

Escuela Politécnica Superior

20
21

Trabajo fin de máster

Explotación de características de secuencias para su uso en sistemas de recomendación



Ricardo Sánchez-Guzmán Hitti

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

**UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR**



**Máster en Investigación e Innovación en Inteligencia Computacional y Sistemas
Interactivos (MU I2-ICSI)**

TRABAJO FIN DE MÁSTER

**Explotación de características de secuencias para
su uso en sistemas de recomendación**

Autor: Ricardo Sánchez-Guzmán Hitti

Tutor: Alejandro Bellogín Kouki

septiembre 2020

Todos los derechos reservados.

Queda prohibida, salvo excepción prevista en la Ley, cualquier forma de reproducción, distribución comunicación pública y transformación de esta obra sin contar con la autorización de los titulares de la propiedad intelectual.

La infracción de los derechos mencionados puede ser constitutiva de delito contra la propiedad intelectual (*arts. 270 y sgts. del Código Penal*).

DERECHOS RESERVADOS

© Septiembre de 2020 por UNIVERSIDAD AUTÓNOMA DE MADRID
Francisco Tomás y Valiente, n° 1
Madrid, 28049
Spain

Ricardo Sánchez-Guzmán Hitti

Explotación de características de secuencias para su uso en sistemas de recomendación

Ricardo Sánchez-Guzmán Hitti

C\ Francisco Tomás y Valiente N° 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

AGRADECIMIENTOS

En primer lugar me gustaría dar las gracias a Alejandro Bellogín por todo su apoyo durante este dos años, estando siempre ahí cuando tenía algún problema aún en estos tiempos tan difíciles para todos.

A mi familia que le agradezco que me haya soportado todo estos años y que siempre han estado ahí para apoyarme en todo este camino que empezó hace ya varios años y que termina de la mejor manera posible.

Y cómo no a mis compañeros de la carrera, sin ellos este camino sin duda habría sido mucho más difícil, y en especial a Sergio por aguantarme en todas las prácticas de la carrera y el máster, espero que todos consigan cumplir sus objetivos.

RESUMEN

El turismo es un sector que en muchos países suele tener un peso muy importante en su economía, como es el caso de España o Italia, siendo un dominio poco explotado (aunque en auge) en el mundo de la recomendación. Un problema que surge al recomendar a un usuario un lugar de interés es que deberíamos tener en cuenta tanto la fecha en la que el usuario realiza el viaje como la zona a la que viaja para poder recomendarle lugares que sean cercanos y tengan una cierta relación con los gustos del usuario.

Este Trabajo de Fin de Máster consiste en el estudio y explotación de características de dominios turísticos, aplicando algoritmos de aprendizaje automático como las redes neuronales para poder aprender representaciones eficientes de estas características. Para lograr aprender de forma eficiente estas representaciones utilizaremos vectores pre-entrenados usando modelos famosos utilizados en el dominio del procesamiento del lenguaje natural, como Word2Vec o fastText.

Para medir el rendimiento del modelo hemos utilizado dos conjuntos de datos relacionados con el dominio turístico de ámbito público: Foursquare y Yelp. Se han utilizado varios baselines de distintas familias para medir el rendimiento de nuestro modelo: técnicas básicas, secuenciales y modelos más complejos como las redes neuronales de tipo CNN y RNN. Evaluaremos los resultados haciendo una división temporal de los datos y utilizaremos las métricas de precisión y recall para medir la eficacia de los algoritmos.

Como conclusión, hemos observado que nuestro modelo obtiene resultados prometedores en ambos conjuntos de datos y, aunque es dependiente de la cantidad de atributos que usemos, puede funcionar bien en dominios con escasez de información. Además, hemos comprobado que los atributos geográficos suelen dar mejores resultados que los atributos contenidos en las reseñas escritas por los usuarios y que la fecha suele ser uno de los atributos más relevantes. Cabe destacar que no siempre se da el caso de que más atributos produzcan mejor resultado, ya que puede depender de la operación que se realice sobre ellos (bien sea la media o el máximo al calcular los vectores) y puede llegar a empeorar los resultados utilizar atributos con resultados pobres.

PALABRAS CLAVE

Sistemas de recomendación, Secuencias, Redes neuronales, Puntos de interés

ABSTRACT

Tourism industry is a very important sector in many countries, as it usually has a very important weight in their economy – for instance, this is the case of Spain or Italy. However, it is an under-exploited (although steadily growing) domain in the recommendation community. Some problems arise when we have to recommend a user a place of interest, since we must take into account the date on which the user makes the trip or the area in which she travels to, so that near places can be recommendable, at the same time, they should keep some (positive) relation with the user's preferences.

This Master's Thesis consists of the study and exploitation of characteristics of tourism domains, by applying machine learning algorithms such as neural networks to learn efficient representations of these characteristics. To achieve efficient learning of these representations we will use pre-trained vectors, training them with famous models used in the natural language processing domain, such as Word2Vec or fastText.

To measure the performance of the model we have used two public domain datasets from the tourism domain: Foursquare and Yelp. Several baselines of different families have been used to compare against the performance of our model: basic techniques, sequential, and more complex models such as those based on neural networks like CNN and RNN. We evaluate the results by dividing the data in a timely fashion and then using precision and recall metrics to measure the performance of the algorithms.

As a conclusion, we have observed that our model obtains promising results in both datasets, although it depends on the amount of attributes we use; interestingly, it can work well in domains with scarce information. In addition, we have found that geographic attributes usually give better results than attributes extracted from reviews written by the users, and the date is usually one of the most relevant attributes. Note that it is not always the case that more attributes give better results, since it may depend on the operation performed on them (either the average or maximum of the vectors), and in some situations it may worsen the results.

KEYWORDS

Recommendation systems, Sequences, Neural networks, Points of interest

ÍNDICE

1	Introducción	1
1.1	Motivación del proyecto	1
1.2	Objetivos y enfoque	2
1.3	Estructura del documento	2
2	Estado del arte	5
2.1	Sistemas de Recomendación	5
2.1.1	Notación y conceptos básicos	5
2.1.2	Fuentes de información alternativa	5
2.1.3	Estrategias de recomendación	7
2.2	Redes Neuronales	10
2.2.1	Redes neuronales recurrentes	11
2.2.2	Redes neuronales convolucionales	14
2.2.3	Embeddings	17
2.3	Sistemas de recomendación basados en secuencia	19
2.3.1	Procesamiento de Datos	19
2.3.2	Algoritmos	20
2.3.3	Estrategias generales de entrenamiento	22
2.4	Evaluación de Sistemas de Recomendación	23
3	Diseño e Implementación	25
3.1	Diseño	25
3.1.1	Diagrama de trabajo	25
3.1.2	Descripción del entorno de trabajo	26
3.1.3	Librerías externas utilizadas	26
3.2	Descripción del modelo	27
3.2.1	Arquitectura de la red neuronal	27
3.2.2	Tratamiento de las características	28
4	Experimentos y Resultados	33
4.1	Configuración del entorno	33
4.1.1	Conjuntos de datos	33
4.1.2	Metodología experimental	34
4.1.3	Baselines	36

4.2 Foursquare	37
4.2.1 Influencia de los atributos	38
4.2.2 Análisis de hiper-parámetros	40
4.3 Yelp	40
4.3.1 Influencia de los atributos	43
4.3.2 Análisis de hiper-parámetros	44
4.4 Discusión de los resultados	44
5 Conclusiones y Trabajo Futuro	47
5.1 Conclusión	47
5.2 Trabajo Futuro	48
Bibliografía	51
Definiciones	53

LISTAS

Lista de ecuaciones

2.1	Ecuación de similitud usando la estrategia basada en contenido.	8
2.2	Ecuación de Filtrado colaborativo usando KNN basado en usuario.	9
2.3	Ecuación de la constante de normalización para KNN en filtrado colaborativo.	9
2.4	Ecuación de Filtrado colaborativo usando KNN basado en ítem.	9
2.5	Ecuación de similitud mediante Coseno.	9
2.6	Ecuación de similitud mediante Correlación de Pearson.	10
2.7	Ecuación de similitud mediante Jaccard.	10
2.8	Ecuación de la puerta de entrada de la unidad LSTM.	12
2.9	Ecuación de la puerta de salida de la unidad LSTM.	12
2.10	Ecuación de la puerta del olvido de la unidad LSTM.	12
2.11	Ecuación del nuevo estado calculado en la unidad LSTM.	13
2.12	Ecuación de la combinación del estado anterior y el nuevo en la unidad LSTM.	13
2.13	Ecuación de la salida de la unidad LSTM.	13
2.14	Ecuación de la puerta de actualización de la unidad GRU.	13
2.15	Ecuación de la puerta de reinicio de la unidad GRU.	14
2.16	Ecuación de estado de la unidad GRU.	14
2.17	Ecuación de salida de la unidad GRU.	14
2.18	Ecuación de la convolución dilatada.	16
2.19	Ecuación de TOP1.	22
2.20	Ecuación de BPR.	23
2.21	Ecuación de entropía cruzada.	23
2.22	Ecuación softmax.	23
2.23	Ecuación de MAE.	23
2.24	Ecuación de RMSE.	24
2.25	Ecuación de Precisión.	24
2.26	Ecuación de Recall.	24
4.1	Ecuación de Haversine para calcular la distancia entre dos POIs.	37

Lista de figuras

2.1	Ejemplo de matriz de utilidad.	6
-----	-------------------------------------	---

2.2	Arquitectura de una red neuronal recurrente	11
2.3	Arquitectura de una red Long-Short Term Memory	13
2.4	Arquitectura de una red GRU	14
2.5	Arquitectura de una red neuronal convolucional	15
2.6	Diagrama de una red convolucional formada por tres capas ocultas	16
2.7	Diagrama de una red convolucional dilatada formada por tres capas ocultas	17
2.8	Representación de los dos modelos propuestos en Word2Vec	18
3.1	Metodología seguida durante el proyecto	25
3.2	Arquitectura general de nuestro modelo	28
3.3	Ejemplo de fusión de atributos utilizando la operación de max-pooling y average-pooling	29
3.4	Diagrama de Prod2Vec basado en el modelo Skyp-Gram de Word2Vec	30
4.1	Estudio de la sensibilidad de los hiperparámetros con el conjunto de datos de Foursquare	41
4.2	Estudio de la sensibilidad de los parámetros con el conjunto de datos de Yelp	45

Lista de tablas

2.1	Tabla con la notación utilizada en las métricas de Precisión y Recall	24
3.1	Resumen de las características del equipo principal utilizado durante el proyecto	26
3.2	Resultados obtenidos en la predicción de POIs utilizando los modelos basados en Word2Vec y Skip-Gram	30
4.1	Estadísticas de los datos utilizados en los experimentos	34
4.2	Parámetros de los baselines a optimizar utilizando la partición de entrenamiento y validación	35
4.3	Mejor configuración obtenida en cada baseline para el conjunto de datos de Foursquare	35
4.4	Mejor configuración obtenida en cada baseline para el conjunto de datos de Yelp	36
4.5	Resultados de los baselines y de nuestro modelo con el conjunto de datos de Foursquare. En negrita, el mejor resultado para cada métrica.	38
4.6	Utilidad de la información aportada por los distintos atributos. En negrita, el mejor resultado por cada bloque, y subrayado el mejor resultado para esa métrica.	39
4.7	Resultados de los baselines y de nuestro modelo con el conjunto de datos de Yelp. En negrita, el mejor resultado para cada métrica.	42
4.8	Utilidad de la información aportada por los distintos atributos. Mejor resultado en cada bloque, en negrita. Nótese que se ha incluido una fila extra donde se han fusionado todos los atributos de ese bloque para determinar la influencia de dicho bloque (denotado como <i>Fusión</i>). El mejor resultado por cada métrica se indica con subrayado.	43

INTRODUCCIÓN

En este primer capítulo se describe la motivación por la cual se realiza este proyecto, los objetivos del mismo y la estructura del documento final.

1.1. Motivación del proyecto

Durante los últimos años, el uso de los sistemas de recomendación ha tenido un desempeño fundamental en grandes empresas como Netflix, Spotify o YouTube, debido a que en la era del *Big Data* el poder seleccionar los elementos más relevantes para un usuario entre tanta información es algo vital. El objetivo principal de esta herramienta es el de filtrar los elementos más relevantes para un usuario entre tanta información y así lograr que el usuario no se sature.

Los sistemas de recomendación son indispensables a la hora de trabajar con elementos como películas, música o productos de comercio electrónico (*e-commerce*) que han tenido un gran auge durante todo este período, pero a la hora de recomendar se echa en falta la explotación de más características de los usuarios en estos sistemas, por ejemplo, gracias a esta información auxiliar se podrían mitigar problemas como el arranque en frío (*cold start*) [1].

Esto hace unos años era una tarea compleja puesto que existían pocos conjuntos de datos con más información relevante además de los ids de usuario y de items, así como su puntuación por parte del usuario. La mayoría de estos conjuntos de datos pertenecían al dominio de las películas como MovieLens, y además no poseían una buena calidad temporal de los datos (por ejemplo, existen casos donde se realizan múltiples interacciones en el sistema en menos de un segundo), pero en los últimos años han empezado a surgir conjuntos de datos públicos que cuentan con más características.

Por ejemplo, existen características de los comentarios o reseñas hechos por un usuario como la fecha en la que se hizo o si a la gente le pareció de utilidad que, aunque no es un factor determinante, en muchas ocasiones los usuarios se basan en los comentarios de terceros para decidirse a adquirir un producto. Esto ocurre a menudo en dominios como el e-commerce o a la hora de elegir un sitio al que ir, por ejemplo, para comer como un restaurante o un bar [2].

En particular, en dominios como el turístico, donde se recomiendan sitios para visitar en un momento dado, características como en qué fecha se realiza un viaje o qué categorías suelen tener los lugares que visita un usuario suelen ser buenos indicadores de lo que le gusta a un usuario [3]. Además, pueden ser una información muy útil a la hora de recomendar, puesto que un usuario que esté visitando en un momento dado un lugar seguramente le interese visitar lugares cercanos a esa localización, y también aprender a qué lugares viaja un usuario en fechas concretas puesto que en verano un usuario seguramente vaya más a la playa y en invierno decida otros lugares como la montaña.

En este trabajo se propone el uso de modelos basados en redes neuronales profundas (ó *Deep Learning*) y técnicas de *embeddings* para capturar la información de las secuencias de un usuario en un período de tiempo determinado. El objetivo es utilizar las características de los puntos de interés (ó *Points of interest (POIs)*) contenidos en una secuencia de lugares visitados por un usuario para poder recomendar el siguiente lugar con mayor acierto. Para comprobar el rendimiento del sistema de recomendación se utilizarán como *baselines* librerías externas y dos conjuntos de datos de dominio público como Yelp y Foursquare.

1.2. Objetivos y enfoque

Los objetivos que propone este TFM son los siguientes:

- Plantear un nuevo modelo de red neuronal basado en la explotación de características dada una secuencia vista por un usuario.
- Realizar un estudio de la influencia de los atributos utilizados en el modelo.
- Desarrollo de un framework con distintos modelos basados en redes neuronales y la integración de librerías externas en el mismo para poder realizar una comparación entre las mismas.
- Evaluación de las distintas configuraciones del modelo neuronal mediante medidas de ranking como Precisión y Recall. Los conjuntos de datos utilizados serán el de Yelp (versión 2020) y Foursquare (New York).

Se pretende utilizar los conocimientos obtenidos en las asignaturas siguientes del máster en Investigación e Innovación en Inteligencia Computacional y Sistemas Interactivos: Recuperación de Información (introducción a los sistemas de recomendación y medidas de evaluación) y Aprendizaje Automático (conocer las técnicas de aprendizaje automático más importantes para aplicarlas a los sistemas de recomendación) y, más concretamente, para desarrollar, ampliar y combinar estos conocimientos en el ámbito de los sistemas de recomendación y redes neuronales.

1.3. Estructura del documento

El presente documento está dividido en 5 capítulos, los cuales se detallan a continuación:

Capítulos

Capítulo 1. Introducción: Breve descripción de la motivación del proyecto, los objetivos a cumplir y la estructura del documento.

Capítulo 2. Estado del Arte: Se lleva a cabo un repaso de las áreas de conocimiento estudiadas, haciendo una especial mención a las técnicas utilizadas en los sistemas de recomendación, las redes neuronales y el uso de estas en los sistemas de recomendación.

Capítulo 3. Diseño e Implementación: En este capítulo se detalla el entorno de trabajo, las librerías utilizadas y la descripción del modelo.

Capítulo 4. Pruebas y Evaluación: Se analizan y comparan los resultados obtenidos con distintos conjuntos de datos y se hace una discusión de los mismos.

Capítulo 5. Conclusiones y Trabajo Futuro: Se exponen las conclusiones obtenidas y el potencial del proyecto en futuros ámbitos de investigación.

