividido extract_embeddings en tres partes con el fin de modularizar el sistema para permitir realizar modificaciones y ampliarlo de la manera más sencilla posible:

1.– **Subsistema de estandarización de los datos:** Este subsistema comprueba qué archivos hacen falta y si los genera si fuera necesario. Es imperativo para el correcto funcionamiento del resto de subsistemas que existan: el fichero con las valoraciones de los usuarios, el fichero con los datos de los ítems y el fichero con los datos de los usuarios.
   Tras esto, este subsistema crea las particiones. Para hacer este paso, aprovechamos los scripts que MovieLens trae consigo en la versión 100K (la que tiene cien mil valoraciones de usuario). Estos nos sirven para generar cinco pares de ficheros de entrenamiento y test con validación cruzada.

2.– **Subsistema de generación de embeddings:** Este subsistema incluye las distintas Redes Neuronales con las que hemos trabajado. Tras la comprobación de que los datos han sido generados correctamente o que ya existían, son utilizados para correr cada tipo de red con cada par entrenamiento-test creado anteriormente. Las redes guardan automáticamente los valores de los embeddings en disco para que puedan ser utilizados posteriormente.

3.– **Subsistema de extracción de vecinos:** El último subsistema genera a partir de los embeddings los vecinos de los usuarios y los ítems. Permite obtener elementos similares, hacer recomendaciones u obtener los ítems más utilizados.

Estos subsistemas pueden ser ejecutados de manera independiente, con el fin de permitirnos no repetir acciones innecesarias si ya hemos acabado una parte, como volver a generar los embeddings o extraer de una red los vecinos si ya los tenemos.

Ejemplo de secuencia del proceso automático

En el diagrama 3.2 podemos ver el proceso por el cual el sistema extrae automáticamente los vecinos de los elementos a partir de los embeddings generados por las redes. Podemos ver que la única interacción que debe realizar el usuario es tras el trabajo de las redes, eligiendo a partir de cuál de ellas quiere generar los vecinos.

3.2.2. Subsistema de generación de embeddings - Redes Neuronales

Como hemos visto anteriormente hay distintos tipos de Redes Neuronales orientadas a la recomendación. Con el fin de utilizar un modelo sencillo pero eficaz, se han decido utilizar las redes expuestas en el artículo [17]. En dicho artículo, los autores proponen unas redes profundas con una estructura sencilla pero a su vez con unos resultados aceptables, lo que nos ayudará a la hora de manejarlas y entenderlas.

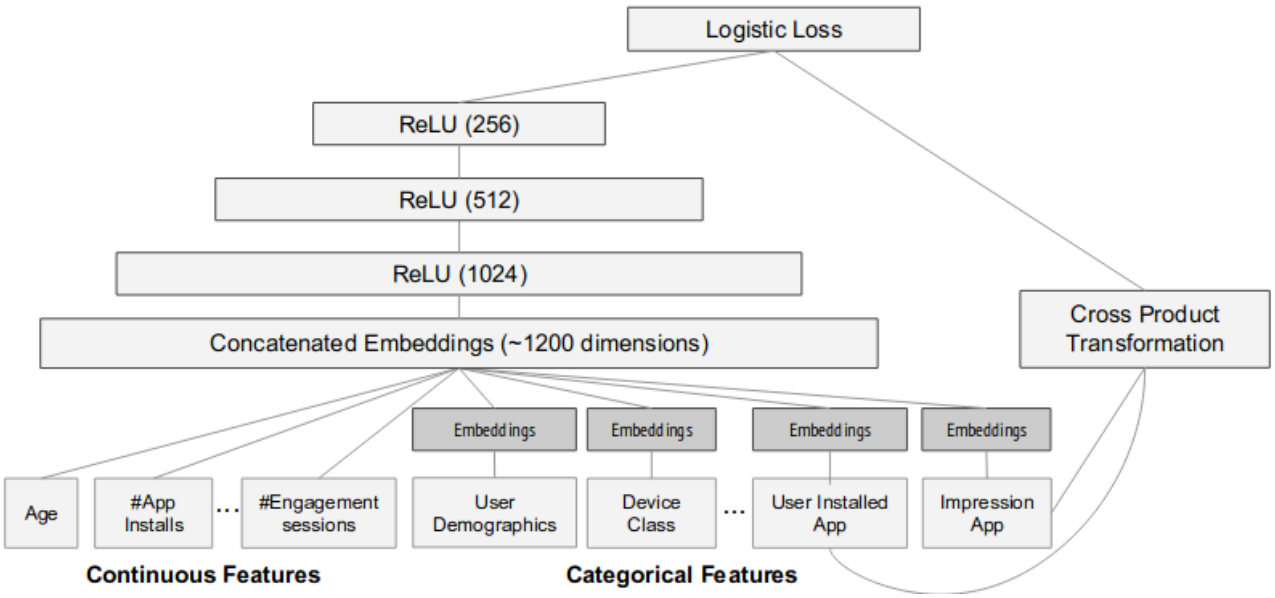


Figura 3.3: Estructura de la Red Profunda deep_fm propuesta en [17].

3.2.3. Subsistema de extracción de vecinos - Annoy

Para la extracción de los vecinos hemos utilizado la librería Annoy, creada por Erik Bernhardsson. Esta librería tiene varias características que nos han llevado a elegirla: es una de las librerías con un cálculo de vecinos más rápidos y crea una estructura de datos en disco muy eficiente, basada en índices a partir de los embeddings, que permite reutilizar los valores precalculados en distintos procesos, lo que se adapta muy bien a nuestros requisitos al igual que su bajo uso de memoria. Por último, destacar que Annoy ha sido desarrollada en C++, pero puede ser utilizada desde Python de manera que es fácil de acoplar al resto del sistema.