ESTADO DEL ARTE

2.1. Sistemas de Recomendación

En estos últimos años los grandes volúmenes de datos que circulan por la red hacen que los sistemas de recomendación hayan ganado un peso muy importante en nuestras vidas, esto se debe a que es necesario una herramienta que filtre los datos que puedan tener valor para un usuario. A continuación se describen las estrategias más famosas utilizadas en los sistemas de recomendación:

2.1.1. Notación y conceptos básicos

Un sistema de recomendación se puede definir como un sistema compuesto por usuarios $u \in U$ e ítems $i \in I$, donde $r(u, i)$ representa la valoración de un usuario sobre un ítem específico, el objetivo de estos sistemas suele ser la predicción de la puntuación de un ítem i para un usuario u que aún no ha visto o interactuado con ese ítem [4].


Una forma eficiente de tratar esta información es mediante la representación de una matriz que relacione usuarios e ítems: en las filas de la matriz colocamos los usuarios y en las columnas los ítems donde la intersección de estos elementos (el contenido de la celda de la matriz) es la puntuación que le da ese usuario a ese ítem; normalmente esta puntuación está acotada entre unos rangos predefinidos, los más habituales suelen ser [1-5] o [1-10]. En la Figura 2.1 se puede ver un ejemplo de una matriz de utilidad.

Esta matriz generalmente tiene una gran dispersión de los datos y en muchas ocasiones es necesario rellenar los ítems que no han sido puntuados, por ejemplo calculando la media de puntuaciones de ese ítem hechas por otros usuarios.

2.1.2. Fuentes de información alternativa

Existen ocasiones en las que no hay disponible información explícita de los usuarios e ítems como las puntuaciones. En estos casos es necesario utilizar otras fuentes de información para mitigar los

Items \mathcal{I}



Users \mathcal{U}








			1	3		2		4			3	
	1	2	5		4		1		2	4		5
	4		?	3	5		5				2	
		2			5	4	4		5			4
		3	4		5			4	3	5		4
	3		2		1	5		3			5	
	3			2			3		5		1	

Figura 2.1: Matriz de puntuaciones donde los usuarios forman las filas de la matriz y los ítems las columnas. Extraída de las transparencias de la asignatura de recuperación de información (con permiso del profesor).

efectos de la escasez de datos. Por este motivo, los sistemas de recomendación deben ser capaces de utilizar más características sobre un usuario o un ítem. A continuación se describen las fuentes de información más importantes en el dominio turístico, que es el foco de este trabajo:

Información temporal: una de las características más importantes a la hora de poder predecir el siguiente ítem en una secuencia es saber cuándo se hicieron las anteriores recomendaciones, esta información nos permite saber los gustos de un usuario a lo largo del tiempo: desde saber los sitios que visita en un fin de semana a los lugares a los que va en invierno. Se podría ajustar la recomendación y no cometer posibles errores como recomendar a un usuario visitar la playa en invierno.

Información geográfica: otro aspecto importante es la localización geográfica de un lugar; imaginemos que estamos visitando la ciudad de Amsterdam y queremos visitar los sitios más populares posibles, pero el sistema de recomendación nos sugiere un restaurante en Róterdam. Al no tener en cuenta las distancias entre distintos sitios se pueden cometer errores graves puesto que tiene sentido que un usuario visite primero lugares cercanos a su posición y luego lugares más lejanos.

Información basada en los comentarios: en muchas ocasiones nos fijamos en los comentarios de otras personas para ver qué opinión tienen estos usuarios sobre un producto o la experiencia vivida en un lugar. Si la cantidad de opiniones negativas es alta normalmente el usuario opta por rechazar ese lugar; por otro lado, existen características de los comentarios como la cantidad de personas a las que le ha parecido útil o gracioso un comentario, esto permite tener un *feedback* de los usuarios validado por ellos mismos que puede ser explotado por el sistema de recomendación.

Aunque nos hemos centrado en las características del dominio turístico para el trabajo de este TFM, existen cientos de atributos que pueden ser explotados en los sistemas de recomendación en general, por ejemplo, en YouTube [5] se utiliza el tiempo que ha estado un usuario viendo un vídeo o la cantidad de vídeos que ha visto un usuario en un canal para realizar la recomendación, y en ocasiones en los que no se tiene datos de ese usuario se suelen utilizar los rasgos demográficos para mitigar el arranque en frío.

2.1.3. Estrategias de recomendación

A día de hoy existen varias estrategias a la hora de realizar una recomendación personalizada, donde las técnicas más conocidas son:

Recomendación basada en contenido (ó *Content-Based*): se utiliza la información que tienen los ítems como las palabras de la descripción o los meta-datos que identifiquen a ese ítem, y después se observa el historial de usuario a largo plazo y se recomiendan los ítems más similares basándose en los ítems vistos previamente por el usuario.

Ventajas: la primera ventaja es que no es necesario tener las puntuaciones de un ítem, solo sus meta-datos; esto permite hacer una recomendación de cualquier tipo de ítem que tenga algún tipo de categoría o descripción. Permite realizar una recomendación personalizada puesto que se basa en los ítems que ha visto un usuario previamente, sin ser influenciado por el resto de usuarios.

Desventajas: esta estrategia tiene varios problemas importantes, el primero es la necesidad de tener mucha información de un usuario para poder hacer una recomendación eficiente y normalmente esto no suele suceder a no ser que el usuario sea muy activo, y el segundo es la falta de diversidad, ya que si al usuario le gustan varias películas de un género concreto como el drama llegará un punto en el que el sistema sólo recomendará películas de drama.

Filtrado colaborativo (*del inglés, Collaborative Filtering*): en este caso se explotan patrones entre usuarios e ítems para recomendar, por ejemplo, para recomendar películas que un usuario ha visto y el otro no, suelen utilizarse rasgos demográficos o patrones de comportamiento entre otras características, principalmente existen dos estrategias: las basadas en KNN y las basadas en modelos; ambas se comentarán en detalle más adelante.

Ventajas: la principal ventaja con respecto a la recomendación basada en contenido es que no es necesario tener los meta-datos de un ítem, solo su puntuación; en este caso hay más posibilidades de obtener ítems novedosos.

Desventajas: de nuevo volvemos a encontrarnos los mismos problemas que en el caso anterior: es necesario tener suficiente información del usuario y en caso

de usuarios o ítems nuevos esta tarea se vuelve compleja.

Recomendación híbrida: este método consiste en la unificación de los dos métodos anteriores (o, en principio, de cualesquiera que se quieran combinar), se puede hacer de varias formas como la combinación de la salida de ambas estrategias o el uso de componentes de ambas estrategias. La combinación de varios métodos suele tener mejores resultados que por separado.

2.1.3.1. Algoritmos de recomendación no personalizada

Hay casos donde existe una escasez de información o se quieren realizar pruebas sobre el sistema, para este tipo de situaciones existen dos métodos de recomendación:

Recomendación aleatoria: este método selecciona ítems al azar y los recomienda; por sí solo no suele ser muy útil ya que depende de la densidad de los datos, pero este método puede ser útil para realizar una exploración previa del sistema de recomendación y, en particular, confirmar si un sistema más complejo produce resultados interesantes o aleatorios (algo que, en general, no es fácil de distinguir, puesto que los valores de las métricas, debido a la baja densidad de los datos, suelen ser muy bajos).

Recomendación basada en popularidad: en este caso normalmente se utilizan las frecuencias de los ítems para recomendar, por ejemplo una película que ha sido vista por muchos usuarios y con una alta puntuación tendrá más posibilidades de ser recomendada. Este método tiende a obtener resultados bastante competitivos en métricas de acierto, aunque es más limitado en otras dimensiones (como novedad). Es interesante incluirlo en las comparaciones experimentales para confirmar si los métodos son verdaderamente personalizados o están aprendiendo a recomendar los ítems más populares de una forma muy compleja.

2.1.3.2. Algoritmos de recomendación personalizada

Los modelos más usados en la estrategia de recomendación basada en contenido son los siguientes: modelo de espacio vectorial (ó *Vector Space Model*), centroides o métodos probabilísticos.

Para la recomendación **basada en contenido** se suele utilizar una función de similitud:

$$\text{sim}_{CB}(u, i) = \cos(u, i) = \frac{\sum_{k \in K} u_k i_k}{|u| \cdot |i|} \quad (2.1)$$

donde u es el vector promedio que representa a todos los ítems puntuados por un usuario e i el vector representativo de un ítem, $|u|$ e $|i|$ serían el modulo de los vectores u e i respectivamente.

- 1.– En un primer paso es necesario tener los meta-datos de los ítems, así que lo primero que se deberá hacer es extraer la información de estos como categorías, géneros u otros atributos relevantes y procesarla.
- 2.– El siguiente paso será crear un perfil por cada usuario dada la lista de sus ítems.
- 3.– Una vez se tenga la representación de los ítems y de los usuarios se recomendará a los usuarios los ítems que menos distancia tengan con el usuario, es decir, aquellos con mayor similitud según la Ecuación 2.1.

Por otro lado, uno de los algoritmos más famosos a la hora de utilizar la técnica de **filtrado colaborativo** es el basado en vecinos cercanos o KNN (del inglés *k-nearest neighbours*), esto es principalmente por dos motivos: es sencillo e intuitivo. A continuación se muestran sus dos variantes:

KNN basado en usuario este algoritmo utiliza los k vecinos más similares a un usuario para asignar puntuaciones a los ítems que un usuario no ha visto pero sus vecinos sí.

$$\hat{r}(u, i) = c \sum_{v \in \eta_k(u)} sim(u, v) r(v, i) \quad (2.2)$$

donde $\eta_k(u)$ indica el número de vecinos con los que se quiere comparar, $sim(u, v)$ es una similitud colaborativa (no de contenido, como la definida anteriormente) entre el usuario u y su vecino v y $r(v, i)$ la puntuación del vecino v sobre el ítem i .

Siendo c una constante para normalizar el rating, esta normalización se aplica tanto en KNN basado en usuario como KNN basado en ítem:

$$c = \frac{1}{\sum_{v \in \eta_k(u)} |sim(u, v)|} \quad (2.3)$$

KNN basado en ítem similar al KNN basado en usuario pero con ítems, en este caso lo que se busca a la hora de dar una puntuación a un ítem que un usuario no ha visto es buscar ítems similares a estos que el usuario ya ha puntuado.

$$\hat{r}(u, i) = c \sum_{j \in I(u)} sim(i, j) r(u, j) \quad (2.4)$$

Para el cálculo de las similitudes tanto de usuarios como de ítems existen múltiples métricas, siendo las más relevantes las siguientes (vamos a indicar la definición de similitudes entre ítems, las de usuario se pueden derivar de manera equivalente):

Similitud coseno: una de las funciones más utilizadas donde cada elemento es representado como un vector, la similitud entre dos vectores será la distancia coseno entre ambos.

$$sim_{CF}(i, j) = cos(i, j) = \frac{\sum_{u \in U} r(u, i)r(u, j)}{\sqrt{\sum_{u \in U} r(u, i)^2 \sum_{u \in U} r(u, j)^2}} \quad (2.5)$$

Si nos fijamos en la fórmula podemos observar que es muy parecida a la usada en el caso de la recomendación basada en contenido (ver Ecuación 2.1), lo único que cambia es la representación de los vectores sobre los que se calcula el ángulo coseno: en este caso son las puntuaciones de cada usuario sobre los dos ítems, en aquél caso eran las palabras o meta-datos correspondientes a los ítems.

Correlación de Pearson: este método es muy parecido a la similitud coseno pero añadiendo otro factor para capturar la tendencia con la que un usuario puntúa, ya que podemos encontrarnos usuarios que normalmente puntúan películas muy alto o muy bajo, con la correlación de Pearson es posible capturar esta tendencia.

$$sim_{CF}(i, j) = Pearson(i, j) = \frac{\sum_{u \in U} (r(u, i) - \bar{r}(i))(r(u, j) - \bar{r}(j))}{\sqrt{\sum_{u \in U} (r(u, i) - \bar{r}(i))^2 \sum_{u \in U} (r(u, j) - \bar{r}(j))^2}} \quad (2.6)$$

Similitud Jaccard: método que realiza operaciones entre conjuntos, se realiza la operación de intersección en el numerador y la de unión en el denominador, el resultado está acotado entre [0-1] siendo 0 el peor valor y 1 el mejor.

$$sim_{CF}(i, j) = Jaccard(i, j) = \frac{|U_i \cap U_j|}{|U_i \cup U_j|} \quad (2.7)$$

donde U_i indica los usuarios que han interactuado con el ítem i .

2.2. Redes Neuronales

En este subapartado se explican los tipos de redes neuronales más importantes de manera general, una descripción de sus variantes y tipos, centrándonos en los conceptos y arquitecturas necesarias para entender el modelo propuesto en este trabajo.

Cuando hablamos de una red neuronal nos referimos a un sistema de procesamiento bioinspirado en el que el procesamiento de la información tiene lugar en elementos llamados neuronas. Las neuronas están conectadas entre sí y transmiten información entre ellas, estas conexiones entre neuronas tienen pesos y cada neurona tiene una función de activación, que en caso de que se supere un cierto umbral disparará la salida de la neurona o la inhibirá [6].

Existen varios tipos de redes como las redes neuronales profundas de la que derivan las redes neuronales recurrentes (sección 2.2.1) o las redes convolucionales (sección 2.2.2). A continuación se

describe cada una de ellas:

2.2.1. Redes neuronales recurrentes

Cuando hablamos de las RNN (del inglés, Recurrent Neural Network) [7] nos referimos a aquellas redes que son capaces de guardar el estado interno y combinarlo con los valores de entrada de la red para producir una salida, es decir, son capaces de tener memoria, para lograr esto se utiliza la retroalimentación. Una misma neurona podría estar conectada a las neuronas de la capa posterior, anterior e incluso a ella misma. Por este motivo el uso de las redes neuronales recurrentes las hace idóneas para la predicción de secuencias, como podrían ser la predicción de cadenas de texto o de sonido. En la figura 2.2 se puede apreciar la arquitectura de la red.

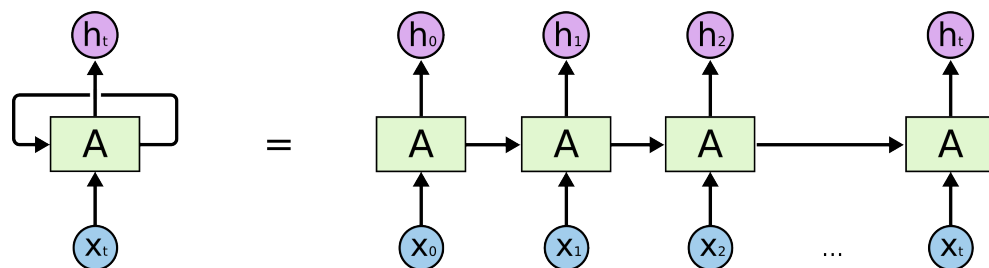


Figura 2.2: Arquitectura de una red neuronal recurrente [8].

Una de las diferencias más significativas de las RNN en comparación con otras redes, es que las capas que forman la red a la hora de predecir la siguiente palabra de una frase comparten los mismos parámetros de entrada, esto permite utilizar menos parámetros a la hora de trabajar con este tipo de redes.

2.2.1.1. LSTM

Existe un gran problema a la hora de utilizar las redes neuronales recurrentes, este problema reside en la cantidad de información que puede "guardar" la red, las RNN son capaces de guardar pequeñas cantidades de información mediante bucles pero en el caso de necesitar guardar datos que tengan una dependencia temporal muy extensa la red comenzaría a fallar en sus predicciones.

Por ejemplo, imaginemos que queremos predecir un punto en una serie temporal caótica, la red utiliza como valores de entrenamiento los últimos ocho puntos de la serie para predecir el noveno, pero ¿y si necesitáramos los últimos cien puntos para predecir el punto ciento uno?, la red comenzaría a fallar, esto es debido a que podemos considerar cada intervalo de tiempo como una capa oculta conectada con el resto, la información pasa de capa en capa para después propagar el error hacia atrás, el problema surge en que si tenemos muchos intervalos de tiempo se produce el problema del desvanecimiento del gradiente.

La red utiliza el descenso por gradiente para entrenar la red, basándose en los errores actuales y pasados para calcular los pesos de las sinapsis, esto hace que el entrenamiento sea muy costoso y los cambios en los pesos no lleguen a producirse por pérdida de información (desvanecimiento del gradiente). Durante mucho tiempo esto fue un problema hasta que en el año 1997 se propone una nueva variante de la red RNN llamada LSTM (*del inglés, Long Short-Term Memory*) [9] por Sepp Hochreiter y Jürgen Schmidhuber. Por tanto, las LSTM resuelven el problema de las RNN ya que consiguen tener una memoria a más largo plazo y esto permite almacenar, por ejemplo, el contexto de un usuario, algo fundamental en la personalización.