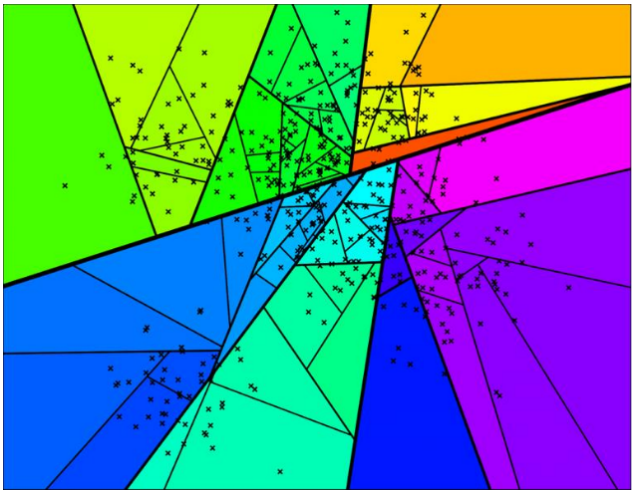
A continuación explicaremos con mayor detalle cómo Annoy obtiene los vecinos.

3.2.3.1. Vecinos próximos aproximados

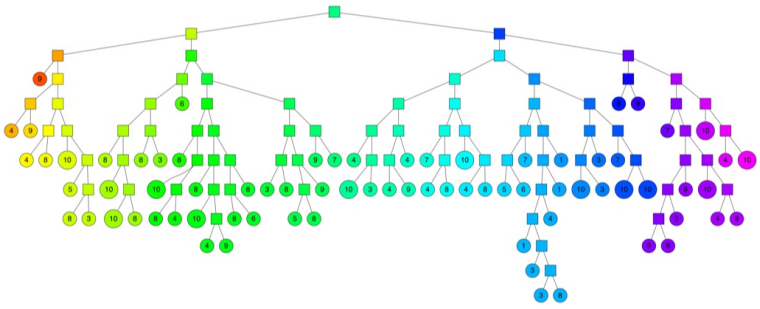
El método que utiliza Annoy funciona de la siguiente manera:

Construcción del Índice

Partiendo del conjunto de datos original elegimos dos puntos al azar, dividiendo el conjunto en dos partes. Para cada una de estas particiones repetimos la división hasta que haya K elementos en cada subconjunto [figura 3.4(a)], esta información se almacena en un árbol [figura 3.4(b)]. La primera ventaja que obtenemos en este paso es el gasto de memoria, siendo n/K en vez de n.



(a) Representación de los datos tras las divisiones



(b) Árbol binario contenedor del índice

Figura 3.4: Construcción del Índice [18].

Búsqueda en el Índice

Si queremos buscar los vecinos de cierto elemento, buscamos el mismo en el árbol. Siendo sus vecinos las hojas que se encuentren colgando de su misma rama. Al haberse realizado las particiones aleatoriamente, tiene varios inconvenientes:

- El elemento que realmente es más cercano no tiene que estar obligatoriamente en la misma rama del árbol binario.
- Dos puntos cercanos pueden estar en distintas partes de una división.

La solución es ir a ambos lados de una rama si está cerca. Esto se calcula navegando por el árbol utilizando una cola de prioridad, la cual está ordenada por la distancia de la hoja a la raíz, siendo prioritaria la hoja de menor distancia.

Figura 3.5: Ejemplo de búsqueda de vecinos [18].

Pero esto no es suficiente, ya que siguen escapándose vecinos relativamente cercanos y puede que devolvamos elementos que realmente no sean tan semejantes. Para ello, utilizamos otro truco más: construir en vez de un único árbol varios y utilizar la misma cola de prioridad para buscar en todos a la vez, obteniendo la unión de los árboles que contienen los elementos más cercanos.

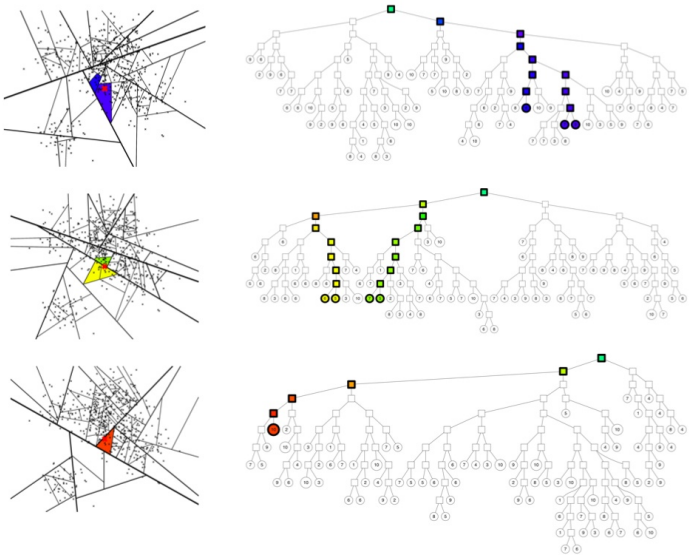


Figura 3.6: El uso de múltiples árboles nos permite obtener distintos puntos como candidatos a vecinos cercanos [18].

Finalmente eliminamos los elementos repetidos devueltos por los distintos árboles y calculamos las

distancias de los restantes, para devolver el ránking de los más cercanos.

Figura 3.7: El resultado de la unión de los elementos devueltos por los árboles corresponde al área dentro de la línea negra. Tras esto calculamos las distancias quedándonos con los más cercanos, los contenidos dentro del círculo rojo [18].

3.3. Solución Implementada

En este apartado vamos a explicar las decisiones tomadas para poder implementar el software descrito en el apartado anterior.

El framework, como se ha descrito anteriormente en el apartado 3.1, consta de tres partes, las cuales son utilizadas por un módulo central, con el fin de proporcionar una interacción con el usuario más sencilla. El sistema permite: preparar los conjuntos de datos, correr las Redes Neuronales, realizar recomendaciones, obtener los ítems más populares ordenados y extraer los vecinos de los elementos. A continuación en la memoria explicaremos con mayor profundidad cómo se ha desarrollado el framework para que permita estas acciones, pero antes de comenzar empezaremos hablando de la base del proyecto, la elección de las Redes Neuronales.

3.3.1. Redes Neuronales implementadas

Con el fin de estudiar y aprender cómo se utilizan las Redes Neuronales para recomendación, comenzamos decidiendo qué tipo de red sería mejor utilizar. Como hemos explicado anteriormente en el apartado 3.2.2, decidimos que una Red Profunda sería la más recomendable. Estas redes son muy comunes y una de las más sencillas de manejar, ayudándonos en el proceso de aprendizaje.

Así pues, para asegurarnos de partir de una base consistente buscamos una red ya implementada que nos permita realizar modificaciones en esta de una manera sencilla. Por todo esto elegimos el modelo proporcionado en un repositorio público [4]. Partir de este ejemplo nos ha permitido tener una red profunda con la que realizar pruebas, *deep_fm*, y variantes más sencillas de la misma para poder comprobar si los resultados mejoran considerablemente gracias a la red que el artículo nos presenta o si no es necesario crear una red tan compleja. Las otras dos razones por las que hemos decidido optar por este proyecto inicial son que está implementado orientado a utilizar los datos de MovieLens del conjunto de cien mil ejemplos y que está desarrollado en TensorFlow.

3.3.1.1. TensorFlow

TensorFlow es una librería de código abierto desarrollada por Google para inteligencia artificial. Desde su aparición se ha mantenido en continuo desarrollo, con más de 2000 contribuidores actualmente y casi 60000 commits.

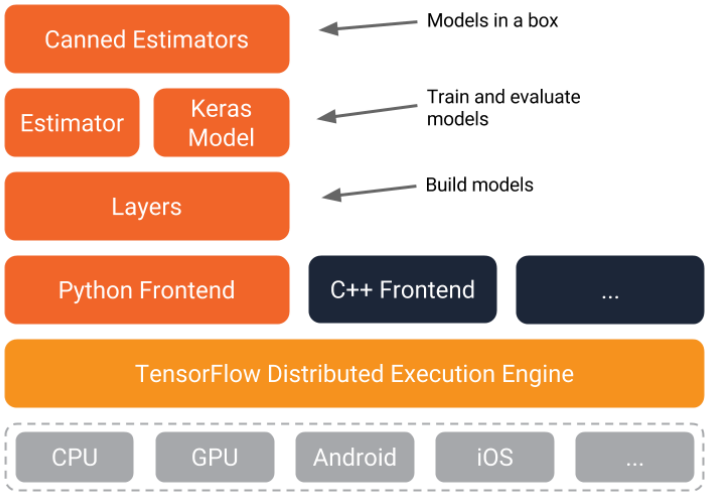


Figura 3.8: Estructura de TensorFlow.

Como hemos visto anteriormente, las redes neuronales se basan en pequeñas operaciones matemáticas, que en conjunto acaban construyendo un sistema que imita a las neuronas de nuestro cerebro. En TensorFlow estas conforman su núcleo, el cual ha sido escrito en C++ para un mayor rendimiento. Sobre este se encuentran dos modelos posibles: los estimadores y Keras. El uso de uno u otro varía completamente la forma de crear una red gracias a TensorFlow, mientras que si utilizamos Keras debemos crear la red teniendo en cuenta qué función de activación utilizamos y el resto de detalles de la estructura de la red, tal como los explicados en 2.2.2, en cambio si utilizamos los estimadores no debemos tener esto en cuenta. Los estimadores ya contienen modelos creados que pueden ser utilizados tanto en local como en un sistema distribuido y que pueden ser lanzados en CPU, GPU o