La red LSTM está formada por bloques de memoria o unidades LSTM, donde cada unidad está formada por tres puertas multiplicativas con retroalimentación: puerta de entrada, puerta de salida y una puerta de olvido, y además incluye una célula de memoria que será la encargada de guardar el valor.

La puerta de entrada se encarga de controlar en qué medida van apareciendo nuevos valores, actúa como filtro en el módulo de memoria, protegiéndola de valores irrelevantes.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (2.8)$$

donde W_i son los pesos recurrentes, h_{t-1} representa la salida de la neurona en un tiempo anterior $t - 1$, x_t será el valor de la secuencia que se analiza en un tiempo t y b_i es el vector de bias. Para el resto de puertas se sigue la misma nomenclatura y el significado de los parámetros es el mismo.

La puerta de salida se encarga de regular el peso del valor contenido en la célula de memoria a la hora de calcular la salida de ese bloque, con esto conseguimos evitar la generación de información irrelevante por parte de nuestra unidad LSTM.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (2.9)$$

La puerta de olvido es la encargada de mantener el valor en la célula de memoria en cada tiempo, su función sería la de elegir qué valores y por cuánto tiempo permanecen en el módulo de memoria.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad (2.10)$$

Todos estos bloques llevan unos pesos asociados, indicados en las ecuaciones como W_i , W_o y

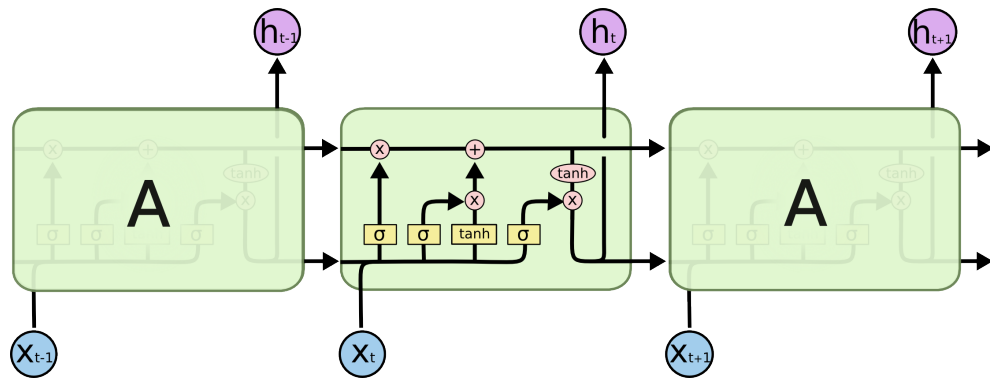


Figura 2.3: Arquitectura de una red Long-Short Term Memory [8].

W_f , que hacen referencia a los pesos de las puertas de entrada, salida y del olvido, respectivamente. El resto de parámetros vienen calculados por las siguientes ecuaciones:

Cálculo del nuevo estado

$$\hat{c}_t = \tanh(W_c[h_{t-1}, X_t] + b_c) \quad (2.11)$$

Combinación del estado anterior y el nuevo en la unidad LSTM.

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \hat{c}_t \quad (2.12)$$

donde C_{t-1} es el estado de la neurona en el tiempo $t - 1$ y f_t e i_t son el peso que tendrán los estados C_{t-1} y \hat{C}_t .

Salida de la unidad LSTM

$$h_t = o_t \cdot \tanh(C_t) \quad (2.13)$$

2.2.1.2. GRU

Las GRU (*del inglés, Gated Recurrent Unit*) se pueden considerar una versión simplificada de las LSTM y constan de dos puertas:

Puerta de actualización: esta puerta se encarga de regular la cantidad de información que se quiere mantener para el siguiente tiempo $t+1$.

$$z_t = \sigma(W_z[h_{t-1}, x_t]) \quad (2.14)$$

Puerta de reinicio: en este caso se encarga de definir la unión de la nueva información

que recibe la neurona con el contenido anterior de esta.

$$r_t = \sigma(W_r[h_{t-1}, x_t]) \quad (2.15)$$

En este caso h_t tiene dos funciones, la primera es la de mantener la información y la segunda ser la salida de la unidad GRU.

$$\hat{h}_t = \tanh(W[z_t \cdot h_{t-1}, x_t]) \quad (2.16)$$

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \hat{h}_t \quad (2.17)$$

al igual que las LSTM, el parámetro W indica el peso de las conexiones de las puertas GRU.

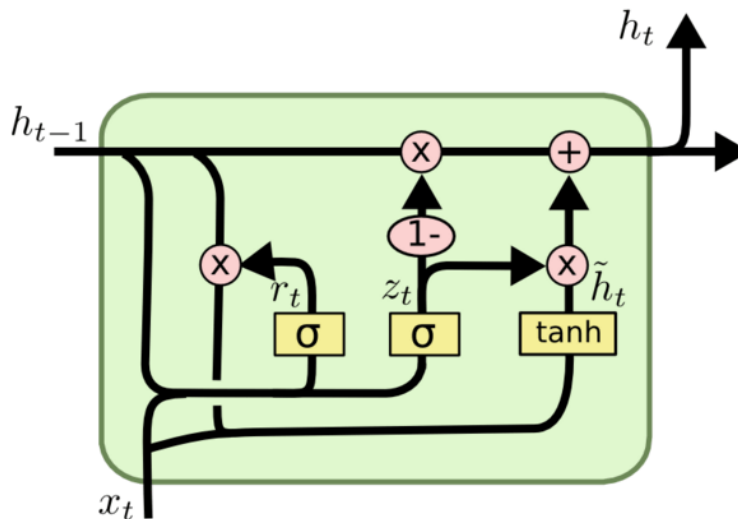


Figura 2.4: Arquitectura de una red GRU [8].

Hay que destacar las ventajas de este tipo de redes neuronales recurrentes y es que, al ser una versión simplificada de las LSTM, tiene menos parámetros y es más rápida que estas a la hora de entrenar el modelo [10], obteniendo mejores resultados en conjuntos de datos pequeños, aunque las LSTM siguen siendo más potentes en contextos más largos y complejos.

2.2.2. Redes neuronales convolucionales

Las redes CNN (*del inglés, Convolutional Neural Network*) [11] han ganado fuerza en la última década y se han utilizado sobre todo en el ámbito del procesamiento de imágenes, ya que son capaces

de analizar características muy complejas con gran precisión. Se basan en la operación de **convolución**, la cual es una función que transforma la señal de entrada en una nueva señal de salida y se suele representar con el símbolo $*$.

El proceso es sencillo: tenemos dos matrices donde la primera será la matriz de números que representa a la imagen o los *embeddings* (según la tarea en la que estemos trabajando), y otra matriz más pequeña que actuará de núcleo (también llamado *kernel*). Lo que haremos entonces será deslizar el núcleo por la matriz de datos e ir realizando la operación de convolución por toda la imagen calculando el producto de un punto entre el núcleo y las entradas de la matriz.

Otra operación importante es la de **pooling**, que reduce los datos a cambio de sacar los datos más representativos, por ejemplo si tenemos una matriz de 3x3 donde el valor más alto es 5 si hacemos una operación max-pooling obtendremos este valor como representación de nuestra matriz de 3x3, también podemos calcular la media de los valores de la matriz como representación general.

Por último, una vez hechas estas dos operaciones realizamos una operación de aplanamiento (ó *flatten*) sobre la matriz obtenida para obtener una salida.

Es importante remarcar que tanto las operaciones de convolución como las de pooling se pueden llevar a cabo tantas veces como se requiera y que la convolución puede ser unidimensional o de múltiples dimensiones.

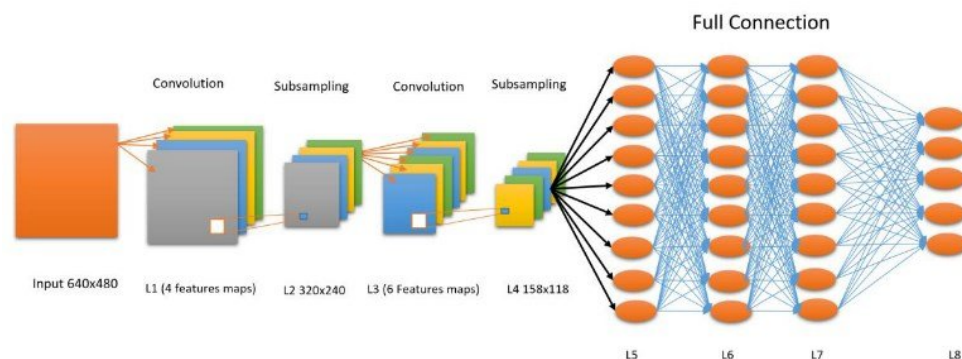


Figura 2.5: Arquitectura de una red neuronal convolucional [12].

2.2.2.1. Convolución casual

A la hora de describir como funciona la convolución casual en el modelado de secuencias se puede pensar en la forma que tiene una secuencia desde x_0, \dots, x_i , donde el objetivo es predecir las salidas correspondientes en cada paso t . Esto se podría indicar como una función $F : X^{(t+1)} \rightarrow Y^{(t+1)}$ donde para predecir y_i se necesitan los x_i anteriores, es decir, un filtro en el tiempo t solo puede ver entradas que no sean posteriores a t .

La entrada de la red convolucional sería una secuencia de x_0, \dots, x_i , y la salida será simplemente la capa de entrada desplazada en un tiempo $t + 1$. Una capa de convolución casual está formada por capas de la misma longitud donde lo único que hacemos es obtener una salida en un tiempo t con los elementos del tiempo t vistos y los anteriores. En la Figura 2.6 se puede observar esta estructura.

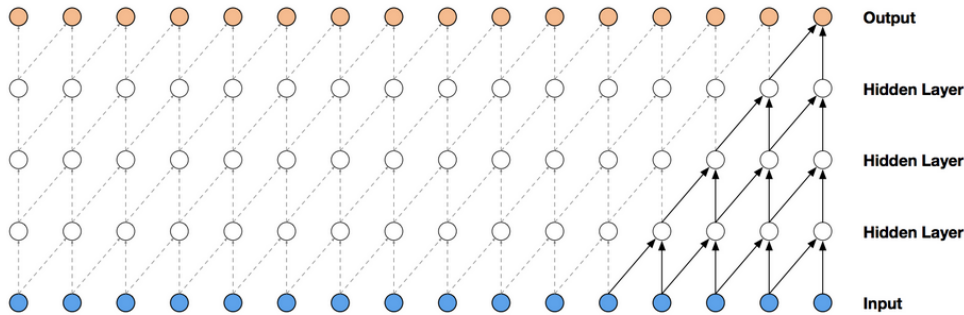


Figura 2.6: Pila de capas de convolución casual [13].

2.2.2.2. Convolución dilatada

La principal diferencia que tiene con respecto a la anterior operación es debido a un hiper-parámetro llamado dilatación, a medida que se incrementa este parámetro se van dejando espacios entre cada celda del núcleo y el resto, a los espacios que se forman se les llama dilataciones.

Normalmente cuando utilizamos un kernel sobre una matriz lo deslizamos sobre las secciones contiguas de la matriz, pero con la dilatación se dejan espacios entre estas secciones.

Por ejemplo supongamos que tenemos una matriz de 9×9 y usamos un kernel de 3×3 con una dilatación de 2 obtendremos una submatriz $m[0 : 2, 0 : 2]$, luego pasaría el kernel de nuevo y obtendría $m[0 : 2, 0 : 4]$.

Este proceso sigue la siguiente ecuación 2.18:

$$(u * D_h)(i) = \sum_j u(i - D_j)h(j) \quad (2.18)$$

donde D_h es la operación de convolución dilatada, u es una función discreta, i indica los espacios en la dilatación y h la función de kernel.

El objetivo principal de estas redes es obtener un campo receptivo mayor que con la anterior operación, con menos parámetros (el número de parámetros crece de forma logarítmica a medida que avanzamos en el número de capas) y menos capas.

Esta técnica suele utilizarse junto a bloques residuales, donde un bloque residual es la concatenación de dos o más capas convolucionales dilatadas juntas, los resultados de la primera capa se añaden

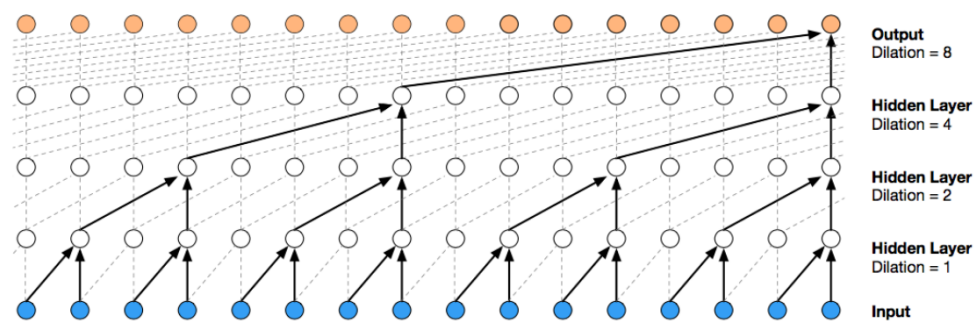


Figura 2.7: Pila de capas de convolución dilatada [13].

a la entrada de la siguiente capa y así con el resto de capas para obtener la salida del bloque entero. Hay que indicar que el número de canales y filtros de las capas deben ser igual, en caso de que no lo sean habría que aplicar una operación de convolución para que coincidieran las dimensiones.

2.2.3. Embeddings

Algo crucial a la hora de trabajar con los atributos que tendrá nuestra red es decidir la representación que tendrán; por ejemplo, los valores categóricos no pueden ser representados directamente y es necesario aplicar técnicas como *one-hot-encoder*, este tipo de técnicas pueden llegar a ser muy costosas o incluso inviables si existen muchos tipos distintos de categorías en un dominio.

Por este motivo se empiezan a utilizar otro tipo de representaciones mucho más eficientes y potentes, como son los embeddings. Mediante los embeddings se consigue representar los atributos categóricos en vectores compuestos de valores continuos pero en el que no es necesario tener vectores con una dimensión tan grande como atributos distintos existan, con esto lo que se consigue es tener un vector que actualizará sus pesos en una fase de entrenamiento para representar la palabra o elemento lo más fielmente posible.

La dimensión de estos vectores normalmente vendrá acotada por el número de valores distintos que tengan los datos pero en raras ocasiones esta será mayor de 250 o 300 suponiendo un verdadero avance en la representación de estos atributos.

Además, existen grandes embeddings pre-entrenados en el contexto del lenguaje del procesamiento natural (ó *Natural Language Processing (NLP)*), embeddings que han sido entrenados con millones de palabras de Wikipedia, Twitter u otros dominios y que pueden ser usados como entradas en otros modelos de redes neuronales. Esto hace que no sea necesario entrenar los pesos de la capa de embedding de un modelo neuronal y permite ahorrar tiempo de cómputo en la fase de entrenamiento.

Existen varios modelos famosos para la generación de embeddings en el ámbito NLP que pueden ser usados en un contexto distinto como son: Word2Vec [14] y fastText [15].

Word2Vec: este modelo tiene dos variantes: Bag of Words (ó *CBOW*) y Skip-gram. El primero de ellos utiliza las palabras de contexto para predecir la palabra que se encuentra en el medio y en el segundo caso se utiliza la palabra que está en el medio para predecir las palabras que se encuentran alrededor de la misma.

Ventajas: más rápido que el modelo fastText y se encuentra implementado en multitud de librerías.

Desventajas: no trabaja bien con palabras con escasa aparición en el texto. El modelo CBOW no tiene en cuenta el orden de las palabras relacionando las palabras más comunes a la hora de elegir las más cercanas.

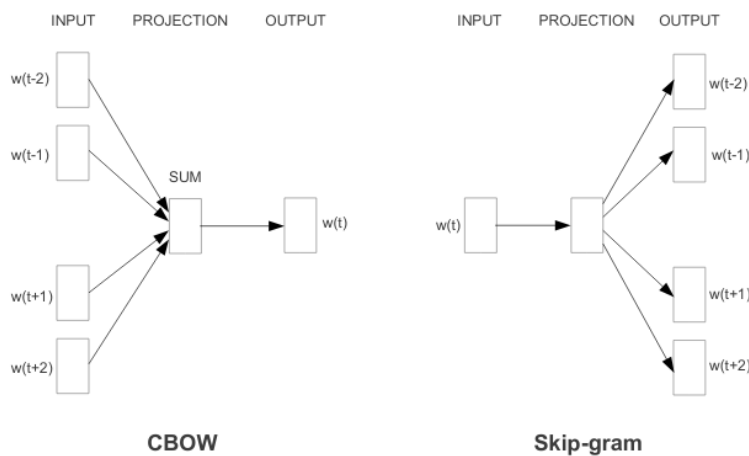


Figura 2.8: El modelo CBOW predice la palabra actual mediante el contexto que la rodea y promediando ese vector, mientras que Skip-gram predice las palabras que tiene alrededor dada la palabra actual. [14].

fastText: se considera una actualización de Word2Vec donde en lugar de aprender los vectores de palabras lo que hace es crear n-gramas de caracteres de las palabras, por ejemplo una palabra como experto con $n=3$, sería equivalente a $\langle ex, exp, xpe, per, ert, rto, to \rangle$. Esto permite capturar el significado de palabras más cortas o con falta de caracteres ya que permite que los embeddings sean capaces de comprender sufijos y prefijos. Una vez representado se puede usar modelos como Skip-gram o CBOW para aprender los embeddings.