[4] https://github.com/yxtay/recommender-tensorflow

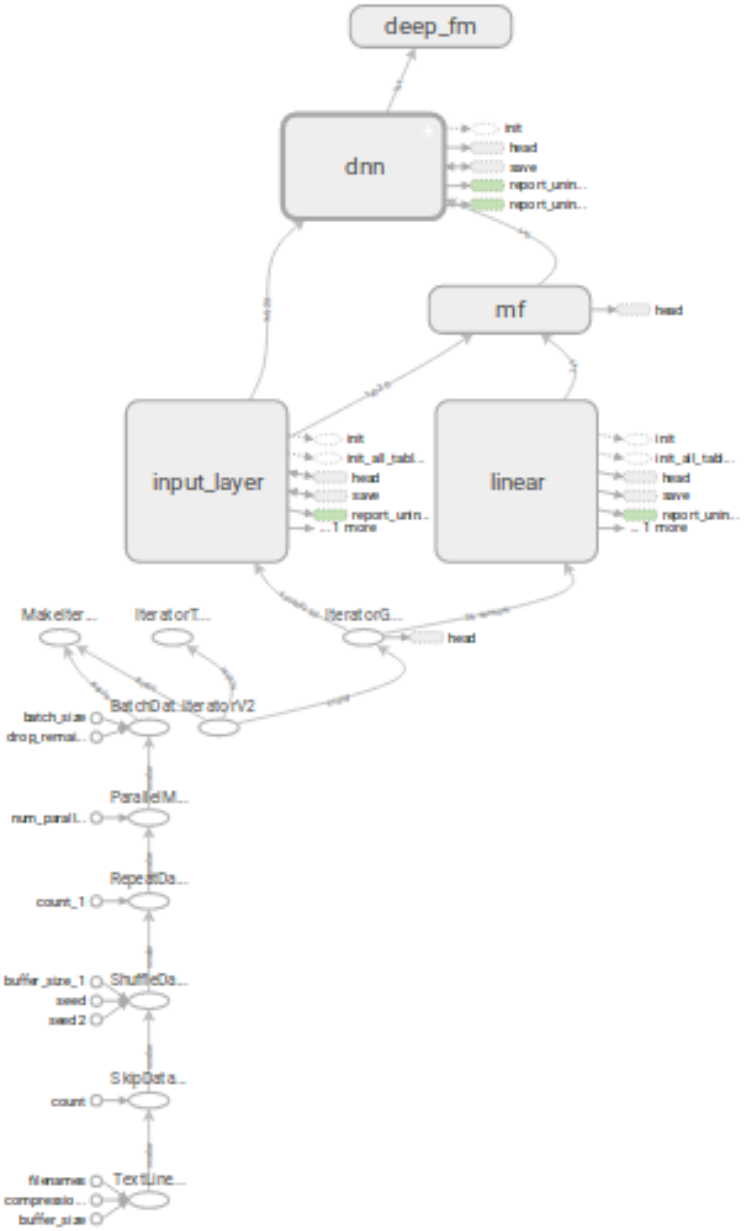


Figura 3.9: Grafo de deep_fm generado por TensorBoard. Los rectángulos corresponden a subredes existentes, puede verse también su estructura en el visualizador.

TPU (Unidad de Procesamiento de Tensor, circuitos integrados aceleradores de Inteligencia Artificial desarrollados por Google específicamente para el aprendizaje automático, principalmente utilizados en Google Cloud [5]). Estos construyen el grafo que conforma la red por si mismos, inicializan las variables, cargan los datos, manejan las excepciones, crean puntos de control por si ocurre algún error y guardan estadísticas para poder revisar el funcionamiento de la red. En el proyecto realizado vamos a aprovechar esta característica, la red neuronal ha sido creada utilizando estimadores que se han modificado para obtener la estructura deseada y que sea más fácil de utilizar. Así también, las variantes de la red que hemos nombrado previamente serán realizadas a partir de estimadores creados por el mismo Google. Siendo así deep.py un modelo basado en Redes Neuronales Densas (donde todas las neuronas de una capa están conectadas a todas las de la siguiente) y prealimentadas (las conexiones entre las neuronas no forman un bucle). Y la red linear_deep.py un modelo mixto de una red lineal y una densa.

Otra de las características de TensorFlow que nos han ayudado a decantarnos por esta opción han sido las ayudas que la librería nos ofrece, entre estas destaca TensorBoard:

TensorBoard

Las operaciones realizadas en TensorFlow, por ejemplo para entrenar la red, pueden ser complejas y confusas. Con el fin de hacer más fácil de entender, debuggear y optimizar los programas en TensorFlow existe la herramienta TensorBoard. Esta puede ser utilizada para visualizar el grafo (un ejemplo de grafo generado sería la figura 3.9), las métricas de evaluación y el comportamiento de las funciones de activación. También dispone de un proyector con el que podemos ver una representación en el espacio de cómo la red clasifica y agrupa los datos. Además al seleccionar un punto de la nube, a partir de los embeddings obtenidos por la red, se nos muestra sus vecinos próximos, pudiendo alternar entre utilizar la distancia euclídea o la distancia coseno.

3.3.2. Implementación de subsistemas

3.3.2.1. Subsistema de estandarización de los datos

Para entender por qué es necesario este módulo y su funcionamiento debemos comenzar por explicar qué datos hemos utilizado y por qué.

MovieLens

MovieLens es un dataset que dispone de una colección de datos de valoraciones de los usuarios obtenidos a partir de su sitio web. Hemos elegido este dataset dado que es muy común su uso en

[5] https://cloud.google.com/tpu/?hl=es

investigación, puesto que es libre y los datos han sido recogidos a partir de una fuente fiable. Dependiendo del tamaño del dataset de MovieLens la estructura de los ficheros es distinta, por lo que necesitamos el módulo de estandarización de los datos para transformarlos para que puedan ser utilizados por las Redes Neuronales. Podemos ver la diferencia entre el número de rátings, usuarios e ítems en la figura 3.1.

	Ratings	Usuarios	Ítems
Movielens 100K	100000	943	1682
Movielens 20M	20000263	138493	27278

Tabla 3.1: Número de ratings, usuarios e ítems de MovieLens en sus variantes de 100K y 20M.

Tras esta breve introducción a MovieLens, explicaremos el trabajo que realiza este subsistema. Este puede ser dividido en dos partes:

La primera consiste en la estandarización de los datos. Esto es, la generación de los archivos en formato .csv o en texto plano con la información necesaria. Dependiendo de la estructura del dataset original se aprovechan unos archivos u otros, estos son:

El dataset de 100K, al ser tomado como el original por el dataset no requiere casi modificaciones. Inicialmente contiene ficheros en texto plano así que únicamente debemos crear ratings.csv y movies.csv. Por un lado ratings.csv se genera a a partir de u.data cambiando el formato a .csv, pero por el otro movies.csv se genera a partir de u.item, este fichero a parte del cambio de formato contiene el listado de géneros con un 1 si la película tiene ese género y 0 en caso contrario.

El dataset de 20M, no contiene información adicional sobre los usuarios y las películas por lo que es necesario encontrar una manera de estandarizarlo para que tenga algún valor y que además la red no lo tenga en cuenta. Esta solución es sencilla, ya que si le damos a todos los mismos valores esos factores no influirán en el resultado final. En cuanto a los archivos, debemos generar: u.data, que es el inverso a ratings.csv como hemos dicho antes, así que únicamente requiere el cambio de formato; u.user, este fichero es necesario ya que contiene los identificadores de los usuarios, pero al no existir el resto de sus campos (edad, género, trabajo y código zip) debemos darle un valor por defecto, siendo la edad igual a 0, el código postal 00000 y el género y el trabajo estas dos mismas palabras; por último u.data es generado, de movies.csv es obtenido el identificador de la película y el título, el resto de valores se dejan en blanco al no hacer falta.

La segunda parte consiste en generar cinco ficheros de entrenamiento y test con validación cruzada. El primer paso consisten en obtener un único fichero con toda la información necesaria a partir de todos los generados. Si partimos del dataset de 100K este proceso ya viene hecho en el proyecto inicial, lo que hacen es cargar la información en dataframes (que son matrices que pueden contener distintos tipos de datos) y después agrupar a las valoraciones la información de los usuarios y as pelí-

culas. Esto lo hacen fácilmente gracias a la librería dask [6], que tiene funciones especiales para cargar los datos y agruparlos por lo índices. Pero al intentar imitar este proceso en el dataset de 20M no podemos hacerlo igual ya que al contener tantos datos consumimos demasiada memoria provocando que sea casi imposible ejecutarlo, violando uno de los requisitos funcionales. La solución tomada ha sido crear una base de datos SQL y a partir de consultas mezclar los datos, ejemplo disponible en apartado 3.3.2.1.

Código 3.1: Ejemplo de consulta utilizada para generar los datos de entrenamiento en el dataset MovieLens 20M.

```
CREATE TABLE train_table AS
    SELECT
        user_id, item_id, rating, timestamp,
        strftime('%Y-%m-%d␣%H:%M:%S',
            datetime(timestamp, 'unixepoch')) as datetime,
        strftime('%Y', datetime(timestamp, 'unixepoch')) as year,
        strftime('%m', datetime(timestamp, 'unixepoch')) as month,
        strftime('%d', datetime(timestamp, 'unixepoch')) as day,
        strftime('%W', datetime(timestamp, 'unixepoch')) as week,
        strftime('%w', datetime(timestamp, 'unixepoch')) as dayofweek,
        age, gender, occupation, zipcode,
        SUBSTR(zipcode, 0, 1) AS zipcode1,
        SUBSTR(zipcode, 0, 2) AS zipcode2,
        SUBSTR(zipcode, 0, 3) AS zipcode3,
        title, release, video_release, imdb,
        unknown, action, adventure, animation, children, comedy,
        crime, documentary, drama, fantasy, filmnoir, horror,
        musical, mystery, romance, scifi, thriller, war, western,
        release as release_date, strftime('%Y',
            datetime(release, 'unixepoch')) as release_year
    FROM TRAIN
    JOIN USERS USING (user_id)
    JOIN ITEMS USING (item_id);
```

Por último comentar que, al tener el dataset de 100K como original, este contiene originalmente dos scripts para generar las particiones: mku.sh y allbut.pl. Hemos decidido aprovecharlos e integrarlos al sistema, esto es por lo que en el dataset de 20M debemos crear tantos ficheros, para que sea similar al dataset de 100K. Además, el fichero generate.sh que hemos nombrado anteriormente en la estructura del fichero contiene la parte de generación de las particiones de mku.sh, que ha sido particionado ya que era necesario que se realizase en dos pasos.

3.3.2.2. Subsistema de generación de embeddings

Para poder obtener los embeddings, tal como hemos explicado anteriormente, hemos utilizado las redes explicadas en el apartado 3.3.1. Al ser esta la base del framework e implementar el resto

[6] https://dask.org/

de módulos de manera que sean compatibles con este, no ha sido necesario realizar demasiadas modificaciones. En un primer lugar añadimos a cada una de las redes (deep_fm, deep y linear_deep) una función que nos permitiese correr las redes con distintos parámetros, esta es run. Gracias a esta podemos seleccionar cualquier dataset, cambiar el número de iteraciones e incluso nos permite realizar el modo automático, al ser posible decirle que corra la red n veces cada una con un dataset.

Otro aspecto que resaltamos en el apartado 3.3.1 es que íbamos a utilizar estimadores, los cuales nos ayudarían a la hora de crear cada red. Estos necesitan saber cómo se les van a introducir los datos: formato de estos y cuantos. Como nuestros datos han sido estandarizados previamente imitando la estructura del dataset de movielens 100K, únicamente debemos tener en cuenta que si cambiamos de dataset debemos avisar a la red sobre el nuevo número de usuarios o de ítems. Hemos optado, ya que en un principio se utilizan únicamente dos datasets, por que la red detecte cuál de los dos se utiliza y a partir de esto seleccione el tamaño. Estuvimos pensando otra alternativa: al estandarizar los datos, guardar en un nuevo fichero estas características con el fin de evitar sobreespecialización a los conjuntos de datos. Pero esto implicaría almacenar el número de usuarios, el número de ítems, el rango de edad de los usuarios, el número de trabajos distintos, el número de diferentes códigos postales y el rango de años de salida de las películas; pensamos que aunque sería fácil, podría aumentar los tiempos de ejecución considerablemente y finalmente no lo llevamos a cabo.

3.3.2.3. Subsistema de extracción de vecinos

Para la extracción de vecinos hemos utilizado la librería Annoy (su funcionamiento interno y lógica ha sido explicado previamente en el apartado 3.2.3.1). Esta se ha integrado en el sistema simplificando su manejo gracias al uso de cuatro funciones implementadas en extract_embeddings.py:

- **build_embeddings_index:** Función que crea el índice de Annoy, permite seleccionar el número de árboles que conforman el bosque. Además nos permite generar los vecinos utilizando las siguientes métricas: distancia coseno, euclídea, manhattan, hamming y producto. Recordar que estas métricas son importantes ya que son utilizadas en la cola de prioridad.
- **get_similar_movies:** Dado un identificador de película, nos permite obtener tantos vecinos de esta como especifiquemos.
- **recommend_new_movies:** Dado un usuario devuelve recomendaciones de películas, tantas como queramos.
- **get_index:** Utilizando una de las métricas listadas antes escribe en dos ficheros los cien primeros vecinos de todos los usuarios y los ítems respectivamente, así como la similitud entre ambos.