Ventajas: permite aprender sobre palabras que no han aparecido en el entrenamiento gracias a los n-gramas. Funciona muy bien en textos con falta de información debido por ejemplo a la ausencia de caracteres.

Desventajas: es más lento debido a la cantidad extra de cálculos a la hora de entrenar la red, puesto que además de procesar la palabra también tiene que entrenar los n-gramas de esa palabra y promediar el resultado, todos estos

cálculos consumen más memoria que Word2Vec.

2.3. Sistemas de recomendación basados en secuencia

La recomendación secuencial se ha vuelto más popular gracias a la capacidad que tiene de no solo aprender preferencias de un usuario a largo plazo, sino también de ser capaz de analizar el momento actual en el que se realiza la recomendación o poder dar un peso distinto a las n últimas recomendaciones [16].

Ahora vamos a describir los métodos necesarios para procesar este tipo de datos, así como los algoritmos más importantes aplicados en el área, tanto clásicos (cadenas de Markov o aprendizaje por refuerzo) como los basados en redes neuronales, pero podemos adelantar que el uso de las redes neuronales en los sistemas de recomendación ha conseguido mejorar los resultados ya existentes en modelos basados en secuencias o sesiones como las cadenas de Markov (MC) [17].

2.3.1. Procesamiento de Datos

Al igual que en otras áreas como la clasificación de imágenes o de textos, cuantas más características y más datos se tengan para la red neuronal más robusto será nuestro modelo y menos problemas sufrirá de sobreajuste (ó *overfitting*). A continuación se describe el uso de embeddings y la técnica de aumento de datos (*del inglés, Data Augmentation*) para obtener modelos más efectivos:

Embeddings: el uso de embeddings nos permite representar características de elementos como el identificador de un usuario o el identificador de un ítem. Como en el caso de los autores de HRM [18] o en Meta-Prod2vec [19] donde utilizan la red Skip-Gram de Word2Vec para dos tareas: aprender la representación de las secuencias de los usuarios para relacionar los ítems más parecidos y además incluyen las características de los ítems como una ayuda auxiliar.

Aumento de datos: siguiendo el ejemplo de la clasificación de imágenes, hay ocasiones en la que tenemos pocas imágenes, están borrosas o demasiado dañadas para hacer un uso de ellas. En la recomendación secuencial pasa exactamente lo mismo, en muchas ocasiones no tenemos apenas información de un usuario porque es nuevo o porque falta información de la sesión debido a que un usuario elige no identificarse y realiza las interacciones con el sistema utilizando un perfil anónimo.

Por estos motivos es recomendable aumentar la cantidad de datos que tenemos utilizando los datos existentes y para ello podemos plantearnos las siguientes dos estrategias:

Subsecuencias: una estrategia sería la de, dada una secuencia $(l_1, l_2, l_3, l_4, l_5)$,

realizar pequeñas subsecuencias de la misma (l_1, l_2) , (l_1, l_2, l_3) , (l_1, l_2, l_3, l_4) llegando a tener hasta 4 secuencias en total, donde el objetivo de cada una de ellas será el último elemento de cada secuencia. Con esto logramos reducir el sobreajuste de la red y dar más peso a los valores intermedios [20], se puede entender como un proceso similar a obtener n-gramas, como explicamos anteriormente.

Añadir ruido: otra estrategia es la de añadir ruido a los datos de la red para lograr un modelo más robusto; dada la anterior secuencia $(l_1, l_2, l_3, l_4, l_5)$, podemos eliminar l_3 sustituyéndolo por un elemento neutro como 0 que representa la falta de información, quedando la secuencia $(l_1, l_2, 0, l_4, l_5)$. De esta forma lograremos que la red tenga una mayor tolerancia a fallos. Esta técnica se puede utilizar con longitudes fijas e ir añadiendo relleno a las posiciones de la secuencia que queramos eliminar o combinar con la anterior técnica para lograr subsecuencias más variadas.

2.3.2. Algoritmos

En esta sección se hace una breve descripción de los métodos más tradicionales y los basados en redes neuronales profundas enfocados en la recomendación secuencial.

2.3.2.1. Métodos tradicionales

Cadenas de Markov (ó Markov Chain (MC)): este tipo de modelos no tienen en cuenta largas dependencias entre usuarios. Normalmente solo se considera la última interacción o las n últimas interacciones más recientes, esto desaprovecha las dependencias de secuencias que sean extensas y no logra capturar toda la información de secuencias complejas.

Métodos basados en factorización: la factorización de matrices descompone la matriz de utilidad usuario-ítem en dos sub-matrices de menor dimensión; un usuario estará representado por un vector X_u en U y el ítem en un vector y_i en V , la multiplicación de estos vectores se asemejará (por definición) a la puntuación que le daría el usuario a ese ítem $r_{ui} = x_u^T \cdot y_i$. Existen muchos tipos de factorización, por ejemplo, BPR-MF [21] (ver ecuación 2.20 un poco más adelante) utiliza la función objetivo por descenso del gradiente estocástico (SGD) para clasificar pares usuario-ítem. Por otro lado, FPMC [22] combina la técnica de factorización de matrices y las cadenas de Markov. FOSSIL (*Factorised Sequential Prediction with Item Similarity Models*) [23] a su vez utiliza la combinación de las similitudes de ítems y cadenas de Markov de alto orden. Como en el caso anterior, estos modelos no son capaces de tener en cuenta largas dependencias y a no ser que utilicen algún tipo de información temporal suelen ignorar dependencias entre distintas sesiones.

Aprendizaje por refuerzo (ó *Reinforcement learning*): estos métodos se basan en la actualización de las recomendaciones basándose en las interacciones que tiene un usuario con el sistema de recomendación. Cuando un sistema recomienda un ítem a un usuario y a este le parece relevante le asigna una puntuación positiva, el objetivo del sistema es el de tratar de acumular la mayor cantidad de puntos positivos. Estos sistemas se adaptan dinámicamente pero no existen muchos frameworks capaces de llevar a cabo experimentos con cierta solidez.

2.3.2.2. Métodos basados en redes neuronales profundas

Redes neuronales recurrentes (RNN): las RNN actualmente cubren la mayor parte de los estudios relacionados con la recomendación basada en secuencias [24]. La principal diferencia con respecto a los métodos tradicionales es que las RNN son capaces de capturar las dependencias entre distintas sesiones o subsecuencias de un usuario, el punto más negativo de estos modelos lo encontramos en el coste computacional: a mayor longitud de la secuencia, mayor coste, siendo su fase de entrenamiento inviable si no se posee un entorno de altas prestaciones.

Redes neuronales convolucionales (CNN): las CNN se suelen utilizar en el procesamiento de señales [13] o para el análisis de imágenes. Son ideales para la captura de información local y permiten capturar características de la sesión utilizando las operaciones de *Max* y *Average Pooling*; además, si se utilizan variantes como las TCN (*Temporal Convolution Network*), son capaces de modelar información temporal. Como ejemplo, los autores de 3D-CNN [25] plantean un modelo 3D compuesto por los embeddings del ID del usuario e ID del ítem para aplicar las operaciones de convolución y pooling obteniendo buenos resultados y sin tener el coste de una red RNN.

Ejemplos de métodos basados en redes neuronales profundas

GRU4Rec [26]: este modelo basado en RNNs (sección 2.2.1) usando GRU (ver sección 2.2.1.2) se utiliza para la recomendación secuencial y no tiene en cuenta la identidad de un usuario, sólo distingue las secuencias de sesiones distintas mediante la diferencia entre tiempos. La entrada de la red es una sesión compuesta por uno o varios elementos en un rango determinado de tiempo. Las capas del modelo se componen de unidades GRU y la salida de la red es la probabilidad de que un elemento aparezca en el siguiente tiempo t . Además, GRU4Rec utiliza mini-batches paralelos para tener sesiones con la misma longitud, aunque esto puede llegar a ocasionar problemas puesto que las sesiones de los usuarios pueden variar mucho y tener duraciones distintas, y el uso ejemplos negativos basados en popularidad en la fase de entrenamiento.

CASER [27]: esta red utiliza un modelo basado en redes CNNs (sección 2.2.2) que visua-

liza la secuencia como si fuera una “imagen” a la que se le pasan filtros convolucionales horizontales y verticales para capturar los patrones secuenciales. Para poder capturar dependencias a largo plazo se utiliza el ID del usuario concatenado al final de la salida de las capas de convolución.

COSREC [28]: como en el caso anterior, Cosrec se basa en el uso de CNNs (sección 2.2.2) pero utilizando un modelo con dilatación y bloques residuales para la recomendación secuencial (ver sección 2.2.2.2). Este modelo consigue capturar dependencias a corto plazo con mejores resultados, así como dependencias a largo plazo.

En conclusión, si comparamos ambos métodos se puede ver que las redes neuronales profundas son capaces de aprender de secuencias extensas superando a modelos como MC y FPMC, obteniendo mejores resultados a la hora de recomendar un ítem relevante a un usuario y se adaptan mejor a los datos dispersos. No obstante, cuentan con los grandes problemas inherentes de las redes neuronales: la falta de explicabilidad del modelo obtenido (es decir, poder decir por qué un modelo neuronal es efectivo), el elevado coste del ajuste de parámetros de la red (que puede llegar a durar semanas) y la necesidad de tener grandes volúmenes de datos.

2.3.3. Estrategias generales de entrenamiento

Existen varias estrategias a la hora de entrenar nuestro modelo que pueden ayudar a mejorar el rendimiento de nuestro modelo neuronal en la fase de entrenamiento. A continuación se realiza una descripción de dos de las técnicas más importantes.

Muestreo negativo (ó *Negative sampling*): en los sistemas de recomendación uno de los métodos más comunes de muestreo es el basado en popularidad. Este método utiliza la frecuencia de aparición de un ítem: cuanto más popular sea ese ítem más probable es que un usuario conozca ese ítem. Si el usuario no ha visto el ítem aún siendo un ítem popular el método considerará que la probabilidad de que el usuario no le guste es mayor.

Función de pérdida (ó *Loss function*):

TOP1: esta función utiliza el muestreo tanto de ejemplos positivos como de ejemplos negativos y se divide en dos partes: la primera parte es la que penaliza los errores en la muestra positiva i y la muestra negativa j donde N_S indica el numero de muestras negativas, y la segunda parte que se utiliza como una regularización.

$$TOP1 = \frac{1}{N_S} \sum_{j=1}^{N_S} \sigma(r_j - r_i) + \sigma(r_j^2) \quad (2.19)$$

donde σ es una función sigmoideal, r_i y r_j son la puntuación en el ranking para

los ejemplos i y j .

BPR: este método es otro ejemplo de uso de muestro negativo, donde como en el caso anterior se calcula la función sigmoial sobre la diferencia entre la puntuación negativa y positiva.

$$BPR = -\frac{1}{N_S} \sum_{j=1}^{N_S} \log(\sigma(r_i - r_j)) \quad (2.20)$$

Entropía cruzada: además de las funciones de clasificaciones que hemos mencionado antes también existen funciones de pérdida de entropía cruzada:

$$ec(o, i) = \log(\text{softmax}(o)_i) \quad (2.21)$$

donde o es la salida del modelo e i es el elemento de test. El mayor problema de este método es el uso de una función *Softmax*, ya que cuanto mayor sea la cantidad de ítems a recomendar mayor será la complejidad del cálculo.

La función softmax es la encargada de asignar probabilidades a cada una de las posibles clases, estas probabilidades están normalizadas y la suma de todas ellas debe dar 1.0, gracias a esto la red tarda menos en converger.

$$\text{Softmax} = P(y = j|x) = \frac{e^{(w_j^T x + b_j)}}{\sum_{k \in K} e^{w_k^T x + b_k}} \quad (2.22)$$

siendo W^T los pesos de las conexiones, x la clase y b_j el sesgo.

2.4. Evaluación de Sistemas de Recomendación

A la hora de realizar la evaluación tenemos distintas métricas que estiman el nivel de acierto que ha tenido un sistema sobre un usuario; existen métricas que evalúan la diferencia entre la puntuación predicha por el sistema y la predicción real, y otras más actuales que se encargan de medir la posición en la que ha quedado un ítem en un ranking. A continuación se describen las métricas más importantes en estas estrategias:

Error absoluto medio (del inglés, Mean Absolute Error (MAE)): se calcula la distancia entre el valor real y el valor predicho para después aplicarle el valor absoluto al resultado de la operación.

$$MAE = \frac{1}{n} \sum_{j=1}^n |y_j - y_i| \quad (2.23)$$

Error cuadrático medio (del inglés, Root Mean Square Error (RMSE)): es parecido a MAE pero en vez de aplicar el valor absoluto a la diferencia entre el valor real y el predicho lo eleva al cuadrado.

$$RMSE = \sqrt{\frac{1}{|n|} \sum_{j=1}^n (y_j - y_i)^2} \quad (2.24)$$

Aunque MAE y RMSE se han convertido en métricas populares para medir cómo de bien se ha predicho una puntuación existen otras métricas que han ganado mucha fuerza en las últimas dos décadas. Estas métricas son la precisión (ver ecuación 2.25) y el recall (ver ecuación 2.26), que consideran si la recomendación de un ítem dentro de un ranking es relevante o no, por ejemplo, un ítem es relevante cuando aparece en la partición de test, sin embargo, si el ítem que aparece en test tiene un rating bajo se podría considerar que al usuario no le ha gustado y en ese caso podríamos considerar el ítem no relevante.

Siguiendo la tabla 2.1, se detallan las métricas más importantes a la hora de medir la relevancia, donde hay que tener en cuenta que estas métricas se suelen calcular hasta un *cutoff* (profundidad del ranking) N , es decir, el número de ítems recomendados serán menor o igual a dicho N :

Precisión: indica el ratio entre el número de elementos relevantes devueltos y el número de elementos totales obtenidos. La precisión final será la media de todos los patrones de entrada.

$$Precision = \frac{\#TP}{\#TP + \#FP} \quad (2.25)$$

Recall: indica el ratio entre el número de elementos relevantes devueltos y el número de elementos relevantes totales. El recall final será la media de todos los patrones de entrada.

$$Recall = \frac{\#TP}{\#TP + \#FN} \quad (2.26)$$

Recomendado	No recomendado
Positivo Verdadero (TP)	Falso Negativo (FN)
Falso Postivo (FP)	Negativo Verdadero (TN)

Tabla 2.1: Notación utilizada en las métricas de Precisión y Recall.

DISEÑO E IMPLEMENTACIÓN

En este capítulo se describe el trabajo de desarrollo llevado a cabo durante todo el proyecto, indicando los pasos seguidos a lo largo del mismo, librerías utilizadas, reglas usadas para filtrar los datos y, por último, una descripción del modelo.

3.1. Diseño

En esta sección se detallan los recursos utilizados en la elaboración del proyecto, las librerías externas utilizadas.

3.1.1. Diagrama de trabajo

En la siguiente figura 3.1 se puede ver el proceso llevado a cabo en este Trabajo.

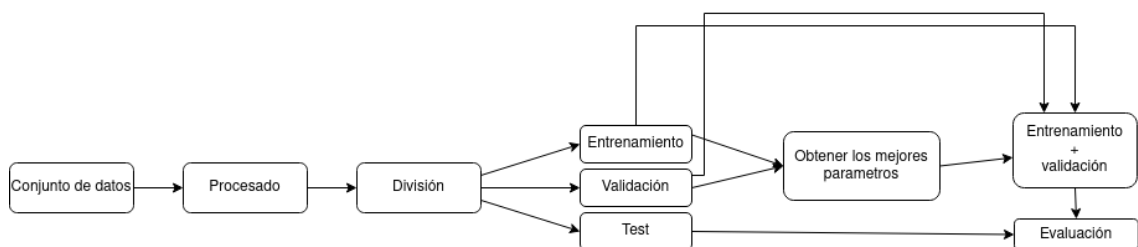


Figura 3.1: Metodología seguida durante el proyecto.

En un primer lugar se realiza el preprocesado de los datos, seleccionando los datos más relevantes de acuerdo a unos criterios específicos (ver sección 4.1.2), después de la limpieza de datos se obtienen nuevos datos a partir de los existentes en los conjuntos de datos.