4

Pruebas y Resultados

En este apartado probaremos el funcionamiento del framework desarrollado. Para ello aplicaremos las siguientes pruebas:

- Con el fin de analizar las diferencias entre las distintas Redes Neuronales, comprobaremos si los resultados obtenidos entre estas tienen alguna similitud en términos de los vecinos obtenidos con cada una. Como dijimos previamente en el apartado 3.3.1.1 todas son redes profundas, aunque cada una es una variante distinta. Para llevar esto a cabo aplicaremos Jaccard, ya que mide el grado de similitud entre dos conjuntos.
- Comprobaremos cuán buenas son las distintas variantes implementadas comparando respecto a los vecinos de usuarios e ítems obtenidos contra los mismos obtenidos gracias a la librería RankSys.
- Finalmente realizaremos otra comparación contra otros sistemas de recomendación también implementados en la librería RankSys.

El framework como hemos comentado previamente ha sido desarrollado utilizando los datasets MovieLens 100K y MovieLens 20M, pero a la hora de realizar las pruebas mostradas en este apartado hemos decidido utilizar únicamente la variante de MovieLens de 20M, ya que los resultados que podemos obtener con este son más precisos y fiables. Además, antes de comenzar, introduzcamos brevemente RankSys para entender por qué utilizamos esta librería.

RankSys

Es un framework para la implementación y evaluación de algoritmos de recomendación que surgió como resultado de un trabajo de investigación y que actualmente es utilizado en distintas publicaciones y tesis [1]. El framework ha sido desarrollado en Java 8, el cual es la última versión del lenguaje y aprovecha muchas de las últimas características y ventajas que aporta, tal como las funciones lambda, streams y la posibilidad de paralelizar el código. Gracias a esto y a su manejo de grandes datasets, como los que nosotros tenemos que utilizar, lo convierten en la solución perfecta para realizar las pruebas necesarias en este proyecto.

Gracias a la gran variedad de algoritmos y métricas de evaluación que dispone RankSys, nos ha permitido realizar todas las comprobaciones deseadas de una manera sencilla y eficaz. De entre los distintos algoritmos hemos elegido:

[1] https://github.com/RankSys/RankSys/wiki/References

- Vecinos próximos orientado a usuarios.
- Vecinos próximos orientado a ítems.
- Recomendador aleatorio.
- Recomendador basado en la popularidad.
- Factorización de Matrices.

Y de las distintas métricas de RankSys, las cuales han sido previamente explicadas en el apartado 2.1.3, probaremos nuestro sistema con las siguientes:

Métricas individuales

- Precisión.
- Recall.
- nDCG.
- EPC (Expected Popularity Complement).

Métricas del sistema

- AD (Aggregate Diversity).
- Gini.

4.1. Resultados

4.1.1. Diferencias entre las Redes Neuronales implementadas

	@5	@10	@15	@20	@50
Deep vs Deep_fm	1.661E-5	2.975E-5	4.641E-5	6.261E-5	1.545E-4
Deep_fm vs Linear_deep	1.704E-5	3.062E-5	5.334E-5	6.17E-5	1.527E-4
Deep vs Linear_deep	1.718E-5	3.293E-5	5.368E-5	7.134E-5	1.775E-4

Tabla 4.1: Jaccard entre usuarios de las distintas redes.

	@5	@10	@15	@20	@50
Deep vs Deep_fm	1.188E-4	2.290E-4	3.645E-4	4.257E-4	1.024E-3
Deep_fm vs Linear_deep	1.015E-4	1.588E-4	2.615E-4	3.785E-4	9.191E-4
Deep vs Linear_deep	1.102E-4	1.739E-4	2.636E-4	3.592E-4	9.150E-4

Tabla 4.2: Jaccard entre ítems de las distintas redes.

Como podemos ver en los resultados obtenidos en las tablas 4.1 y 4.2, las distintas redes devuelven vecinos fácilmente diferenciables, ya que los valores de Jaccard obtenidos (que mide la intersección entre dos conjuntos normalizada por la unión) son muy bajos. Esto nos permite comprobar como

aunque tenemos tres Redes Neuronales del mismo tipo (son Redes Profundas) y que dos de ellas son variantes de una original con ligeras diferencias entre sí (Deep y Linear_deep son las dos redes densas), los vecinos obtenidos se asemejan mínimamente.

También es cierto que al haberse creado las Redes Neuronales a partir de los estimadores de TensorFlow y estos abstraernos de los pequeños detalles de la red, no podemos controlar la estructura exacta del grafo. Y es que los estimadores tienen la capacidad de añadir neuronas y ajustar parámetros (como la Función de Activación a utilizar o el valor de la Tasa de Aprendizaje), haciendo que dos redes teóricamente muy similares puedan devolvernos resultados completamente distintos. Todo esto es fácil de comprobar ya que como vimos en el grafo generado en TensorBoard en la figura 3.9, este dista de ser semejante a la figura 3.3, el cual sería el ideal. Esta abstracción puede ser un inconveniente en ciertas ocasiones, ya que nos complica el control del grafo otorgándonos a cambio una red mucho más segura y fácil de implementar.

4.1.2. Comparación de usuarios e ítems

	Precisión	Recall	nDCG	EPC	AD	Gini
Modelo Híbrido, Deep	0.025	0.041	0.035	0.829	20693.190	0.188
Modelo Híbrido, Deep_fm	0.025	0.042	0.042	0.829	20259.590	0.194
Modelo Híbrido, Linear_deep	0.025	0.042	0.034	0.829	20259.590	0.191
Ranksys	0.019	0.032	0.026	0.889	8694.600	0.078

Tabla 4.3: Comparación vecinos orientados a usuarios de Ranksys vs el modelo híbrido.

	Precisión	Recall	nDCG	EPC	AD	Gini
Modelo Híbrido, Deep	5.832E-4	9.198E-4	7.466E-4	0.988	8460,60	0,117
Modelo Híbrido, Deep_fm	5.083E-4	8.431E-4	6.299E-4	0,977	8366,80	0,103
Modelo Híbrido, Linear_deep	5.468E-4	7.967E-4	6.073E-4	0,978	8507,20	0,122
Ranksys	0.023	0,039	0,032	0.849	20259,59	0,226

Tabla 4.4: Comparación vecinos orientados a ítems de Ranksys vs las distintas variantes del modelo híbrido.

En las tablas 4.3 y 4.4 podemos observar la comparación de dos sistemas de recomendación, orientados respectivamente a usuarios y a ítems. La finalidad de ambas es comprobar si el modelo híbrido (el cual es el resultado de este trabajo de fin de grado) en sus distintas variantes mejora al modelo clásico de vecinos próximos. Cada uno de los tres modelos híbridos se diferencian en la manera de obtener los embeddings. Siendo: *Modelo Híbrido, Deep*, el sistema en el cual la Red Neuronal Deep es la que obtiene los embeddings que serán utilizados por vecinos próximos; *Modelo Híbrido, Deep_fm*, en el cual Deep_fm obtiene los embeddings y *Modelo Híbrido, Linear_deep*, en el que los embeddings se obtienen de la red Linear_Deep.

Podemos ver que el sistema realizado es bueno en todas sus variantes si es orientado a usuarios, pero muy malo orientado a ítems. Es posible que esto se deba a que, como dijimos en el apartado 3.3.2.1, al no tener en el dataset de MovieLens de 20M tanta información sobre las películas como en el de 100K nuestro sistema tenga problemas a la hora de diferenciarlas. También es probable que, por cómo está definida la red, los embeddings de ítems no estén tan bien generados o definidos como los de usuarios, es decir, estas redes están más orientadas a aprender embeddings de usuarios que de ítems.

Otro aspecto a comentar es el comportamiento de RankSys, el cual es mejor en su variante orientada a los ítems (al contrario del modelo creado en este Trabajo de Fin de Grado). Esta variante además es la más equilibrada de todas las observadas junto a los modelos híbridos creados orientados a usuarios. Comparando más en profundidad estos sistemas de recomendación, ya que han sido los mejores, podemos ver que destaca el valor de nDCG de la variante híbrida que obtiene los embeddings a partir de la red Deep_fm. Por otro lado, RankSys orientado a ítems nos aporta la recomendación más equilibrada y diversa, al tener un valor de Gini mayor que el resto.

Es interesante observar que las distintas variantes del sistema híbrido creado, pese a que como vimos en el apartado anterior devuelven resultados muy distintos entre sí, tienen un rendimiento muy parecido en todas las métricas. Además podemos comprobar que el modelo original del que hemos partido propuesto en [17] no nos aporta grandes ventajas pese a ser el más complejo de los tres.

4.1.3. Comparación con otros sistemas de recomendación

	Precisión	Recall	nDCG	EPC	AD	Gini
Popularity	0.027	0.045	0.037	0.818	8596.69	0.048
Random	2.563E-4	4.329E-4	3.474E-4	0.993	19752.01	0.844
Factorización de Matrices	0.025	0.042	0.036	0.840	1913.99	0.020

Tabla 4.5: Resultados obtenidos con otros recomendadores de Ranksys.

En la tabla 4.5 podemos ver tres nuevos sistemas de recomendación. El de popularidad y el aleatorio los hemos elegido puesto que son muy sencillos y comunes, por otro lado, hemos considerado interesante comprobar también el método de Factorización de Matrices ya que es similar al método utilizado por las Redes Neuronales para extraer los embeddings de los elementos.

Comparando la tabla 4.5 con 4.3 y 4.4, con el fin de comprobar si nuestro sistema funciona bien contra otros recomendadores distintos, podemos ver que el recomendador por popularidad pese a ser uno de los más sencillos es a la vez de los que mejor funciona. Obteniendo un acierto mejor que todos los vistos previamente (en términos de Precisión, algo que no ocurre en términos de nDCG), en contraposición se puede comprobar que su valor Gini es demasiado bajo. Esto significa que, tal como el sentido común indica, al recomendar únicamente los ítems más populares la probabilidad de que al

usuario le guste dicho ítem es alta, pero al centrarse únicamente en ciertos ítems se queda estancado sin lograr realizar nuevas recomendaciones que puedan sorprender y agradar al usuario.

También es relevante observar cómo al realizar las recomendaciones de manera aleatoria el acierto es muy bajo, esto se debe a que en el dataset de MovieLens de 20M hay 27278 películas y es casi imposible acertar eligiendo al azar. Por otro lado, el algoritmo de Factorización de Matrices tiene un acierto similar a los híbridos basados en usuarios propuestos en este trabajo (ver Tabla 4.3), pero aún así por debajo del recomendador basado en popularidad.

5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

Con la aparición de las nuevas tecnologías y su desarrollo se ha visto impulsado el crecimiento de internet. Esto ha incitado la búsqueda de nuevos sistemas de recomendación que mejoren la experiencia de los usuarios y permitan a las compañías obtener mayores beneficios. El estudio e investigación de nuevas técnicas se encuentra en pleno auge, con miles de profesionales por todo el mundo intentando encontrar el mejor sistema de recomendación.

El razonamiento anterior ha sido la motivación principal de este Trabajo de Fin de Grado, el cual ha consistido de un trabajo de investigación en el que hemos visto algunos de los sistemas de recomendación de mayor relevancia hoy en día, así como las distintas variantes de estos y métricas de evaluación, y hemos intentado dar con una solución que pueda mejorar a los sistemas clásicos.