Una vez se tienen los datos se llevan a cabo tres particiones: entrenamiento, validación y test, la partición de validación la usaremos para encontrar los parámetros óptimos de los baselines básicos, secuenciales y los modelos basados en redes, por último, en las pruebas reales usaremos la partición de entrenamiento y validación para entrenar el modelo y la de test para evaluarlo.

3.1.2. Descripción del entorno de trabajo

Los recursos utilizados en este proyecto son los que aparecen en la tabla 3.1. Como entorno auxiliar se ha utilizado la plataforma **Google colab**. Es muy fácil de configurar y permite utilizar recursos como tarjetas gráficas de gama alta o memoria RAM de más de 16 GB.

Recursos	Características
CPU	i7-8550U
GPU	GeForce 1070 ti 8GB GDRR5
S.O	Kubuntu 18.04
Versión python	2.7/3.6
RAM	32GB

Tabla 3.1: Equipo utilizado para el entrenamiento y pruebas.

Hemos utilizado la librería de **Conda** para configurar los entornos de cada baseline sin tener problemas de versiones o incompatibilidades entre sí.

El framework del modelo esta basado en **PyTorch**, aunque se empezó el desarrollo en Keras por su facilidad y su potencia (por ejemplo, con las redes recurrentes usando GPU). Sin embargo, existen algoritmos que aún no han sido implementados en este framework o no tienen un mantenimiento igual de frecuente que el encontrado en PyTorch o TensorFlow, esto es debido a que aunque Keras está creciendo rápidamente, aún no cuenta con una comunidad tan activa como las dos anteriores, menos aún, por lo que parece, en el área de los sistemas de recomendación.

3.1.3. Librerías externas utilizadas

Para el desarrollo de este trabajo se han utilizado varias librerías externas con el fin de ahorrar tiempo en la preparación y puesta a punto de los baselines y en el uso de un modelo basado en redes convolucionales dilatas (ver sección 2.2.2.2).

Caser¹: esta librería ha tenido dos funciones: la de actuar como baseline y la segunda de esqueleto para nuestro modelo utilizando la parte de predicción y particionado de datos.

Case Recommender² [29]: es una librería hecha en python que incluye una gran variedad de algoritmos de recomendación populares. El framework incluye algoritmos para la recomendación de ítems, la predicción de puntuaciones y clustering.

Fossil³: igual que en el caso de Caser, Fossil (sección 2.3.2) es un baseline en sí mismo, hemos utilizado la implementación de los algoritmos FMC y FPMC de esta librería.

¹https://github.com/graytowne/caser_pytorch

²<https://github.com/caserec/CaseRecommender>

³<https://sites.google.com/a/eng.ucsd.edu/ruining-he/>

TCN⁴ [30]: para el modelado de secuencias de usuario hemos utilizado un modelo que utiliza redes convolucionales dilatadas con bloques residuales (ver sección 2.2.2.2), se han realizado cambios en la librería para trabajar con las secuencias de un usuario y poder obtener como salida un vector que represente las características de esa secuencia; para lograr esto hemos eliminado las dos últimas capas del modelo que incluyen una capa flatten y una capa softmax (ver sección 2.3.3), concatenando directamente la salida con el vector de atributos.

3.2. Descripción del modelo

La principal función de este modelo neuronal es la de incorporar más atributos de los ítems incluidos en una secuencia para mejorar la recomendación. Al tener más información de una secuencia se podría mitigar problemas de los sistemas de recomendación como la falta de información o mejorar las recomendaciones en conjuntos de datos con más información a parte de las puntuaciones de los usuarios.

3.2.1. Arquitectura de la red neuronal

A la hora de capturar el comportamiento secuencial de un usuario y sus preferencias hemos utilizado tres módulos diferenciados en el modelo: el de secuencias, el de atributos y el de usuario. En la figura 3.2 se pueden apreciar estos módulos.

Secuencia: para modelar la secuencia del usuario hemos utilizado un modelo **TCN** al que le hemos aplicado una serie de modificaciones para predecir secuencias. Las principales ventajas del uso de las TCN sobre redes recurrentes son:

Paralelismo: en las redes RNN para obtener la salida de una unidad es necesario que las unidades anteriores hayan visto la secuencia, esto con la red TCN no es necesario puesto que el kernel que se utiliza es el mismo en todas las capas de la red y se puede realizar la operación de convolución en paralelo, tanto en la fase de entrenamiento como en la fase de test.

Flexibilidad en el tamaño del campo receptivo: la red tiene mayor flexibilidad a la hora de cambiar el tamaño del campo receptivo jugando con varios parámetros como: utilizar más capas convolucionales, aumentar o disminuir el hiper-parámetro de dilatación o aumentar el tamaño del filtro o núcleo.

Uso de memoria eficiente: las redes LSTM y GRU (ver sección 2.2.1) tienen un problema de coste a la hora de tratar con secuencias muy largas, esto se

⁴<https://github.com/locuslab/TCN>

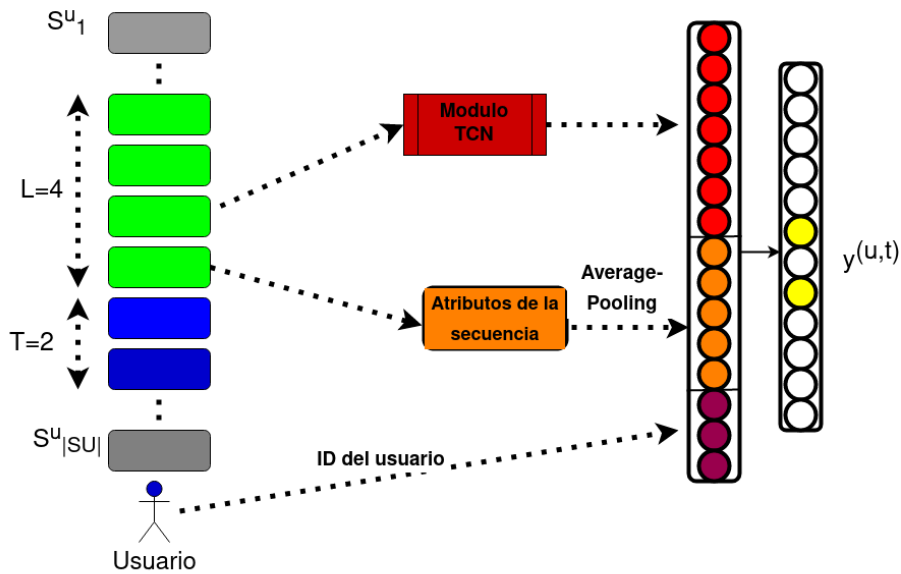


Figura 3.2: Arquitectura general de nuestro modelo.

debe a que necesitan guardar los estados de cada una de las unidades de memoria que utilizan y los resultados de estas. Sin embargo, en las redes TCN los filtros se comparten entre capas y el número de parámetros solo dependerá de la profundidad de las mismas.

Información local: además de poder capturar información temporal estas redes son capaces de capturar relaciones locales entre elementos de la secuencia.

Atributos: Para el modelado de los atributos se ha optado por elegir las técnicas convencionales de embedding descritas en la sección 2.2.3 y la inicialización de pesos mediante los modelos Word2Vec y fastText (ver sección 2.3.1).

Usuario: se utiliza el embedding del ID de usuario para lograr capturar dependencias a largo plazo de un usuario y concatenarlo a la salida de la red TCN y al vector concatenado de características obtenidas de los ítems de la secuencia.

Por último, la función de coste utilizada es la basada en ejemplos negativos **BPR** (sección 2.3.3).

3.2.2. Tratamiento de las características

Para representar las características de los elementos contenidos en una secuencia el uso de embedding es una tarea indispensable. A la hora de tratar estos embeddings dentro de la red existen varios métodos, por ejemplo, si tenemos un vector que representa la fecha de cuándo se hizo una reserva en un hotel y otro que indica el código postal de ese hotel podríamos realizar varias operaciones entre ellos dentro de una red neuronal, las dos más famosas son la **concatenación** y la **multiplica-**

ción de esos vectores. Estas operaciones no suelen tener mucha diferencia entre sí, aunque en la gran mayoría de artículos revisados suele producir mejores resultados la operación de concatenación [31].

La **operación concatenación** da buenos resultados en entornos donde trabajamos con pocas características y cuando los vectores concatenados tienen una dimensión baja, pero en caso de tener muchas características y de grandes dimensiones, como por ejemplo al usar imágenes, estos pueden llegar a tener un gran coste computacional a la hora de entrenar el modelo.

En el caso de la **operación de multiplicación** además de obtener peores resultados si trabajamos como en el caso anterior con grandes vectores, la multiplicación entre ambos puede ser muy costosa en la fase de entrenamiento.

Para evitar el problema de tener vectores de embedding demasiado grandes y poder capturar la mayor cantidad posible de información de los atributos generados, se ha optado por seguir una filosofía similar a la que utilizan las CNN con las imágenes, es decir, usamos la operación de max-pooling y average-pooling pero sin reducir su dimensión.

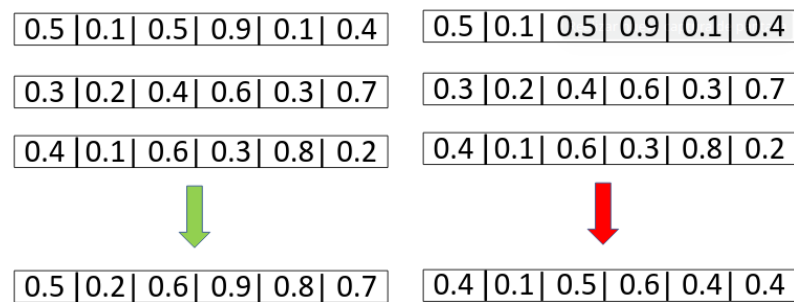


Figura 3.3: A la izquierda se puede ver el resultado de ejecutar la operación de max-pooling sobre tres vectores que representan tres características distintas de un ítem y a la derecha se pueden ver los mismos vectores pero ejecutando la operación de average-pooling.

A la hora de inicializar los pesos de las capas de embeddings (sección 2.2.3) se puede optar por varias opciones: inicializar todos a cero, inicializarlos de forma aleatoria o inicializarlos con otros valores de un modelo pre-entrenado. En este caso, una vez se tienen las subsecuencias de tamaño L y las particiones de entrenamiento/test hemos optado por inicializar los pesos con un modelo pre-entrenado. Hemos utilizado modelos ya existentes como Word2Vec y fastText, muy utilizados en el dominio del procesamiento del lenguaje natural. Para decidir cuál de estos dos modelos funciona mejor en nuestro contexto, se ha codificado un modelo propio basado en Prod2Vec [32] (ver figura 3.4) y se han realizado pruebas para comprobar el rendimiento de cada uno.

En la fase de pruebas para elegir una técnica u otra le pasamos al modelo modelo Prod2Vec únicamente la secuencia de ítems y se activan/desactivan las configuraciones de Word2Vec y fastText con los conjuntos de datos de Foursquare y Yelp (ver sección 4.1.1), para medir el rendimiento que tiene cada uno de estos en el modelo (los valores como el tamaño de ventana o el número de iteraciones

del modelo son los que utilizan por defecto los autores del artículo de Prod2Vec). Una vez hechas las pruebas se elige la técnica con mejores resultados y se desactiva la otra opción en la fase de evaluación final.

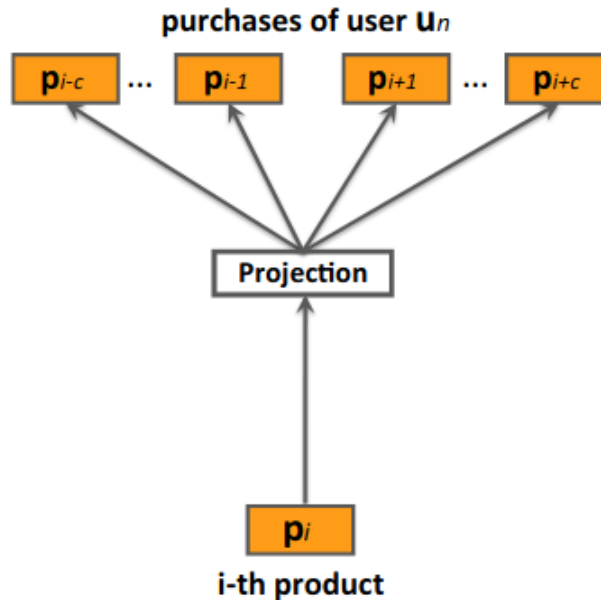


Figura 3.4: Modelo Skip-Gram de Word2Vec utilizado para Prod2Vec [32].

Una vez entrenado el modelo se le pasan los $n - 1$ POIs vistos por el usuario anteriormente, y se añade un factor f_i para dar un peso a las K recomendaciones hechas por el recomendador; de manera que, a medida que nos alejamos de los POIs más recientes, el peso que tengan esos POIs recomendados será menor.

Conjuntos de datos	Modelos	Prec@1	Prec@5	Prec@10	Recall@1	Recall@5	Recall@10
Foursquare	Word2Vec	0.0129	0.0111	0.0096	0.0129	0.0554	0.0759
	fastText	0.0264	0.0130	0.0088	0.0264	0.0549	0.0761
Yelp	Word2Vec	0.0129	0.0111	0.0096	0.0129	0.0554	0.0949
	fastText	0.0128	0.0105	0.0093	0.0128	0.0524	0.0921

Tabla 3.2: Resultados obtenidos con el modelo Word2Vec y fastText en ambos conjuntos de datos.

Si observamos la tabla 3.2 lo primero que vemos es que en un contexto distinto del procesamiento del lenguaje natural, las diferencias entre los dos métodos no parecen afectar en gran medida a los resultados. Para el conjunto de datos de Foursquare obtenemos una prec@5 más elevada en fastText que en Word2Vec así que en este caso elegiremos este método para pre-entrenar los vectores de atributos. Por otro lado, en el caso de Yelp con una diferencia más ajustada que en el caso ante-

rior utilizaremos Word2Vec aunque en este caso la precisión es muy similar, donde observamos más diferencias es con respecto a la métrica de recall.

Cabe mencionar que hemos podido trabajar con todos los atributos que queríamos de los conjuntos de datos salvo con uno: el texto de los comentarios hechos por un usuario. Una de las ideas iniciales que se abordó en este proyecto fue la de usar el texto de los comentarios o reseñas (*reviews*) para ayudar en la predicción de los siguiente POIs dada una secuencia, pero este atributo tiene una serie de complicaciones, por ejemplo, muchos usuarios no suelen tener más de dos o tres comentarios, con lo cual había una gran escasez de datos.

Otro problema con este tipo de atributo es cómo tratar los embeddings de un comentario puesto que ya no se trata de tener un embedding que represente una palabra sino a un conjunto de palabras: si se concatenaban los embeddings de las palabras podía tener un tamaño inviable; otra idea era hacer un promedio de las palabras contenidas en un comentario pero no se obtuvieron buenos resultados.

Por último, otra opción que se probó fue utilizar solo el último o los dos últimos (en este caso se concatenaban los vectores o se multiplicaban) comentarios para entrenar el modelo, todas estas estrategias produjeron peores resultados, por lo que se descartó el uso de los comentarios para el resto del proyecto, aunque se plantea como trabajo futuro cómo integrarlos de una manera que permita mejorar los resultados sin unos costes muy altos de tiempo y memoria.

EXPERIMENTOS Y RESULTADOS

4.1. Configuración del entorno

4.1.1. Conjuntos de datos

Para los experimentos hemos utilizados dos conjuntos de datos diferentes: Yelp y Foursquare, donde todas las interacciones poseen una marca de tiempo y tenemos más atributos aparte de la puntuación. Ambos conjuntos de datos pertenecen al dominio turístico, donde los ítems son lugares turísticos o puntos de interés (POIs). En el caso de Yelp contamos con una gran variedad de atributos sobre POIs, comentarios y usuarios. Por otro lado, en Foursquare nos encontramos con un grupo muy reducido de atributos que relacionan los lugares que ha visitado un usuario.

El conjunto de datos de Yelp ¹ cambia cada año, en nuestro caso hemos elegido la versión más actual ofrecida por la misma compañía para usos académicos, la de 2020. Para el conjunto de datos de Foursquare, nos hemos basado en unos datos hechos públicos ² por unos investigadores [33]; para ello, combinaron información de este sitio web con llamadas a Twitter, puesto que no es posible obtener información de los sitios visitados por los usuarios sólo usando la API de Foursquare o su página web. Debido a que estos datos son globales, hemos usado sólo aquellos pertenecientes a la ciudad de Nueva York ³.

Hemos aplicado una serie de filtros en ambos conjuntos de datos, eliminando aquellos datos con valores vacíos o erróneos. Además, en el caso de Yelp hemos eliminado las interacciones con una puntuación inferior a 4, tanto en los comentarios como en la puntuación de los POIs, por ejemplo, si tenemos una secuencia de ítems l_1, l_2, l_3, l_4, l_5 , y los ítems l_1 y l_3 tienen una puntuación inferior a 4, nos quedaríamos con la secuencia l_2, l_4, l_5 ; esta secuencia será la que después dividamos en la fase de entrenamiento y test, esto nos permitirá centrarnos solo en los ítems que le gusten a un usuario. Por otro lado, imponemos un número mínimo de interacciones en ambos conjuntos de datos de $K = 5$. Los resultados finales de los conjuntos de datos una vez filtrados se pueden ver en la tabla 4.1.

¹<https://www.yelp.com/dataset>

²<https://sites.google.com/site/yangdingqi/home/foursquare-dataset>

³https://www.dropbox.com/s/wxqx23pwy7c0ty4/US_NewYork__categories_coord.zip?dl=0

Conjuntos de datos	Usuarios	Ítems	Interacciones	Dispersión de los datos
Yelp	29.466	1.985	361.850	99,69 %
Foursquare	7.348	24.136	344.518	99,89 %

Tabla 4.1: Estadísticas de los datos utilizados en los experimentos.

4.1.2. Metodología experimental

Al utilizar conjuntos de datos con información temporal, hemos de aplicar una evaluación basada en marcas de tiempo. En primer lugar, lo que haremos será ordenar los conjuntos de datos por usuario y por tiempo (las primeras interacciones serán las más antiguas y las últimas las más recientes) para poder realizar una división de los datos según la dimensión temporal. Utilizaremos el 60 % de las interacciones más antiguas de un usuario para la fase de entrenamiento, el 20 % de datos posteriores para la partición de validación y, por último, el 20 % de los datos más recientes para test. La partición de validación se utilizará para evaluar los métodos entrenados con el conjunto de entrenamiento con el objetivo de encontrar los parámetros óptimos de cada algoritmo. En el informe final de resultados la partición de entrenamiento será la combinación de la partición de entrenamiento (60 %) y la de validación (20 %), usando la partición de test (por primera vez en todo este proceso) para evaluar los métodos utilizando los parámetros óptimos encontrados como se ha descrito previamente.

Generaremos un ranking de tamaño k , eliminando del ranking aquellos ítems recomendados por el modelo que ya han sido puntuados por el usuario en la fase de entrenamiento. Además, un ítem solo podrá ser recomendado si aparece al menos una vez en la fase de entrenamiento. Una vez obtenido el ranking se aplican las métricas de precisión y recall (ver sección 2.4), considerando relevantes todos los ítem que aparecen en el conjunto de test.

Para realizar una comparación equitativa de los resultados entre nuestro modelo y el resto de baselines se han realizado pequeñas modificaciones en los mismos para poder optimizar sus parámetros. Para cada baseline hemos utilizado los parámetros que se muestran en la tabla 4.2.

En la fase de optimización de parámetros de cada baseline hemos utilizado los conjuntos de datos de entrenamiento y validación. Es importante recordar que en la fase de optimización de parámetros solo se utilizan dos de las tres divisiones de los datos. Será en la evaluación final del proyecto donde se utilice en la fase de entrenamiento tanto la partición de entrenamiento como la de validación, y la de test para la evaluación final.

Siguiendo con la optimización de parámetros para la selección de los valores óptimos de cada modelo hemos elegido la métrica $Prec@5$, como hemos indicado en el párrafo anterior, estos resultados dependerán de los resultados obtenidos en el conjunto de datos de validación. En las tablas 4.3 y 4.4 se pueden ver los parámetros óptimos de cada baseline en los diferentes conjuntos de datos.

Tipo	Recomendador	Parámetros
Básico	POP	Ninguno
	BPRMF	$k = \{10, 50, 100\}$, $\lambda_u = \lambda_i = \{0.0005, 0.001, 0.0025, 0.005, 0.01, 0.1\}$, $\lambda_0 = \{0, 0.5, 1\}$, $\lambda_j = \{0.00005, 0.0001, 0.00025, 0.0005, 0.001, 0.01\}$
	ItemKNN	$sim = \{cosine, jaccard\}$, $k = \{40, 60, 80, 100, 120\}$
Secuencial	MC	$k=\{2, 5, 10, 20\}$, $\lambda=\{0.1, 0.2\}$
	FPMC	$k=\{2, 5, 10, 20\}$, $\lambda=\{0.1, 0.2\}$
	FOSSIL	$k=\{2, 5, 10, 20\}$, $\lambda=\{0.1, 0.2\}$, $L=\{1, 2, 3\}$
Redes neuronales	Caser	$T=\{1, 2\}$, $d=\{10, 50\}$, $nh=\{4, 16\}$, $L=\{4, 5\}$, $n_iters=\{30\}$, $l_rate=\{0.003\}$, $nv=\{4\}$, $drops=\{0.5\}$, $ac_convs=relu$, $ac_fcs=relu$, $batch_size=\{512\}$, $l2=\{10^{-6}\}$
	Cosrec	$T=\{1, 2\}$, $d=\{10, 50\}$, $fc_dim=\{100, 150, 200\}$, $L=\{4, 5\}$, $n_iters=\{30\}$, $l_rate=\{0.003\}$, $drops=\{0.5\}$, $batch_size=\{512\}$, $l2=\{10^{-6}\}$, $loss=\{bpr-max, cross-entropy, top1\}$, $final_act=\{elu-0.5, softmax, tanh\}$, $n_epochs=\{30\}$, $learning_rate=\{0.01\}$, $n_sample=\{1024, 2048\}$, $batch_size=\{512\}$
	GRU4Rec	$T=\{1, 2\}$, $d=\{10, 50\}$, $fc_dim=\{100, 150, 200\}$, $L=\{4, 5\}$, $n_iters=\{30\}$, $l_rate=\{0.003\}$, $drops=\{0.5\}$, $batch_size=\{512\}$, $l2=\{10^{-6}\}$, $loss=\{bpr-max, cross-entropy, top1\}$, $final_act=\{elu-0.5, softmax, tanh\}$, $n_epochs=\{30\}$, $learning_rate=\{0.01\}$, $n_sample=\{1024, 2048\}$, $batch_size=\{512\}$

Tabla 4.2: Parámetros optimizados de los recomendadores, entre {} los valores que se prueban para cada parámetro.

Tipo	Recomendador	Parámetros
Básico	POP	Ninguno
	BPRMF	$k = 100$, $\lambda_u = 0.01$, $\lambda_i = 0.0025$, $\lambda_0 = 0.5$, $\lambda_j = 0.001$
	ItemKNN	$sim = jaccard$, $k = 120$
Secuencial	MC	$k=20$, $\lambda=0.1$
	FPMC	$k=2$, $\lambda=0.1$
	FOSSIL	$k=20$, $\lambda=0.2$, $L=3$
Redes neuronales	Caser	$T=1$, $d=50$, $nh=16$, $L=4$, $n_iters=30$, $l_rate=0.003$, $nv=4$, $drops=0.5$, $ac_convs=relu$, $ac_fcs=relu$, $batch_size=512$, $l2=10^{-6}$
	Cosrec	$T=1$, $d=50$, $fc_dim=100$, $L=4$, $n_iters=30$, $l_rate=0.003$, $drops=0.5$, $batch_size=512$, $l2=10^{-6}$, $loss=bpr-max$, $final_act=elu-0.5$, $n_epochs=30$, $learning_rate=0.01$, $n_sample=2048$, $batch_size=512$
	GRU4Rec	$T=1$, $d=50$, $fc_dim=100$, $L=4$, $n_iters=30$, $l_rate=0.003$, $drops=0.5$, $batch_size=512$, $l2=10^{-6}$, $loss=bpr-max$, $final_act=elu-0.5$, $n_epochs=30$, $learning_rate=0.01$, $n_sample=2048$, $batch_size=512$

Tabla 4.3: Mejor configuración obtenida para cada recomendador para el conjunto de datos de Foursquare.

Tipo	Recomendador	Parámetros
Básico	POP	Ninguno
	BPRMF	$k = 100, \lambda_u = 0.005, \lambda_i = 0.01, \lambda_0 = 1, \lambda_j = 0.0005$
	ItemKNN	$sim = jaccard, k = 120$
Secuencial	MC	$k=10, \lambda=0.1$
	FPMC	$k=20, \lambda=0.1$
	FOSSIL	$k=10, \lambda=0.1, L=3$
Redes neuronales	Caser	$T=1, d=50, nh=16, L=4, n_iters=30, l_rate=0.003, nv=4, drops=0.5, ac_convs=relu, ac_fcs=relu, batch_size=512, l2=10^{-6}$
	Cosrec	$T=1, d=50, fc_dim=100, L=4, n_iters=30, l_rate=0.003, drops=0.5, batch_size=512, l2=10^{-6}$
	GRU4Rec	$loss = bpr-max, final_act=tanh, n_epochs=30, learning_rate=0.01, n_sample=2048, batch_size=512$

Tabla 4.4: Mejor configuración obtenida para cada recomendador para el conjunto de datos de Yelp.

4.1.3. Baselines

POP: recomendador no personalizado que basa sus recomendaciones en el número de interacciones de los ítems en la fase de entrenamiento. Para la implementación de este algoritmo hemos utilizado la librería CaseRecommender ⁴.

BPRMF: combinando este método con el modelo de factorización de matrices se logra un recomendador personalizado avanzado. Misma implementación que en el caso anterior.

ItemKNN: algoritmo de vecinos próximos basando en ítems, utiliza la similitud de los ítems para recomendar los más semejantes. Misma implementación que en el caso de BPR y POP.

MC: algoritmo de recomendación basado en cadenas de Markov de primer orden. Utilizamos la implementación hecha por los autores de Fossil ⁵ cambiando la fase de evaluación para obtener un ranking por cada usuario.

FPMC: utiliza la técnicas de factorización de matrices junto con cadenas de Markov. Misma implementación que en el caso anterior.

Fossil: se basa en el uso cadenas de Markov de alto orden y el uso de modelos de similitud para tratar las preferencias generales de un usuario. Misma implementación que en el caso de MC y FPMC.

Caser ⁶: modelo basado en redes neuronales convolucionales que utiliza filtros horizontales y verticales para capturar los patrones secuenciales de un usuario.

Cosrec ⁷: modelo basado en redes neuronales convolucionales que utiliza filtros 2D para

⁴<https://github.com/caserec/CaseRecommender>

⁵<https://drive.google.com/file/d/0B9Ck8jw-TZUEeEhSWXU2WWloc0k/view>

⁶<https://github.com/graytowne/caser>

⁷<https://github.com/zxxslp/CosRec>

capturar los patrones secuenciales.

GRU4Rec⁸: recomendador basado en sesiones que utiliza redes neuronales recurrentes para capturar las dependencias entre secuencias.

4.2. Foursquare

El conjunto de datos de Foursquare pertenece al dominio turístico y ofrece atributos como las coordenadas del POI o las fechas de cuándo se realizó el check-in de un usuario. En este caso utilizaremos las características de Foursquare (New York) para ver el rendimiento que tiene nuestro modelo cuando no existen una gran cantidad de atributos.

Además de los atributos ya incluidos, hemos generados dos nuevos atributos: el intervalo de tiempo transcurrido entre los POIs de la secuencia y la distancia entre cada POI de la interacción, esto ha sido posible gracias a los atributos ya contenidos en el conjunto de datos: las marcas de tiempo y las coordenadas. Para generar el primero de estos atributos hemos calculado la diferencia entre las marcas de tiempo y una vez hecho esto le hemos asignado un identificador único. Es importante indicar que hemos utilizado el mes, día y hora de la fecha descartando el resto de valores de la misma puesto que no tienen la misma relevancia y podían generar ruido; para el segundo atributo hemos utilizado la fórmula del semiverseno o distancia Haversine (ver ecuación 4.1).

$$\begin{aligned}
 \Delta\textit{latitud} &= \textit{latitud2} - \textit{latitud1} \\
 \Delta\textit{longitud} &= \textit{longitud2} - \textit{longitud1} \\
 a &= \sin^2(\Delta\textit{latitud}/2) + \cos(\textit{latitud1}) \cdot \cos(\textit{latitud2}) \cdot \sin^2(\Delta\textit{longitud}/2) \\
 c &= 2 \cdot \textit{atan2} \left(\sqrt{a}, \sqrt{1-a} \right) \\
 d &= R \cdot c
 \end{aligned} \tag{4.1}$$

donde R indica el radio de la tierra.

Una vez calculados los dos nuevos atributos tendremos un total de seis atributos a los que aplicaremos la técnica de embedding (ver sección 2.2.3):

Tipo 1-2-3: el tipo en Foursquare indica la taxonomía del lugar que está dividida en tres niveles, a mayor nivel mayor descripción del sitio; por ejemplo, el tipo 1 puede indicar que

⁸<https://github.com/hidasib/GRU4Rec>

el lugar es un restaurante mientras que el tipo 3 para ese mismo lugar podría indicar que es un restaurante de cocina tailandesa.

Fecha: momento en el que el usuario realizó el check-in.

Intervalo temporal: tiempo transcurrido entre dos check-ins.

Distancia geográfica: distancia entre dos lugares.

Tipo	Recomendador	Prec@1	Prec@5	Prec@10	Recall@1	Recall@5	Recall@10
Básico	POP	0.0885	0.0428	0.0304	0.0231	0.0570	0.0789
	BPR	0.0200	0.0149	0.0123	0.0058	0.0200	0.0339
	ItemKNN	0.0163	0.0112	0.0094	0.0059	0.0192	0.0313
Secuencial	MC	0.0522	0.0440	0.0347	0.0092	0.0442	0.0703
	FPMC	0.0492	0.0434	0.0361	0.0116	0.0477	0.0790
	FOSSIL	0.0502	0.0442	0.0324	0.0374	0.0428	0.0796
Redes neuronales	Caser	0.0637	0.0390	0.0281	0.0177	0.0516	0.0721
	Cosrec	0.0652	0.0412	0.0274	0.0175	0.0554	0.0714
	GRU4Rec	0.0402	0.0160	0.0100	0.0402	0.0801	0.1006
	Nuestro modelo	0.0889	0.0495	0.0372	0.0234	0.0648	0.0846

Tabla 4.5: Resultados de los baselines y de nuestro modelo con el conjunto de datos de Foursquare. En negrita, el mejor resultado para cada métrica.

En la tabla 4.5 mostramos los resultados obtenidos por los baselines y nuestro modelo con el conjunto de datos de Foursquare. A excepción de GRU4Rec en términos de recall, nuestro modelo superó al resto de baselines. Si nos fijamos en los métodos de referencia secuenciales (MC, FPMC y FOSSIL) se puede apreciar que son superiores de forma general a los baselines que no toman en cuenta la información secuencial salvo en el caso de POP, en este caso hay que destacar que dentro de la familia de algoritmos básicos da muy buenos resultados en este conjunto de datos, esto puede indicar que la mayoría de los usuarios que tenemos en nuestro conjunto de datos visitan los mismos lugares (los más famosos o populares) aunque existan una gran cantidad de ellos. Además, podemos observar que la personalización que hacen FPMC y FOSSIL en este caso no tiene unos resultados muy alejados de los obtenidos sin personalización por MC.

Por otro lado, si nos fijamos en los baselines que utilizan redes CNN, nuestro modelo obtiene resultados muy por encima de estos, y si nos fijamos en el modelo GRU4Rec que utiliza RNN podemos ver que en términos de precisión obtiene resultados muy pobres pero que en recall es capaz de superar a nuestro modelo.

4.2.1. Influencia de los atributos

Para medir el rendimiento de los atributos hemos probado a entrenar nuestro modelo con cada uno de ellos por separado (lo que se conoce como un *ablation analysis*). El modelo tendrá como entrada: el identificador de usuario, la secuencia de ítems y el atributo seleccionado en cada caso;

el identificador de usuario permite guardar un registro a largo plazo de las interacciones para cada usuario, la secuencia de POIs es la secuencia de interacciones del usuario l_1, l_2, \dots, l_N y el atributo seleccionado en cada caso que aportará información de cada POI de la secuencia.

Una vez hecha la división por tiempo de las secuencias utilizaremos el modelo que mejores resultados tenga (Word2Vec o fastText) (ver sección 3.2.2) con las secuencias de entrenamiento para detectar relaciones entre diferentes POIs (similar a la detección de similitudes entre palabras), una vez generados los vectores de embedding para cada POI guardaremos en estos los resultados para cargarlos posteriormente en la capa de embedding de nuestro modelo (*transfer learning*).

Además de medir la efectividad de los atributos hemos probado otras dos configuraciones en el modelo: en la primera el vector de embedding generado para la secuencia de ítem va concatenado directamente con el vector generado de atributos, es decir, no se utiliza la salida del módulo TCN (ver sección 3.1.3) y en la segunda configuración se utiliza el módulo TCN pero no se añade ningún atributo al modelo, por último, se utiliza la mejor configuración de atributos y se le pasa como entrada al módulo TCN la secuencia de ítems.

Tipo	Nombre	Prec@1	Prec@5	Prec@10	Recall@1	Recall@5	Recall@10
Atributos	Tipo 1	0.0830	0.0458	0.0312	0.0215	0.0598	0.0779
	Tipo 2	0.0836	0.0472	0.0320	0.0212	0.0626	0.0822
	Tipo 3	0.0859	0.0478	0.0320	0.0221	0.0628	0.0816
	Fecha	0.0851	0.0452	0.0315	0.0219	0.0588	0.0799
	Intervalo temporal	0.0843	0.0433	0.0304	0.0227	0.0554	0.0766
	Distancia geográfica	0.0757	0.0404	0.0287	0.0202	0.0523	0.0718
Modelo	Sin Módulo TCN	0.0739	0.0441	0.0309	0.0197	0.0587	0.0788
	Sin Atributos	0.0759	0.0456	0.0316	0.0202	0.0603	0.0804
	Mejor modelo (MAX)	0.0808	0.0425	0.0293	0.0218	0.0548	0.0727
	Mejor modelo (AVG)	0.0889	0.0495	0.0372	0.0234	0.0648	0.0846

Tabla 4.6: Utilidad de la información aportada por los distintos atributos. En negrita, el mejor resultado por cada bloque, y subrayado el mejor resultado para esa métrica.

Como se puede ver en la tabla 4.6, los tipos 1-2-3 tienen valores muy similares pero se puede apreciar que a mayor aporte de información (tipo 3) mejores son los resultados. En concreto, generalizar e indicar información simple como si el POI es un bar o un restaurante aporta buenos resultados (hay atributos con peores resultados en general) pero indicar más detalles sobre el ítem parece que sirve para aprender mejor y, por lo tanto, termina obteniendo mejores resultados. Otro atributo que da muy buenos resultados es la fecha de cuándo se hizo el check-in, superando al tipo 1; esto parece indicar que la información de cuándo un usuario visita un lugar es un elemento importante a tener en cuenta. En el caso de los intervalos vemos cómo parecen ser de los atributos con peores resultados, ya que los valores obtenidos con la diferencia temporal entre un lugar de interés y otro son peores en comparación con los citados anteriormente, aunque aporta información que nos podría ser útil; sin embargo, no es el caso de la distancia geográfica que, en este conjunto de datos, aporta resultados por debajo de la media.

Por último, si analizamos la importancia que tiene el módulo TCN y los atributos podemos ver que cuando no utilizamos TCN con la secuencia de ítems los resultados son peores aún utilizando la mejor combinación de atributos, esto indica que las redes convolucionales con dilatación parecen funcionar en secuencias y son componentes a tener cuenta. En el caso de eliminar los atributos y contar con el módulo TCN se obtienen peores resultados en términos de precisión pero no de recall.

Si aplicamos las operaciones de max-pooling (Mejor modelo (MAX)) y average-pooling (Mejor modelo (AVG)) sobre las distintas configuraciones de los atributos del conjunto de datos de Foursquare obtenemos los mejores resultados con la operación de average-pooling y utilizando todos los atributos salvo la distancia geográfica entre POIs puesto que este atributo al hacer la media empeora los resultados.

4.2.2. Análisis de hiper-parámetros

A continuación analizamos los siguientes hiper-parámetros: dimensión del embedding, número de iteraciones, número de ejemplos negativos y longitud de la secuencia con respecto a la precisión@5.

Para analizar el rendimiento de estos parámetros hemos fijado los parámetros óptimos y solo hemos variado el valor del parámetro a analizar. Como podemos ver en la figura 4.1(a), una mayor dimensión en los vectores de embedding no es un indicativo de mejores resultados, en este caso la dimensión ideal es $D = 50$ y el resto de valores tienen resultados similares.

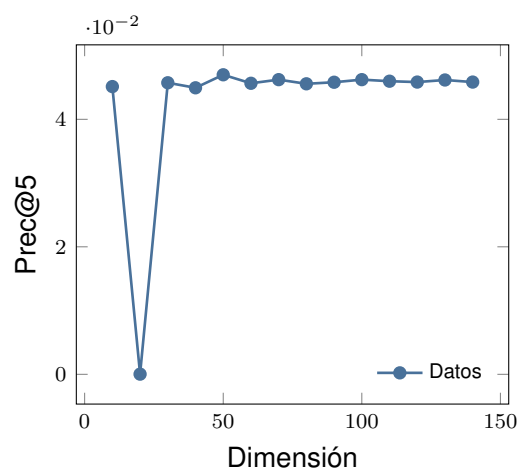
En la figura 4.1(b) vemos que el modelo obtiene su mejor rendimiento en la iteración $I = 2$, una buena noticia si lo que buscamos es celeridad en el entrenamiento.

Si nos fijamos en la figura 4.1(c), el número de ejemplos negativos óptimo en este caso es $N = 3$ y, como en el caso de la dimensión, el aumento de ejemplos negativos en entrenamiento no garantiza la mejora de resultados, aunque por otro lado sí se observa que siempre es mejor el uso de ejemplo negativos a la hora de realizar el entrenamiento,

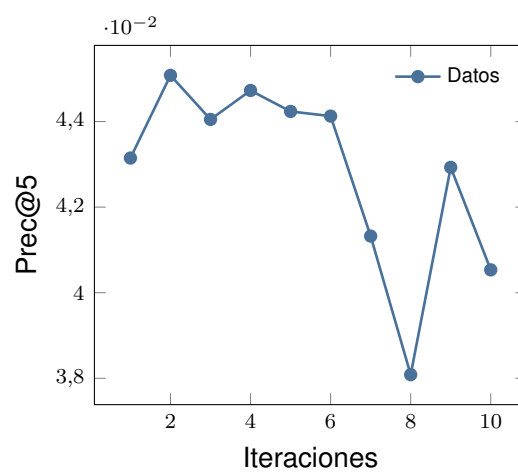
Por último, en la figura 4.1(d) se aprecia que las longitudes cortas $L = 1, 2, 3$ no tienen muy buenos resultados puesto que el modelo aprende peor, los mejores resultados se obtiene con longitudes entorno a $L = 4, 5$ ítems.

4.3. Yelp

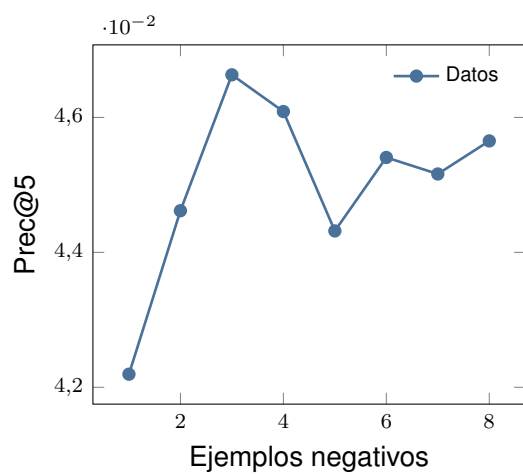
Yelp es un conjunto de datos que, al igual que en caso anterior, pertenece al ámbito turístico pero, que al contrario de Foursquare, posee una gran cantidad de atributos; por ejemplo, podemos encontrar información de los comentarios hechos por un usuario o información más específica de los POIs. El objetivo de usar Yelp es ver cómo se comporta nuestro modelo cuando existen una gran variedad de



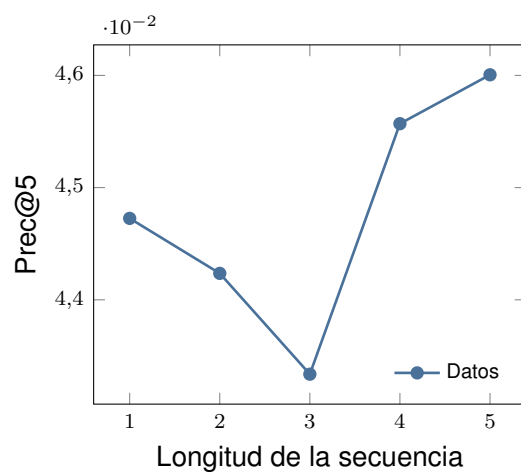
(a) Dimensión



(b) Iteraciones



(c) Ejemplos negativos



(d) Longitud

Figura 4.1: Estudio de la sensibilidad de los hiperparámetros con el conjunto de datos de Foursquare.

atributos; no obstante, al igual que en Foursquare, hemos generado dos nuevos atributos a partir de las marcas de tiempo y coordenadas (ver ecuación 4.1).

Una vez calculados los dos nuevos atributos tendremos un total de diez, igual que con Foursquare aplicaremos la técnica de embedding (ver sección 2.2.3) para todos ellos:

Utilidad: el número de usuarios a los que un comentario les ha parecido útil.

Gracioso: el número de usuarios a los que un comentario les ha parecido gracioso.

Interesante: el número de usuarios a los que un comentario les ha parecido interesante.

Fecha: el momento en el que se realizó el comentario.

Dirección: la localización del punto de interés.

Ciudad: nombre del lugar de la ciudad donde está el POI.

Estado: territorio al pertenece el lugar de interés.

Código postal: código que identifica un lugar del país.

Intervalo temporal: tiempo transcurrido entre dos check-ins.

Distancia geográfica: distancia entre dos lugares.

Tipo	Recomendador	Prec@1	Prec@5	Prec@10	Recall@1	Recall@5	Recall@10
Básico	POP	0.0044	0.0049	0.0046	0.0020	0.0113	0.0207
	BPR	0.0225	0.0185	0.0164	0.0100	0.0418	0.0737
	ItemKNN	0.0315	0.0247	0.0204	0.0149	0.0541	0.0940
Secuencial	MC	0.0256	0.0204	0.0180	0.0098	0.0396	0.0691
	FPMC	0.0306	0.0256	0.0227	0.0120	0.0501	0.0874
	FOSSIL	0.0294	0.0242	0.0211	0.0110	0.0466	0.0812
Redes neuronales	Caser	0.0311	0.0260	0.0235	0.0119	0.0491	0.0874
	Cosrec	0.0316	0.0273	0.0239	0.0117	0.0504	0.0880
	GRU4Rec	0.0113	0.0087	0.0075	0.0113	0.0435	0.0758
	Nuestro modelo	0.0386	0.0315	0.0275	0.0149	0.0597	0.1014

Tabla 4.7: Resultados de los baselines y de nuestro modelo con el conjunto de datos de Yelp. En negrita, el mejor resultado para cada métrica.

Como podemos observar en la tabla 4.7, nuestro modelo superó al resto de baselines tanto en términos de precisión como de recall. Esto es debido a que contamos con una gran cantidad de atributos que podemos usar para mejorar la predicción de nuestro modelo. Si hablamos de los baselines básicos como POP y BPR vemos que su rendimiento es muy pobre, en cambio ItemKNN obtiene unos resultados muy buenos, sobre todo en términos de recall. En el caso de los modelos que toman en cuenta la información secuencial, tanto FPMC como FOSSIL superan a MC lo que indica en este caso la importancia de la personalización. En los modelos neuronales podemos ver cómo Cosrec es ligeramente superior a Caser y GRU4Rec obtiene resultados muy pobres en términos de precisión.

4.3.1. Influencia de los atributos

Tipo	Nombre	Prec@1	Prec@5	Prec@10	Recall@1	Recall@5	Recall@10
Comentarios	Utilidad	0.0328	0.0264	0.0231	0.0129	0.0508	0.0877
	Gracioso	0.0325	0.0264	0.0232	0.0125	0.0502	0.0875
	Interesante	0.0316	0.0269	0.0237	0.0121	0.0511	0.0890
	Fecha	0.0332	0.0271	0.0240	0.0128	0.0511	0.0911
	<i>Fusión</i>	0.0348	0.0295	0.0259	0.0132	0.0551	0.0961
Geografía	Dirección	0.0339	0.0276	0.0250	0.0134	0.0528	0.0938
	Ciudad	0.0343	0.0283	0.0249	0.0129	0.0535	0.0939
	Estado	0.0336	0.0270	0.0239	0.0129	0.0513	0.0903
	Código postal	0.0358	0.0294	0.0255	0.0137	0.0565	0.0952
	<i>Fusión</i>	0.0360	0.0290	0.0255	0.0139	0.0552	0.0981
Intervalos	Intervalo temporal	0.0304	0.0251	0.0221	0.0114	0.0473	0.0824
	Distancia geográfica	0.0287	0.0238	0.0213	0.0107	0.0447	0.0795
Modelo	Sin Módulo TCN	0.0345	0.0265	0.0231	0.0132	0.0500	0.0867
	Sin Atributos	0.0335	0.0283	0.0249	0.0128	0.0536	0.0933
	Mejor modelo (MAX)	0.0340	0.0283	0.0251	0.0132	0.0542	0.0942
	Mejor modelo (AVG)	0.0386	0.0315	0.0275	0.0149	0.0597	0.1014

Tabla 4.8: Utilidad de la información aportada por los distintos atributos. Mejor resultado en cada bloque, en negrita. Nótese que se ha incluido una fila extra donde se han fusionado todos los atributos de ese bloque para determinar la influencia de dicho bloque (denotado como *Fusión*). El mejor resultado por cada métrica se indica con subrayado.

Al igual que hicimos con Foursquare, en esta sección analizamos la influencia de cada uno de los atributos, así como de dos variantes del modelo neuronal en sí. Observamos en la tabla 4.8 que existen dos grandes bloques de atributos que nos aportan información sobre los POIs que visita un usuario: comentarios y geografía; si nos fijamos en el cómputo general de estos bloques por separado los atributos geográficos parecen tener mejores resultados que la información aportada por los comentarios, aunque si fusionamos los atributos de cada bloque con la operación de average-pooling y comprobamos la efectividad de cada uno de ellos (indicado en la columna *Fusión*), vemos que los resultados no son muy distintos entre ambos bloques.

Dentro de cada bloque podemos ver cómo en los comentarios el atributo más importante es la fecha de cuándo se ha realizado el comentario sobre el ítem, seguido de si el comentario le pareció interesante a otros usuarios; en el caso de los atributos geográficos el más útil parece ser el código postal. Podríamos derivar de este resultado que los usuarios que suelen viajar a un POI también visitan otras zonas de interés cercanas a este dentro del mismo código postal, esta teoría gana fuerza si nos fijamos en el resto de atributo como la ciudad o la dirección. El peor de los atributos lo encontramos en el estado donde está el POI, esto puede ser debido a que es un atributo demasiado general y no permite personalizar ni concretar de manera suficiente. Si vemos los intervalos son los atributos más débiles en comparación con el resto, aunque igual que en el otro conjunto de datos, el mejor es el intervalo temporal.

Por otro lado, observamos que si eliminamos el módulo TCN obtenemos un rendimiento bastante pobre, lo que indica que aunque tengamos muchos atributos útiles a la hora de entrenar, el valor más importante sigue siendo el tratamiento de la secuencia de ítems; si añadimos esta secuencia como un simple embedding (sin módulo TCN) el rendimiento que obtenemos es mucho menor.

Por último, si quitamos los atributos al modelo y dejamos el módulo TCN se obtienen mejores resultados reafirmando la teoría anterior.

Finalmente, la mejor configuración la obtenemos seleccionando todos los atributos del bloque de comentarios, geografía y el intervalo temporal, dejando fuera el intervalo geográfico, además, al aplicar la operación de average-pooling obtenemos los mejores resultados.

4.3.2. Análisis de hiper-parámetros

Igual que con Foursquare, a continuación analizamos los siguientes hiper-parámetros: dimensión del embedding, número de iteraciones, número de ejemplos negativos y longitud de la secuencia con respecto a la precisión@5.

Al observar la figura 4.2(a), podemos ver cómo la dimensión alcanza su rendimiento ideal con $D = 40$ y que, a medida que aumentamos el tamaño del vector, se aprecia cómo la tendencia se vuelve negativa. Esto se debe a que un modelo obtiene su mejor rendimiento con una D correcta y a mayor tamaño de D mayor sobreaprendizaje del modelo. Por otro lado, en la figura 4.2(b) podemos ver cómo en la iteración $I = 7$ obtenemos los mejores resultados.

Además, según la figura 4.2(c) el número de ejemplos negativos óptimo estaría en los $N = 8$ ejemplos. Por último, en la figura 4.2(d), se observa que salvo en el caso de $L = 1$, el resto de casos obtienen resultados muy similares; en este caso $L = 5$ es el mejor valor para la longitud.

4.4. Discusión de los resultados

Si comparamos los resultados obtenidos con los dos conjuntos de datos podemos ver que, a mayor cantidad de atributos, mejores son los resultados obtenidos siempre y cuando se realicen las operaciones correctas como max-pooling o average-pooling y se utilicen vectores pre-entrenados para la inicialización de pesos en las capas de embedding de la red. Existen atributos que parecen aportar más información que otros como los atributos geográficos o la fecha de cuándo se realizó un check-in o un comentario, en cambio, los atributos que utilizan el contexto entre dos POIs como los intervalos parecen no dar muy buenos resultados comparados con el resto.

Por otro lado, podemos ver que los resultados con los algoritmos varían si cambiamos de un conjunto de datos a otro, aunque normalmente se cumple que los algoritmos secuenciales son mejores

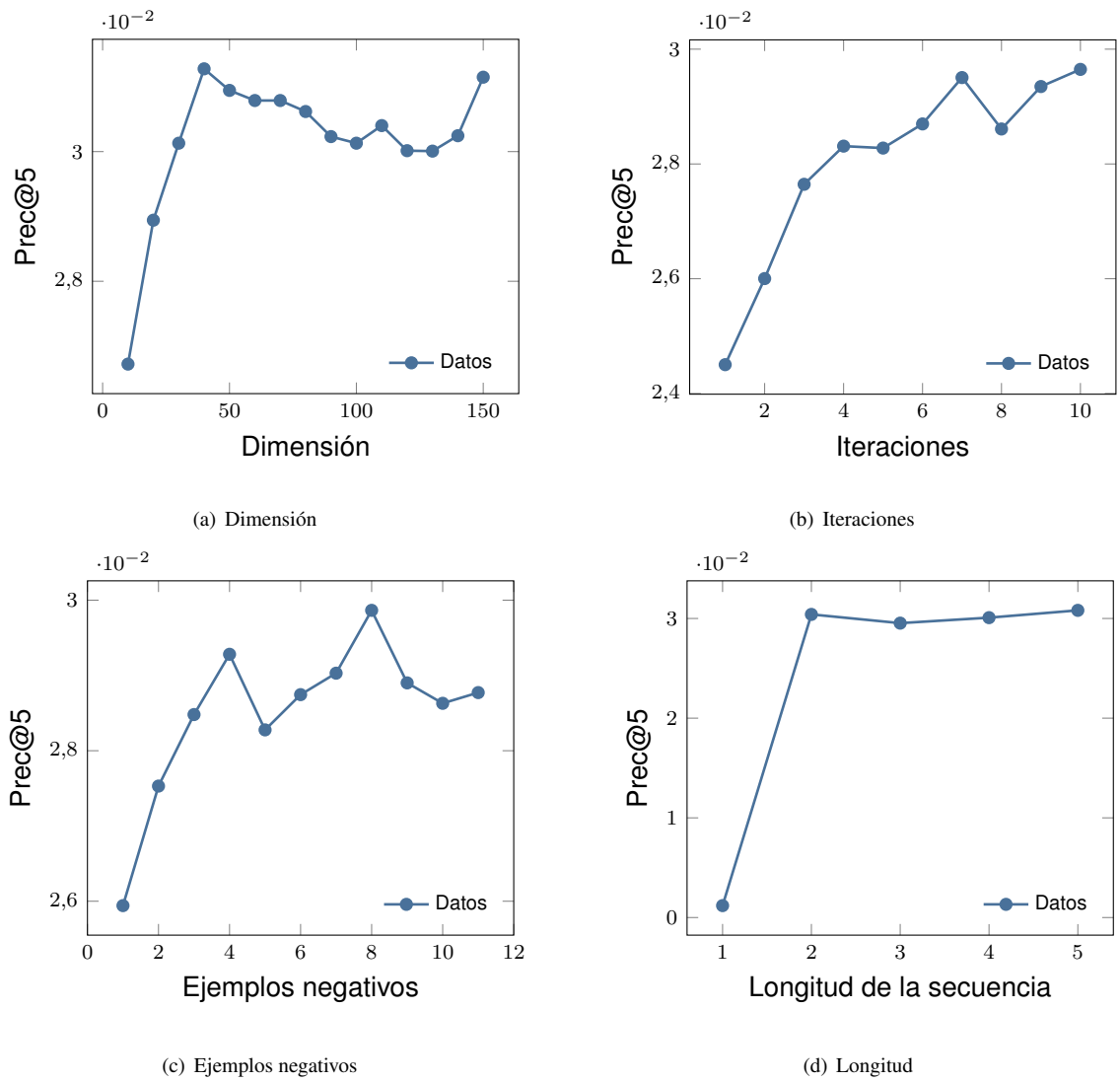


Figura 4.2: Estudio de la sensibilidad de los parámetros con el conjunto de datos de Yelp.

que los básicos, existen casos como POP en Foursquare o ItemKNN en Yelp que pueden dar muy buenos resultados y no se deben descartar o ignorar estas opciones (algo que tiende a ocurrir en algunos de los artículos estudiados). Con los algoritmos secuenciales también observamos que, por lo general, los algoritmos con personalización suelen ser mejor que sin personalización en la mayoría de los casos, pero pueden darse excepciones.

En el caso de las redes neuronales vemos cómo redes tipo CNN (como Caser y Cosrec) obtienen resultados muy similares, siendo este último modelo superior en ambos conjuntos de datos, mientras que la red RNN (GRU4Rec) en el caso de Yelp obtiene resultados peores, aunque en Foursquare obtiene resultados mejores que nuestro modelo en términos de recall.

Otro punto muy importante a destacar es la utilización de los hiper-parámetros correctos de los modelos, como la dimensión de los vectores de embedding para evitar sobre-aprendizaje o el número de ejemplos negativos. Por lo tanto, siempre que se utilicen este tipo de modelos es necesario hacer un barrido de los parámetros para optimizar la red.

CONCLUSIONES Y TRABAJO FUTURO

5.1. Conclusión

En este trabajo, hemos analizado el problema de la recomendación personalizada, en concreto en el dominio turístico; para ello, hemos propuesto un modelo basado en red neuronal que es capaz de aprender los comportamientos secuenciales de un usuario. Para lograr esto hemos diseñado una red neuronal que trabaja con los atributos de los lugares de interés y que es capaz de tener memoria a largo plazo utilizando el identificador de un usuario y que tiene una fase rápida de entrenamiento gracias al uso de de redes neuronales convolucionales. Para medir estos resultados hemos utilizado dos conjuntos de datos públicos distintos como son Foursquare y Yelp de dominio turístico. Gracias a estos dos conjuntos de datos hemos podido ver el comportamiento que tiene nuestro modelo en un escenario con escasez de atributos y otro con abundancia de ellos.

Hemos visto cómo nuestro modelo es capaz de aprender patrones secuenciales de forma eficiente y de modelar los factores latentes mediante embeddings obteniendo resultados superiores a otras redes que también utilizan redes convolucionales o redes neuronales recurrentes, siendo un método mucho más rápido que estas. Es importante mencionar que no todos los atributos pueden ayudarnos a mejorar los resultados y que existen atributos más importantes que otros en el dominio turístico, normalmente los atributos geográficos son mejores que el resto, lo que puede indicar que un usuario visita lugares cercanos entre sí y que hay lugares que solo se visitan en fechas concretas, siendo la fecha otro atributo muy importante.

Como es de esperar, no todos los modelos funcionan igual y hemos podido ver que, dependiendo del conjunto de datos con el que trabajemos, se obtienen resultados algo distintos, aunque, en general, los modelos secuenciales suelen ser superiores a los modelos básicos y los modelos más avanzados, como las redes neuronales, suelen tener mejores resultados que los baselines secuenciales.

Otro punto a tener en cuenta es el uso de modelos pre-entrenados para la inicialización de los vectores de embedding, esta idea trasladada del contexto de las imágenes o del procesamiento del lenguaje natural obtiene en la mayoría de ocasiones mejores resultados y es algo que no suele costar mucho tiempo de cómputo, puesto que una vez que se tiene el vector entrenado solo queda cargar los

datos en el modelo.

Para acabar, es importante tener en cuenta qué tipo de operación es la más eficaz para extraer las características de un ítem, si la media o el máximo de un vector, para ello siempre habrá que probar estas opciones porque, como ocurre con los atributos, esto puede variar de un conjunto de datos a otro, al igual que la optimización de los parámetros de la red.

Por último, todo el código y los scripts desarrollados se encuentran disponibles en el siguiente repositorio: <https://bitbucket.org/Rsanchezguzman/tesnet/>.

El autor de este trabajo fin de máster ha sido beneficiario de la ayuda para el fomento de la Investigación en Estudios de Máster-UAM 2019/2020 de la Universidad Autónoma de Madrid.

5.2. Trabajo Futuro

En un futuro, podríamos ampliar estos experimentos y extenderlos a otros conjuntos de datos de distintos dominios, como películas o música y ver cuáles son las características más determinantes a la hora de trabajar con ellos. Por otro lado, podríamos utilizar otro tipo de redes neuronales como los autoencoders o redes que implementen mecanismos de atención para ver el rendimiento de este tipo de redes a la hora de capturar patrones secuenciales.

Como hemos comentado anteriormente, sería interesante ver qué resultados se obtendrían si utilizáramos el texto de los comentarios hechos por un usuario para entrenar el modelo neuronal. Otro atributo interesante sería utilizar las imágenes de un POI visitado por un usuario, creemos que este tipo de atributos normalmente suelen funcionar bien aunque el coste computacional aumente debido al entrenamiento con imágenes; no obstante, ya hay algunos trabajos en el área que apuntan a que tendría un impacto positivo utilizar esta información [34].

Existen trabajos donde se utilizan las salidas de distintos algoritmos de recomendación para hacer re-ranking y obtener una salida mixta, otra posible vía a estudiar en el futuro sería la de combinar la salida de algoritmos básicos, secuenciales y basados en redes neuronales para hacer re-ranking de sus salidas, asignando una ponderación individual a cada uno de ellos para luego combinarlos en un ranking final que sea capaz de mejorar los resultados obtenidos en este trabajo.

También cabe la posibilidad de utilizar atributos de un dominio rico en información secuencial para entrenar el modelo neuronal y aplicarlo en un dominio con escasa información pero donde el usuario y la temática sea la misma; es decir, en nuestro caso por ejemplo podríamos entrenar un modelo con el conjunto de datos de Yelp y aplicarlo en la recomendación de ítems de Foursquare, siempre que sepamos que existe una identificación entre usuarios e ítems en ambos dominios. Esto permitiría solucionar problemas como el arranque en frío (*cold start*) [1] y poder realizar recomendación personalizada de mayor calidad.

BIBLIOGRAFÍA

- [1] B. Lika, K. Kolomvatsos, and S. Hadjiefthymiades, "Facing the cold start problem in recommender systems," *Expert Syst. Appl.*, vol. 41, pp. 2065–2073, Mar. 2014.
- [2] B. Chang, Y. Park, S. Kim, and J. Kang, "Deeppim: A deep neural point-of-interest imputation model," *Information Sciences*, vol. 465, pp. 61–71, Oct. 2018.
- [3] B. Chang, Y. Park, D. Park, S. Kim, and J. Kang, "Content-aware hierarchical point-of-interest embedding model for successive poi recommendation," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI'18*, p. 3301–3307, AAAI Press, 2018.
- [4] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, *Recommender Systems Handbook*. Berlin, Heidelberg: Springer-Verlag, 1st ed., 2010.
- [5] 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.
- [6] L. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. USA: Prentice-Hall, Inc., 1994.
- [7] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol. abs/1506.00019, 2015.
- [8] S. F. Felix Gers, Fred Cummins, "Understanding lstm networks." <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>, 2015.
- [9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.
- [10] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.
- [11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.
- [12] S. Contreras and F. De la Rosa, "Aplicación de deep learning en robótica móvil para exploración y reconocimiento de objetos basados en imágenes," in *Aplicación de Deep Learning en Robótica Móvil para Exploración y Reconocimiento de Objetos basados en Imágenes*, 09 2016.
- [13] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," 2016.
- [14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.
- [15] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *arXiv preprint arXiv:1607.04606*, 2016.

- [16] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system," *ACM Computing Surveys*, vol. 52, p. 1–38, Feb 2019.
- [17] H. Fang, G. Guo, D. Zhang, and Y. Shu, "Deep learning-based sequential recommender systems: Concepts, algorithms, and evaluations," in *Web Engineering* (M. Bakaev, F. Frasincar, and I.-Y. Ko, eds.), (Cham), pp. 574–577, Springer International Publishing, 2019.
- [18] P. Wang, J. Guo, Y. Lan, J. Xu, S. Wan, and X. Cheng, "Learning hierarchical representation model for nextbasket recommendation," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '15*, (New York, NY, USA), p. 403–412, Association for Computing Machinery, 2015.
- [19] F. Vasile, E. Smirnova, and A. Conneau, "Meta-prod2vec," *Proceedings of the 10th ACM Conference on Recommender Systems*, Sep 2016.
- [20] Y. K. Tan, X. Xu, and Y. Liu, "Improved recurrent neural networks for session-based recommendations," 2016.
- [21] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," 2012.
- [22] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in *Proceedings of the 19th International Conference on World Wide Web, WWW '10*, (New York, NY, USA), p. 811–820, Association for Computing Machinery, 2010.
- [23] R. He and J. J. McAuley, "Fusing similarity models with markov chains for sparse sequential recommendation," in *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain* (F. Bonchi, J. Domingo-Ferrer, R. Baeza-Yates, Z. Zhou, and X. Wu, eds.), pp. 191–200, IEEE Computer Society, 2016.
- [24] M. Quadrana, P. Cremonesi, and D. Jannach, "Sequence-aware recommender systems," 2018.
- [25] T. X. Tuan and T. M. Phuong, "3d convolutional networks for session-based recommendation with content features," in *Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys '17*, (New York, NY, USA), p. 138–146, Association for Computing Machinery, 2017.
- [26] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," 2015.
- [27] J. Tang and K. Wang, "Personalized top-n sequential recommendation via convolutional sequence embedding," 2018.
- [28] F. Yuan, A. Karatzoglou, I. Arapakis, J. M. Jose, and X. He, "A simple convolutional generative network for next item recommendation," 2018.
- [29] A. da Costa, E. Fressato, F. Neto, M. Manzato, and R. Campello, "Case recommender: A flexible and extensible python framework for recommender systems," in *Proceedings of the 12th ACM Conference on Recommender Systems, RecSys '18*, (New York, NY, USA), pp. 494–495, ACM, 2018.
- [30] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv:1803.01271*, 2018.

- [31] S. Mizrahi and P. Levin, “Combining context features in sequence-aware recommender systems,” in *RecSys*, 2019.
- [32] M. Grbovic, V. Radosavljevic, N. Djuric, N. Bhamidipati, J. Savla, V. Bhagwan, and D. Sharp, “E-commerce in your inbox: Product recommendations at scale,” in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15*, (New York, NY, USA), p. 1809–1818, Association for Computing Machinery, 2015.
- [33] D. Yang, D. Zhang, and B. Qu, “Participatory cultural mapping based on collective behavior data in location-based social networks,” *ACM Trans. Intell. Syst. Technol.*, vol. 7, no. 3, pp. 30:1–30:23, 2016.
- [34] M. Sertkan, J. Neidhardt, and H. Werthner, “Eliciting touristic profiles: A user study on picture collections,” in *Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization, UMAP 2020, Genoa, Italy, July 12-18, 2020* (T. Kuflik, I. Torre, R. Burke, and C. Gena, eds.), pp. 230–238, ACM, 2020.

DEFINICIONES

campo receptivo Área de entrada de una neurona.

Collaborative Filtering Sistema de recomendación que utiliza las similitudes entre usuarios para recomendar nuevos ítems, definida en la sección 2.1.3.

Content-Based Sistema de recomendación que basa sus recomendaciones en la similitud del contenido previo visto por un usuario, definida en la sección 2.1.3.

distancia Haversine Fórmula que determina la distancia entre dos puntos en una esfera dado las longitudes y latitudes.

embedding Representación de un atributo categórico mediante vectores de dimensión fija.

función de pérdida Método que indica lo bien que el algoritmo trabaja entrenando el modelo, un alto índice de pérdida indicará malos resultados.

item Objeto que se recomienda a un usuario y que puede ser relevante o no para él.

KNN (del inglés, k-nearest neighbors) método de clasificación supervisada que asigna la clase mayoritaria de los k vecinos más cercanos a un nuevo objeto para clasificarlo.

marca de tiempo Registro de tiempo de cuando se realizó una acción.

muestreo negativo Ítems que el usuario no ha puntuado que se utilizan en la función de coste de la fase de entrenamiento.

POI Un lugar específico (del inglés, Point of Interest) que puede resultar útil o interesante para alguien.

PyTorch Librería de aprendizaje automático de código abierto basada en la biblioteca Torch.

UAM

UNIVERSIDAD AUTONOMA

DE MADRID