Tras una ardua investigación tomamos la decisión de implementar un sistema de recomendación híbrido. Decidimos esto dado que, como hemos visto en el período de estudio, los recomendadores basados en vecinos próximos nos aseguran obtener grandes resultados con un bajo coste de recursos. Además, descubrimos que los mejores sistemas híbridos que utilizan vecinos próximos se encuentran estructurados en dos partes: una primera parte compuesta por una Red Neuronal que obtiene los embeddings (que son caracterizaciones de los usuarios e ítems) de los datos y una segunda parte, el recomendador de vecinos, que a partir de estos embeddings realiza las recomendaciones necesarias. Además, con el fin de utilizar un sistema de recomendación de vecinos que nos permita tratar con conjuntos de datos de gran tamaño asegurándonos buenos resultados, decidimos utilizar Annoy. Esta es la librería utilizada por Spotify para sus recomendaciones, por lo que la hacía perfecta para nuestro sistema.

Tras las pruebas realizadas hemos podido observar que el sistema propuesto en este proyecto funciona. Sus resultados han sido buenos respecto a los vecinos obtenidos en cuanto a los usuarios. Además, hemos descubierto la importancia de la Red Neuronal que genera los embeddings. Hemos propuesto una red en este trabajo, a la que hemos añadido dos variantes para comprobar si es necesario tener una red compleja o podemos utilizar redes más sencillas. Lo que hemos descubierto con

ellas es que estaban demasiado especializadas a obtener los embeddings de los usuarios, obteniendo un peor resultado en los ítems.

5.2. Trabajo Futuro

A partir de los resultados obtenidos hemos comprobado que todavía se puede explorar con mayor profundidad el tema tratado en este Trabajo de Fin de Grado.

Para comenzar, sería interesante probar otros tipos de Redes Neuronales como por ejemplo las Redes Convolucionales para observar cómo se comportan. También se podrían explorar otras Redes Profundas con una estructura diferente a la utilizada, ya que estas abarcan un gran número de tipos distintos. Por otro lado, se podría realizar la misma red, pero en vez de utilizar estimadores utilizar la variante de Keras que TensorFlow nos proporciona (como hemos comentado previamente en este trabajo, los estimadores no nos permiten realizar una red con la estructura exacta deseada). Además, el uso de Keras nos podría ayudar a realizar pruebas en mayor detalle, probando la red con distintos parámetros para buscar una solución que mejore a la propuesta.

Si queremos utilizar la misma red, queda pendiente explorar en mayor medida Annoy. Este nos permite utilizar distintas métricas y construir un bosque con distinto número de árboles, que aunque se han tenido en cuenta y puede ser probado con el framework realizado, no ha sido probado en los experimentos con los algoritmos de recomendación por falta de tiempo. También queda pendiente estudiar en profundidad cómo el tratamiento previo del dataset afecta a las redes, siendo posiblemente esta la causa de que los embeddings de los ítems obtenidos no sean los ideales. E incluso queda pendiente comprobar cómo el tamaño del vector de embeddings de un elemento afecta a la caracterización del mismo. Sería útil encontrar el tamaño ideal, el cual nos permita obtener los mejores vecinos en un tamaño de vector lo más pequeño posible, ya que mientras mayor sea este más recursos consume el sistema y mayores serán los tiempos de ejecución.

Finalmente consideramos que sería interesante comparar la caracterización de los elementos realizada por las redes con la factorización de matrices, haciendo las redes equivalentes a esta. O en su defecto obtener los factores latentes de los usuarios e ítems y hacer la misma búsqueda que realizamos actualmente en Annoy pero en el espacio de los factores latentes.

Bibliografía

[1] C. M. Ian MacKenzie and S. Noble, 2013. How retailers can keep up with consumers.

[2] D. Siganos, "Why neural networks?," *Surprise 96*, 1996. Download.

[3] J. Vinagre, A. Jorge, and J. Gama, "An overview on the exploitation of time in collaborative filtering," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 5, 07 2015.

[4] J. de la Fuente O'Connor, *Técnicas de cálculo para sistemas de ecuaciones, programación lineal y programación entera: códigos en FORTRAN y C con aplicaciones de sistemas de energía eléctrica*. Reverté, 1997.

[5] R. Burke, "Hybrid web recommender systems. in the adaptive web, methods and strategies of web personalization," *The Adaptive Web*, vol. 4321, p. 377–408, 2007. Download.

[6] R. Burke, "Hybrid recommender systems: Survey and experiments," *doi.org*, vol. 12, pp. 331–338, 2002. Download.

[7] P. Castells, N. J. Hurley, and S. Vargas, "Novelty and diversity in recommender systems," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 881–918, Springer, 2015.

[8] M. Hassan, "Performance comparison of feed-forward neural networks trained with different learning algorithms for recommender systems," *Computation*, vol. 5, p. 40, 2017. Download.

[9] T. M. et al, "Recurrent neural network based language mode," *INTERSPEECH-2010*, pp. 1045–1048, 2010. Download.

[10] Y. Gong and Q. Zhang, "Hashtag recommendation using attention-based convolutional neural network," *Proceedings of the Twenty-Fifth Inter-national Joint Conference on Artificial Intelligence*, p. 2782–2788, July 2016. Download.

[11] S. R. y Cajal, "Textura del sistema nervioso del hombre y de los vertebrados," *Nicolas Moya*, 1904. Download.

[12] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *arXiv preprint arXiv:1511.06939*, 2015.

[13] A. Töscher and M. Jahrer, "The bigchaos solution to the netflix grand prize," 01 2009.

[14] Y. Koren, "The bellkor solution to the netflix grand prize," 2009.

[15] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems*, (New York, NY, USA), 2016.

[16] W. Koehrsen, 2018. Neural Network Embeddings Explained: How deep learning can represent War and Peace as a vector.

[17] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah, "Wide & deep learning for recommender systems," *arXiv:1606.07792*, 2016.

[18] E. Bernhardsson. https://www.slideshare.net/erikbern/approximate-nearest-neighbor-methods-and-vector-models-nyc-ml-meetup, accedido en Mayo 2019, 2015. Approximate nearest neighbor methods and vector models.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID