UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

21|22

Trabajo fin de grado

Estudio de técnicas de aprendizaje automático para recomendar localizaciones según el tipo de negocio

Daniel González Pascual

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

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Estudio de técnicas de aprendizaje automático para recomendar localizaciones según el tipo de negocio

Autor: Daniel González Pascual
Tutor: Alejandro Bellogin Kouki

Junio 2022

Daniel González Pascual
Estudio de técnicas de aprendizaje automático para recomendar localizaciones según el tipo de negocio

Daniel González Pascual
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi familia y amigos

La mente es como un paracaídas,
sólo funciona si se abre.

Albert Einstein

Agradecimientos

En primer lugar me gustaría agradecer a mi tutor Alejandro Bellogín por confiar en mí para llevar a cabo este Trabajo de Fin de Grado. Además, quiero destacar su gran trabajo y amabilidad durante todo el proyecto.

También me gustaría agradecer todo el apoyo a mi familia, en especial a mis padres y mi hermana. Gracias a ellos he conocido desde pequeño la importancia de esforzarse por lo que uno quiere. Asimismo, particular mención a mi tío Felipe, que se fue cuando menos lo esperaba y que seguro le hubiese hecho ilusión verme conseguir mis metas.

Por último, y no por ello menos importantes, me gustaría reconocer la importancia de mis amigos, tanto los de mi zona como los de la universidad. Sin ellos no hubiese podido convertirme en lo que soy hoy en día, superar momentos de flaqueza, vivir diferentes experiencias y un largo etcétera.

Resumen

La selección de una ubicación para montar una nueva tienda siempre es un reto para los empresarios, ya que si se elige una buena localización puede reportar grandes beneficios, mientra que de lo contrario, puede ser la ruina del negocio. Habitualmente la selección del lugar se realiza a través del estudio de la zona, por ejemplo, midiendo la afluencia de clientes, la cercanía del transporte o la facilidad de aparcamiento entre otros aspectos. Esto supone un gran esfuerzo cuando hay miles de localizaciones a analizar.

En este Trabajo de Fin de Grado (TFG) se desarrolla un sistema de recomendación que, dada una tienda a montar, recomienda las localizaciones que predice como mejores. Este sistema consiste en un sistema de recomendación basado en filtrado colaborativo, que utiliza la factorización de matrices para realizar la predicción. El modelo se va ajustando hasta encontrar los patrones y así obtener los factores latentes con los que estimar la popularidad de dicha tienda en una zona concreta. Esto junto con el cálculo de las características de localización y de comercio de cada punto de interés hacen posible la resolución del problema planteado.

Seguidamente, se realizan pruebas de rendimiento tanto del recomendador implementado como de diferentes algoritmos de *Scikit-learn*. Estos son Support Vector Machine (SVM), Random Forest, Regresión logística, Naive Bayes, K-Nearest Neighbors (KNN), Perceptrón multicapa y predicción aleatoria. Su evaluación se mide a través de las métricas R-Precision, Mean Average Precision (MAP) y Mean Reciprocal Rank (MRR). Por otro lado, los conjuntos de datos utilizados han sido los proporcionados por Foursquare, que ofrecen información sobre diferentes puntos de interés como su localización o el número de usuarios que lo ha visitado.

Como conclusión, se observa que el modelo es capaz de ajustar las características latentes y los sesgos, aunque los datos estén muy dispersos, alcanzando así valores de las métricas muy similares a otros algoritmos de *Scikit-learn* u otros estudios con diferentes implementaciones. No obstante, el acierto del recomendador se ve condicionada por la densidad de datos en cada ciudad, obteniendo peores resultados cuando no se tienen suficientes datos.

Palabras clave

Sistemas de recomendación, filtrado colaborativo, factorización de matrices, características de localización, características de comercio, recomendación de localizaciones para nuevas tiendas, servicios basados en la localización

Abstract

The selection of a location to set up a new store is always a challenge for entrepreneurs, because if a good location is chosen it can bring high profits, while otherwise, it can be the ruin of the business. Usually, the selection of the place is made through the study of the area, for example, measuring the influx of customers, the proximity of transport or the ease of parking among other aspects. This is a great effort when there are thousands of locations to analyze.

In this Final Degree Project (FDP) a recommendation system is developed to recommend the locations which it predicts as the best to set up the indicated store type. This system consists of a recommendation system based on collaborative filtering, which uses matrix factorization to make the prediction. The model is adjusted to find the patterns and thus obtains the latent factors with which to estimate the popularity of the indicated store in a particular area. This, together with the location and commercial features of each point of interest, makes it possible to solve the problem.

Then, performance tests are made on both the implemented recommender and different algorithms from *Scikit-learn*. These are Support Vector Machine (SVM), Random Forest, Logistic Regression, Naive Bayes, K-Nearest Neighbors (KNN), Multilayer Perceptron, and Random Prediction. Their evaluation is measured through R-Precision, Mean Average Precision (MAP), and Mean Reciprocal Rank (MRR) metrics. At the same time, the datasets used have been those provided by Foursquare, which contain information about different points of interest such as their location or the number of users who have visited them.

To sum it up, it is observed that the model is able to adjust the latent features and biases, even though the data are very sparse, thus achieving very similar metric values to other *Scikit-learn* algorithms or other studies. However, the accuracy of the recommender is determined by the density of data for each city, obtaining worse results when there is not enough data.

Keywords

Recommendation systems, collaborative filtering, matrix factorization, location features, commercial features, recommendation of locations for new stores, location-based services

Índice

Listas

Lista de algoritmos

Lista de ecuaciones

Lista de figuras

Lista de tablas

1

Introducción

Hoy en día hay muchísima información, servicios y artículos, a tan solo un par de clics mediante nuestro *smartphone* u ordenador. Sin embargo, a menudo los usuarios se encuentran desorientados ante tal cantidad y variedad de productos. Por ello, es normal que cada vez tome más importancia el hecho de mostrar al usuario aquello que le puede interesar consumir.

1.1. Motivación del proyecto

Los sistemas de recomendación son herramientas que están en constante evolución gracias a su gran uso en servicios como Amazon o Netflix. Estos sistemas utilizan las valoraciones que los usuarios dan a los productos para tratar de predecir recomendaciones que pueden ser de su interés. Sin embargo, los Location-Based Services (LBS) como Google Maps o Instagram han abierto un nuevo paradigma en el campo de la recomendación, porque han introducido un nuevo dato con el que trabajar: las localizaciones. La popularidad de los teléfonos inteligentes ha permitido dejar en todo momento nuestra huella digital de dónde estamos o hemos estado. A partir de esta información, como se explica a lo largo de este trabajo, se pueden calcular diferentes características por cada Point Of Interest (POI) , como por ejemplo las de localización o las de comercio, con el objetivo de recomendar qué localizaciones son las mejores para montar una tienda. Este tipo de recomendación puede ser de gran interés dentro del mundo empresarial, más concretamente para aquellos empresarios que tienen en mente abrir un nuevo negocio y no saben dónde hacerlo.

1.2. Objetivos

El principal objetivo de este trabajo es afianzar los conocimientos adquiridos durante el grado en distintas asignaturas. En primer lugar, se estudian los sistemas de recomendación desde un punto de vista menos conocido como es la recomendación de localizaciones para montar una tienda. Posteriormente, se utilizan algoritmos de aprendizaje automático y aprendizaje profundo para comparar el rendimiento del recomendador a partir de diferentes métricas. Finalmente, se crea una maqueta de la aplicación web para poder utilizar el recomendador en un entorno real.

1.3. Estructura del documento

El documento se escuentra dividido en cinco capítulos:

Capítulo 1. Introducción: se presenta el problema sobre el que gira todo el trabajo, los objetivos a completar del proyecto y la estructura del documento.

Capítulo 2. Estado del arte: se explica cómo funcionan en general los sistemas de recomendación, las métricas de evaluación utilizadas y se detalla el problema a resolver durante el trabajo.

Capítulo 3. Diseño e implementación: contiene, por un lado, el diseño que se ha llevado a cabo durante el proyecto como puede ser el análisis de requisitos, su estructura general y su ciclo de vida. Por otro lado, se explica detalladamente la implementación del sistema.

Capítulo 4. Pruebas y resultados: se detalla cómo se han realizado y cuáles son los experimentos que se han llevado a cabo para estudiar el rendimiento del recomendador implementado. Además, se muestran y analizan los resultados obtenidos.

Capítulo 5. Conclusiones y trabajo futuro: se exponen las conclusiones finales del trabajo, así como las posibles mejoras que se pueden investigar como trabajo futuro.

2

Estado del arte

En este capítulo se realiza una introducción a los sistemas de recomendación, desde cómo funcionan en general, las técnicas más utilizadas y algunos de sus puntos débiles, hasta las métricas para poder llevar a cabo su evaluación. Además, se plantea y explica un tipo de problema algo diferente a lo habitual, el cual consiste en la recomendación basada en la localización.

2.1. Sistemas de recomendación

Los sistemas de recomendación son algoritmos que detectan patrones de los usuarios y tienen como objetivo sugerir *items* que podrían ser de su interés [1]. Comúnmente se utilizan por ejemplo las puntuaciones, las compras o las búsquedas para localizar dichos patrones y así tratar de predecir si un *item* puede interesarle al usuario o no. Algunos de estos *items* son ropa, vídeos, personas, noticias, etc., es decir, cualquier cosa que tenga sentido ser buscada [2].

La recomendación es un campo que se encuentra en constante evolución y presente en nuestro día a día a través de diferentes servicios. Desde cuando vemos una película hasta cuando puntuamos un lugar. Por ello, en esta sección se describe cómo funcionan en general, las técnicas más utilizadas y algunos de sus principales problemas.

2.1.1. Formulación del problema

La recomendación de aquellos *items* que no ha visto un usuario, a la vez que se optimizan sus preferencias, no es una tarea sencilla. En primer lugar, para que el sistema de recomendación pueda llevar a cabo su trabajo, se necesitan datos, tanto de los usuarios como de los *items*. Además, muchas veces, se requiere de una escala de cuánto le gusta un *item* a un usuario. Esta se conoce como *rating* y suele tener valores entre los intervalos [1, 10], [1, 5] o [0.1, 1.0].

Los usuarios suelen representarse como $\{u_1, ..., u_n\} \in U$ y los *items* como $\{i_1, ..., i_m\} \in I$. A partir de ellos se obtiene la matriz de valoraciones R de tamaño $|U|$ x $|I|$, en la que la intersección de un usuario u y un *item* i es la puntuación que el usuario u da al *item* i (r_{ui}) [3]. Un ejemplo de ello

se puede observar en la figura 2.1 donde las puntuaciones marcadas con una interrogación son los *ratings* a predecir por el sistema de recomendación.

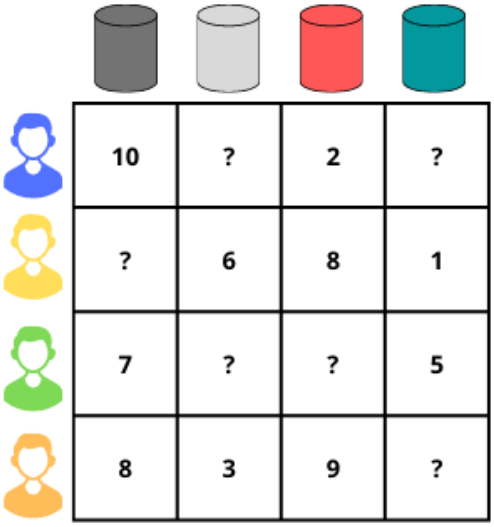


Figura 2.1: Ejemplo de matriz de valoraciones.

Para poder llevar a cabo la resolución del problema de recomendación, existen diferentes algoritmos que se agrupan según la técnica (o los datos) que utilicen [1]: basados en filtrado colaborativo, basados en contenido e híbridos. Todos ellos se explican a continuación, aunque nos vamos a centrar en los sistemas de recomendación basados en filtrado colaborativo ya que ha sido la técnica utilizada para desarrollar este trabajo.

2.1.2. Recomendación basada en filtrado colaborativo

El filtrado colaborativo es una técnica que consiste en predecir lo que un usuario opinaría sobre un *item*, si lo consumiera, en base a lo que otros usuarios con gustos similares han puntuado [4]. Los diferentes métodos que aplican esta técnica se clasifican principalmente en dos grupos: basado en modelo y basado en memoria.

Basado en modelo

Los modelos se van ajustando hasta encontrar patrones de los datos de entrenamiento y así obtener los factores latentes con los que calcular los valores reales. Uno de los métodos más populares es la factorización de matrices [5, 6], el cual ha sido utilizado para el desarrollo de este trabajo y en el que nos centraremos a continuación para conocerlo mejor.

La factorización de matrices consiste en la descomposición matricial de dos o más matrices con la que obtener la matriz objetivo a partir del producto. En la figura 2.2 se puede apreciar un ejemplo de la técnica matemática Singular Value Decomposition (SVD) en la que se basa [7].

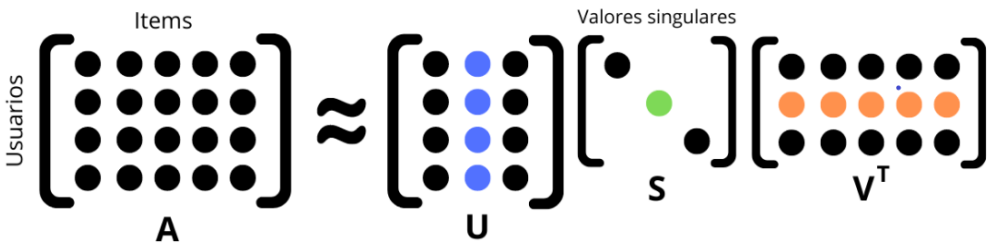


Figura 2.2: Técnica matemática SVD.

La predicción del *rating* de un usuario sobre un *item* se obtiene, como se puede observar en la ecuación 2.1, mediante la multiplicación escalar de los dos vectores.

$$\hat{r}(u,i) = p_u^T q_i \tag{2.1}$$

Siendo p_u el vector de factores latentes correspondiente al usuario u y q_i el vector de factores latentes correspondiente al *item* i.

Durante el entrenamiento se va minimizando el error a partir del algoritmo de optimización descenso por gradiente. Así, como se puede observar en las ecuaciones 2.2, se van ajustando los factores latentes mejorando las predicciones [8].

$$\begin{aligned} e(u,i) &= r(u,i) - \hat{r}(u,i) \\ p_u &= p_u + \alpha \cdot [e(u,i) \cdot q_i - \lambda \cdot p_u] \\ q_i &= q_i + \alpha \cdot [e(u,i) \cdot p_u - \lambda \cdot q_i] \end{aligned} \tag{2.2}$$

Donde $r(u,i)$ es el *rating* real, $e(u,i)$ es la diferencia de error entre el *rating* real y el predicho, α se corresponde con la tasa de aprendizaje y λ es un parámetro regularizador para evitar el sobreajuste.

Además, en nuestro caso, usaremos una versión propuesta en [8] donde se integran los sesgos de los usuarios (b_u) y los *items* (b_i) en el momento de aprender el modelo. Como se puede apreciar en la ecuación 2.3, durante el entrenamiento estos sesgos se ajustan de manera similar a como lo hacen los factores latentes. El objetivo que tienen los sesgos en nuestro trabajo es que se complementen con las características de localización y de comercio, explicadas en la subsección 3.2.2, para así mejorar las predicciones.

$$\begin{aligned} b_u &= b_u + \alpha \cdot [e(u,i) - \lambda \cdot b_u] \\ b_i &= b_i + \alpha \cdot [e(u,i) - \lambda \cdot b_i] \end{aligned} \tag{2.3}$$

Basado en memoria

Los métodos que integran este grupo trabajan con las puntuaciones de los usuarios más similares para así predecir lo que votaría un usuario a un *item*. Es decir, para realizar la predicción de los *ratings* desconocidos de un usuario, se obtienen los usuarios con mayor similitud y se observa qué han votado ellos. Esto se puede utilizar también al revés, es decir, buscar *items* similares a los que le han gustado a un usuario, pero siempre utilizando algún concepto de *similitud*. Como se puede apreciar a simple vista, en comparación con el grupo anterior, son algoritmos más sencillos de implementar y definir, pero que dependen mucho de la cantidad de datos que haya [9].

Uno de los métodos más populares es K-Nearest Neighbors (KNN) , también conocido como k vecinos más próximos [10]. Como hemos mencionado anteriormente, se puede definir tanto basado en usuarios como en *items*.

KNN basado en usuarios: como se puede observar en la ecuación 2.4, se predice el *rating* de un usuario sobre un *item* en base a lo que hayan puntuado a ese *item* los usuarios más similares.

$$\hat{r}(u,i) = \frac{\sum_{v \in N_k(u): r(v,i) \neq 0} sim(u,v) \cdot r(v,i)}{\sum_{v \in N_k(u): r(v,i) \neq 0} |sim(u,v)|} \tag{2.4}$$

Siendo u el usuario al que se desea predecir su *rating* sobre el *item* i, $N_k(u)$ son sus vecinos (*usuarios*) más próximos y $sim(u,v)$ es la función de similitud entre usuarios.

KNN basado en *items*: en este caso, como se puede apreciar en la ecuación 2.5, se predice el *rating* de un usuario sobre un *item* en base a lo que haya puntuado ese mismo usuario a los *items* más similares.

$$\hat{r}(u,i) = \frac{\sum_{j \in N_k(i): r(u,j) \neq 0} sim(i,j) \cdot r(u,j)}{\sum_{j \in N_k(i): r(u,j) \neq 0} |sim(i,j)|} \tag{2.5}$$

Donde u es el usuario al que se desea predecir su *rating* sobre el *item* i, $N_k(i)$ son los vecinos (*items*) del ítem i más próximos y $sim(i,j)$ es la función de similitud entre los *items*.

Algunas de las funciones de similitud más utilizadas son las siguientes (se indican sólo las similitudes entre usuario, las de ítems se pueden obtener de manera análoga):

Distancia euclídea: calcula la distancia entre dos usuarios entendiéndolos como puntos o vectores.

$$sim(u,v) = \sqrt{\sum_{i \in I: r(u,i) \neq 0, r(v,i) \neq 0} [r(u,i) - r(v,i)]^2} \tag{2.6}$$

Coseno: calcula el coseno del ángulo entre dos vectores formados a partir de la representación de los usuarios en el espacio.

$$sim(u,v) = \frac{\sum_{i \in I: r(u,i) \neq 0, r(v,i) \neq 0} r(u,i) r(v,i)}{\sqrt{\sum_{i \in I: r(u,i) \neq 0} r(u,i)^2 \sum_{i \in I: r(v,i) \neq 0} r(v,i)^2}} \tag{2.7}$$

Correlación de Pearson: parecida a la similitud coseno, pero teniendo en cuenta el *rating* promedio, ya que algunos usuarios tienden a poner puntuaciones altas y otros bajas, de manera que se tenga en cuenta esta variabilidad al calcular las similitudes.

$$sim(u,v) = \frac{\sum_{i \in I: r(u,i) \neq 0, r(v,i) \neq 0} [r(u,i) - \bar{r}_u][r(v,i) - \bar{r}_v]}{\sqrt{\sum_{i \in I: r(u,i) \neq 0, r(v,i) \neq 0} [r(u,i) - \bar{r}_u]^2 \sum_{i \in I: r(v,i) \neq 0, r(u,i) \neq 0} [r(v,i) - \bar{r}_v]^2}} \tag{2.8}$$

Jaccard (ecuación 2.9a) y **Dice** (ecuación 2.9b): como se puede apreciar en las ecuaciones anteriores, sólo se tienen en cuenta los *ratings* de los *items* que ambos usuarios han valorado, es decir, dos usuarios pueden tener una gran similitud habiendo puntuado sólo un *item* en común. Jaccard y Dice tienen como objetivo corregir esta situación midiendo el solapamiento entre las valoraciones de ambos usuarios.

$$sim(u,v) = \frac{|u \cap v|}{|u \cup v|} = \frac{|u \cap v|}{|u| + |v| - |u \cap v|} \tag{2.9a}$$

$$sim(u,v) = \frac{2|u \cap v|}{|u| + |v|} \tag{2.9b}$$

Donde u representa los ítems que ha puntuado el usuario u, para realizar operaciones de conjunto entre los usuarios.

2.1.3. Otros tipos de sistemas de recomendación

Basados en contenido

La recomendación basada en contenido, a diferencia de filtrado colaborativo, predice sin tener en cuenta a otros usuarios. Así pues, se recomiendan *items* con características similares a otros que el usuario ha consumido en el pasado [11]. Por ello, al no tener en cuenta el *rating*, puede recomendar *items* que sean nuevos. Sin embargo, no sirve para usuarios nuevos, ya que no tenemos productos de referencia. No obstante, para solucionar este problema se podría utilizar información personal como sus datos demográficos u otras características de los usuarios.

Los *items* se representan como vectores en un espacio de características. Algunos ejemplos de características son la categoría, el autor, el idioma, etc. Por lo tanto, a la hora de calcular similitudes se utilizan funciones como Coseno (ecuación 2.7, pero cambiando los *ratings* por las características) o Jaccard (ecuación 2.9a) para características binarias. Algunos métodos muy utilizados por este tipo de sistemas de recomendación para clasificar los *items* a predecir son KNN y basado en redes neuronales [12].

Híbridos

Los sistemas de recomendación basados en filtrado colaborativo sufren, por ejemplo, cuando hay *items* nuevos. Sin embargo, los basados en contenido lo hacen con los usuarios nuevos. Por ello, como se puede apreciar, su combinación es muy positiva, ya que se retroalimentan para así obtener una mejor predicción. Algunas de las estrategias más conocidas para combinar algoritmos de recomendación son las siguientes [13]:

Ponderado: las predicciones de diferentes métodos de recomendación son unidas para obtener una en conjunto.

Intercambiando: el sistema va cambiando entre los diferentes métodos de recomendación según la situación.

Mezclado: las recomendaciones de los diferentes métodos se presentan a la vez. Por ejemplo, utilizando filtrado colaborativo para los usuarios y basado en contenido para los *items*.

Combinación de las características: las características de distintos métodos de recomendación se unen como entrada a otro.

En cascada: se realiza un proceso por etapas, es decir, un recomendador refina lo predicho por otro.

Aumento de las características: la información de los métodos de recomendación de un tipo se trata como características adicionales de los métodos del otro tipo.

Metalevel: el modelo aprendido por un recomendador es utilizado como dato de entrada de otro.

2.1.4. Puntos débiles

Como ocurre con todos los algoritmos, los sistemas de recomendación no son perfectos, tienen puntos débiles que afectan negativamente a la recomendación. Algunos de ellos son:

Escasez de información

Uno de los principales problemas es lo conocido como **arranque en frío**. Esta situación puede deberse por diferentes razones: el sistema de recomendación se encuentra en una fase inicial, hay usuarios o *items* nuevos, o los usuarios son poco activos y no puntúan suficientes *items* [14]. Posibles soluciones son, para los *items* nuevos, utilizar la recomendación basada en contenido [15] y, para los usuarios nuevos o poco activos, utilizar información estática como los datos personales, cuestionarios, redes sociales, etc. [16]

Manipulación

Los sistemas de recomendación son utilizados por diversos servicios que consumimos a diario, por ejemplo, Amazon o Google Maps. Por ello, surge la posibilidad de que usuarios propietarios de una tienda puedan puntuar positivamente sus productos y/o establecimientos mientras que lo hacen negativamente con los de la competencia para perjudicarlos y así posicionarse mejor. Esto es lo que se conoce como ataques, y hay literatura que intenta obtener algoritmos que sean robustos frente a ellos [17].

Efecto *filter bubble* y sesgos

Los sistemas de recomendación predicen lo que le podría interesar al usuario basándose en su información personal. Sin embargo, se debe tratar de evitar la situación en la que los usuarios son apartados de información que no concuerda con sus puntos de vista y se mantienen aislados en burbujas, como por ejemplo pueden ser ideológicas y culturales [18, 19]. Algo similar ocurre con los sesgos, por lo que se debe procurar evitar tanto los sesgos de escasez como los de popularidad [20].

2.2. Recomendación basada en la localización

Hoy en día se nos hace imposible imaginar una realidad sin *smartphones* o teléfonos inteligentes. Esto se debe a que están muy arraigados en nuestro día a día y, en muchos casos, nos lo simplifica gracias a las facilidades de comunicación, búsqueda de información, etc. que nos brindan.

La popularidad de los móviles ha tenido consigo numerosas actualizaciones e introducciones de nuevos servicios. Uno de ellos, el cual es el que nos interesa en este trabajo, es Global Positioning System (GPS) . Gracias a este servicio el teléfono dispone de localización geográfica.

El GPS se utiliza en numerosas aplicaciones como, por ejemplo, mapeado o búsqueda de puntos de interés. Esto permite que los sistemas de recomendación basados en la localización estén en constante evolución y puedan recomendar servicios ajustándose a los patrones de movimiento. Sin embargo, al ser un dominio concreto, tiene sus particularidades y limitaciones, por ello, lo vamos a analizar con más detalle a lo largo de esta sección.

2.2.1. Datos de entrada

Estos sistemas de recomendación esperan distintos tipos de datos de entrada. En primer lugar, como es obvio, las localizaciones geográficas que representan los diferentes POIs [21]. Seguidamente, como se puede observar en la figura 2.3, la categoría del servicio que ofrecen. Esta es importante conocerla porque hay numerosos tipos de POIs, y muchas veces los patrones surgen a este nivel. Por ejemplo, un usuario puede no repetir el ir a un bar en concreto, sino que tiene el hábito de ir a bares con cierta frecuencia. Por último, una unidad que se pueda utilizar como métrica, ya sea me gustas, valoraciones o, como en nuestro trabajo, *checkins*. Los *checkins* son registros de visitas que hacen los usuarios a los distintos puntos de interés. Estos son muy útiles porque nos permiten extraer diferente información sobre:

Puntuaciones: se pueden normalizar los *checkins* para obtener un *rating* de cada localización.

Favoritos: si un usuario registra numerosos *checkins* de un mismo lugar se puede considerar que es uno de sus sitios favoritos.

Temporalidad: si además del checkin se registra la hora y el día, se puede calcular rutinas de los usuarios, qué sitios son los preferidos en hora punta, en qué época del año son más populares algunos lugares, etc.

Dimensión espacial: se puede obtener la popularidad de cada zona, los diferentes lugares de una misma zona que visita el usuario, el transporte público cercano, etc.

Figura 2.3: Mapa con puntos de interés. *Extraído de [22]*

2.2.2. Dificultades

La popularidad de los LBS es indiscutible. Sin embargo, se enfrentan a dificultades como la escasez de datos, ya que al tener un valor tan importante no son fáciles de conseguir. Además, por lo general, los usuarios son poco activos a la hora de interactuar con los diferentes puntos de interés. Otro punto a tener en cuenta es que la gran mayoría de sistemas de recomendación están definidos para usar *ratings*, sin embargo en este caso no hay *ratings* sino *checkins*, como se ha explicado anteriormente.

La recomendación basada en la localización ya presenta dificultades por sí misma, puesto que es un problema muy complejo de resolver. Habitualmente los métodos basados en filtrado colaborativo son los encargados de tratar de solucionar este problema y predecir las recomendaciones de localizaciones. Para ello la información que se obtiene de los LBS es muy útil como dato de entrada [23]. Como hemos mencionado en el apartado anterior 2.2.1, es muy común normalizar los *checkins* de cada localización para así obtener su *rating*. No obstante, se han propuesto modelos que trabajan con otros datos distintos, como las reseñas, las fotografías, etc. [24]

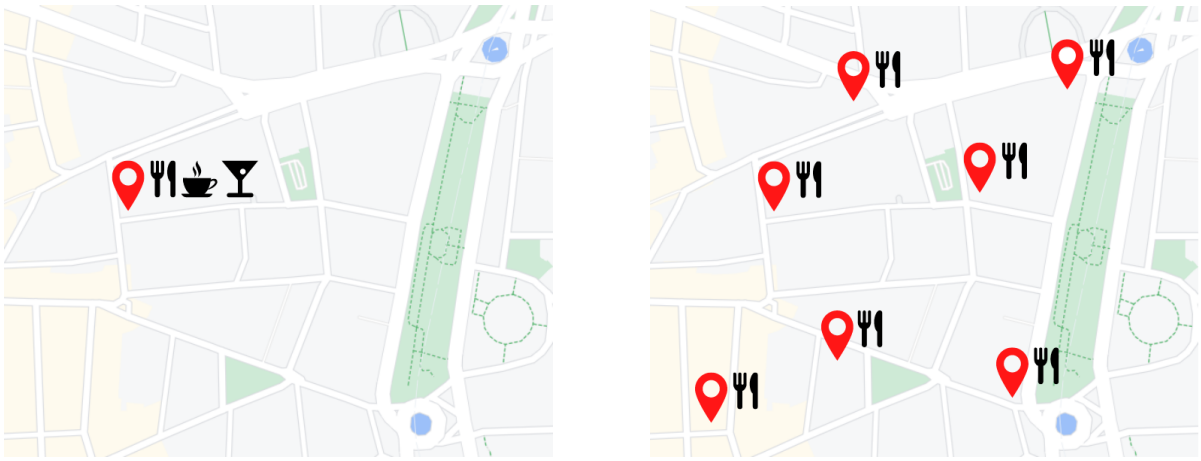
A las dificultades mencionadas cabe añadir las que surgen dependiendo del tipo de problema que se elija. Los diferentes tipos de problema se explican a continuación en la siguiente subsección 2.2.3.

2.2.3. Recomendación de tipos de tiendas o localizaciones

La recomendación basada en la localización suele estar orientada a recomendar *items* a usuarios, sin embargo, a partir de los mismos datos pero explotándolos con algoritmos diferentes, se pueden resolver otras tareas. En concreto, las que nos interesan en este trabajo son la de recomendación de tipos de tiendas y de localizaciones. Son problemas complejos de resolver y poco estudiados en la literatura. Y es que hay que tener en cuenta que ya no se trabaja con usuarios e *items* como los problemas de recomendación tradicionales, sino que se trabaja con tipos de tiendas y localizaciones. Además, una diferencia fundamental entre un tipo de recomendación y otro es que para la recomendación de tiendas y localizaciones es necesario la explotación de las características de localización y de comercio. A lo largo del capítulo 3, diseño e implementación, conoceremos más a fondo cómo se obtienen y se trabaja con ellas. Como se puede observar en la figura 2.4, se pueden resolver estos problemas de dos formas. Por un lado, se introduce una localización al recomendador y este devuelve un ránking con los mejores tipos de tiendas para montar. Por otro lado, se introduce el tipo de tienda que se quiere montar y el recomendador devuelve las localizaciones que considera mejores para hacerlo.

En este trabajo se ha desarrollado la segunda opción, es decir, dado un tipo de tienda a montar, se recomiendan las mejores localizaciones para hacerlo. Sin embargo, se ha partido de la primera opción, ya que se tenía disponible un desarrollo previo de un compañero para su Trabajo de Fin de Máster (TFM) [25]. La idea original en ese trabajo se extrajo del artículo [26], que está centrado únicamente en la primera opción.

Por tanto, puesto que ambos problemas tienen sus similitudes, se ha decidido implementar la segunda opción una vez se ha entendido la primera, asumiendo las dificultades de dar la vuelta al problema, pero consiguiendo, de esta forma, poder comparar nuestros resultados con otros estudios.



(a) Recomendación de tipos de tiendas para una localización

(b) Recomendación de localizaciones para un tipo de tienda

Figura 2.4: Formas de recomendar tipos de tiendas 2.4(a) o localizaciones 2.4(b)

2.3. Evaluación de los sistemas de recomendación

Las métricas de evaluación son herramientas que permiten observar qué tan buenas son las predicciones del sistema de recomendaciones [27]. Las utilizadas en este trabajo han sido:

Métricas de error

Mean Absolute Error (MAE) : el error absoluto medio mide el promedio de la diferencia absoluta entre los *ratings* reales y los *ratings* predichos. Cuanto más cercano su valor a cero, más similares son las puntuaciones reales y las predichas.

$$MAE = \frac{\sum_{(u,i)\in C} |r(u,i) - \hat{r}(u,i)|}{|C|} \tag{2.10}$$

Siendo C los *ratings* de las parejas usuario-*item* a evaluar.

Root Mean Squared Error (RMSE) : el error cuadrático medio, a diferencia de la métrica MAE, calcula la raíz cuadrada del promedio de la diferencia cuadrada, es decir, en vez de utilizar el valor absoluto, eleva la diferencia al cuadrado y hace la raíz cuadrada. Esta métrica penaliza más el error cuanto mayor sea la diferencia.

$$RMSE = \sqrt{\frac{\sum_{(u,i)\in C} [r(u,i) - \hat{r}(u,i)]^2}{|C|}} \tag{2.11}$$

Métricas clásicas

Precision: mide cuántos relevantes se devuelven en la predicción.

$$Precision = \frac{|Relevant \cap Returned|}{|Returned|} \tag{2.12}$$

Recall: indica cuántos relevantes del total se han devuelto.

$$Recall = \frac{|Relevant \cap Returned|}{|Relevant|} \tag{2.13}$$

Métricas del ránking predicho

R-Precision: mide el número de *items* relevantes que se devuelven en el ránking predicho.

$$R - Precision = P@|RR| = \frac{|Relevant \cap Returned@|RR||}{|RR|} \tag{2.14}$$

Donde P es la métrica *Precision*, $|RR|$ es el número de *items* relevantes reales y la notación $@$ marca el *cutoff*, es decir, de los *items* devueltos ($Returned$) sólo tenemos en cuenta para calcular la métrica hasta $|RR|$.

Average Precision (AP) : la precisión media es el promedio de la precisión en las posiciones del ránking en las que se encuentra un item relevante. Mean Average Precision (MAP) es el promedio de los diferentes AP calculados por cada predicción.

$$AP = \frac{\sum_{k \in RP} P@k}{|RP|} \tag{2.15}$$

Siendo RP las posiciones del ránking donde se encuentran los relevantes.

Reciprocal Rank (RR) : es el inverso multiplicativo de la posición del ránking en la que se encuentra el primer relevante. Mean Reciprocal Rank (MRR) es el promedio de los diferentes RR calculados por cada predicción.

$$RR = \frac{1}{rank\_pos\_first\_relevant} \tag{2.16}$$

3

Diseño e implementación

En este capítulo se detalla el diseño que se ha llevado a cabo durante el desarrollo del proyecto como puede ser los requisitos de la aplicación, la estructura general del proyecto y su ciclo de vida. Además, se explica detalladamente la implementación de cada fase y módulo del sistema.

3.1. Diseño

El diseño es una de las fases más importantes de un proyecto ya que será desarrollado según lo que se acuerda en esta fase. Por ello en esta sección se muestran los requisitos que debe cumplir la aplicación, la estructura general del proyecto, las tecnologías utilizadas y el proceso llevado a cabo.

3.1.1. Estructura general

La estructura general del proyecto se compone de 4 módulos como se puede observar en la figura 3.1. Además, estos están enumerados para así conocer el orden de su implementación. A continuación se hará una breve introducción de los módulos mencionados, sin embargo su funcionalidad está explicada con todo tipo detalle en la sección 3.2.

1. Procesado de datos: este módulo es el encargado de tratar los conjuntos de datos para que posteriormente puedan ser utilizados por los recomendadores.

2. Cálculo de las características: a partir de la información de los POIs de los conjuntos de datos ya procesados se calculan las características de localización y de comercio con las que podrán trabajar los sistemas de recomendación.

3. Sistemas de recomendación: este módulo trabaja con las características previamente calculadas para entrenar los modelos. Una vez se lleva a cabo su entrenamiento, dichos modelos están preparados para resolver el problema de recomendación.

4. Visualización: las recomendaciones de cada sistema se pueden visualizar en el mapa de la aplicación web desarrollada.

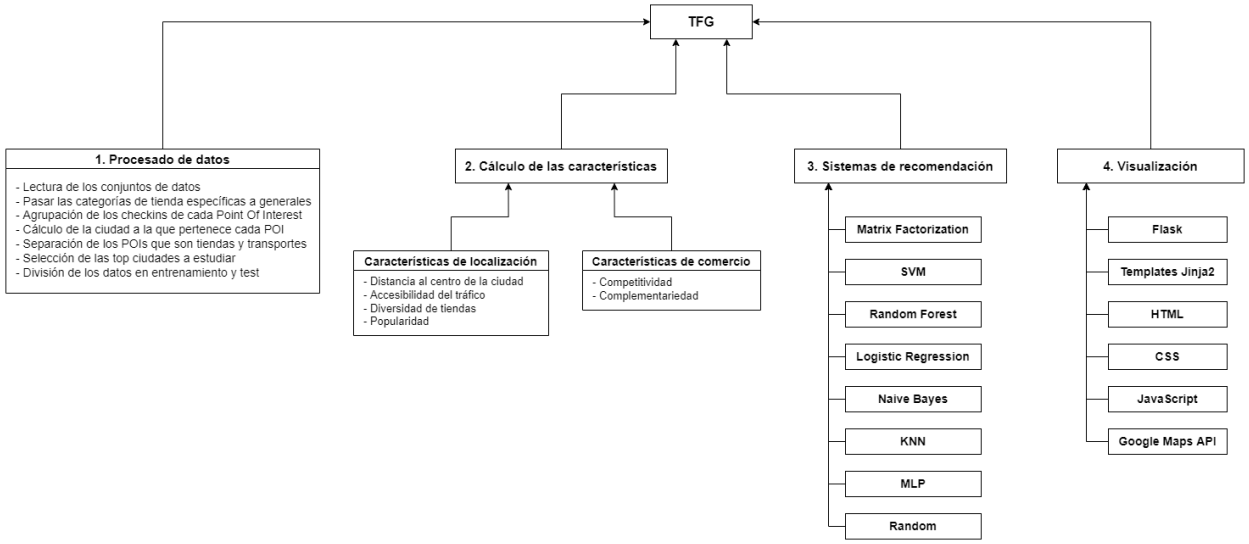


Figura 3.1: Diagrama con los diferentes módulos de la aplicación y su relación.

Diagrama de secuencia

La figura 3.2 muestra el diagrama de secuencia de la aplicación, en él se pueden observar los módulos mencionados anteriormente y el proceso de ejecución entre ellos. En primer lugar, se procesan los conjuntos de datos originales. Una vez se tienen procesados los datos, el *script* se encarga de controlar la lógica del programa para calcular las características y obtener la recomendación. Finalmente, se muestran los resultados en la aplicación web.

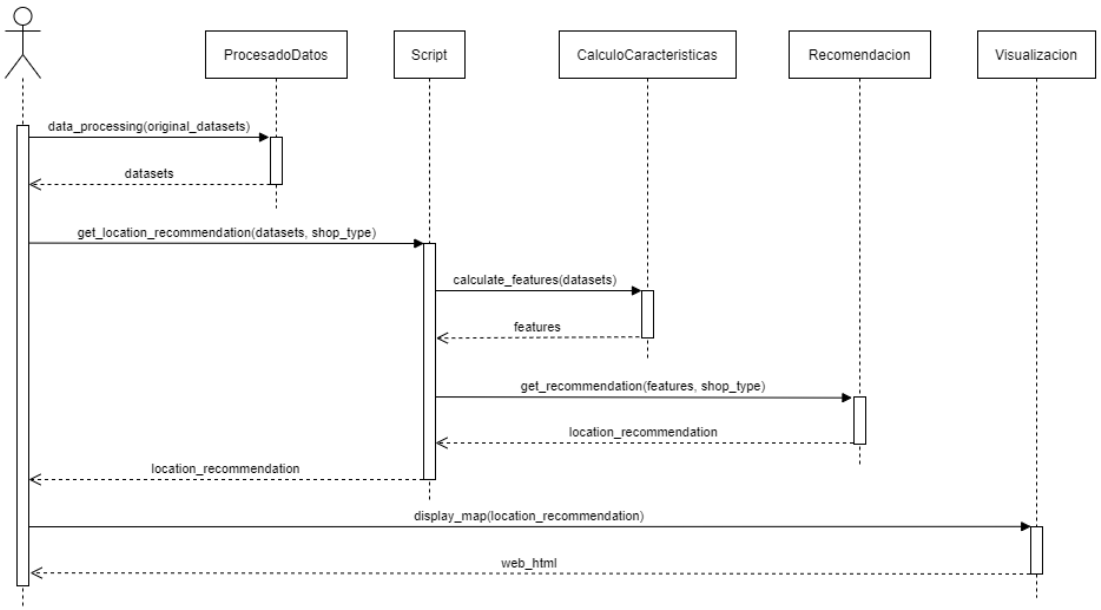


Figura 3.2: Diagrama de secuencia de la aplicación.

3.1.2. Ciclo de vida

Para el desarrollo del proyecto se ha seguido un proceso en cascada, como se puede observar en la figura 3.3. Esta decisión ha sido determinada principalmente porque el proceso del proyecto estaba bien definido desde su comienzo, las diferentes etapas necesitan los resultados de las anteriores para poder llevarse a cabo y se trata de un trabajo individual, sin un equipo con el que poder paralelizar las tareas.

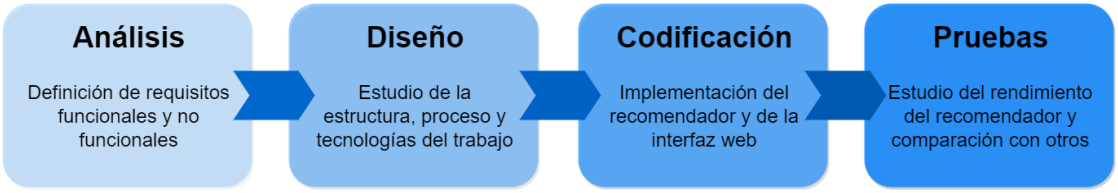


Figura 3.3: Ciclo de vida en cascada del proyecto.

3.1.3. Requisitos

En esta sección se muestran los diferentes requisitos que debe cumplir la aplicación. En primer lugar, los requisitos están divididos en funcionales y no funcionales. Posteriormente, cada requisito funcional se encuentra asociado al módulo que le corresponde y los no funcionales agrupados por categorías.

Requisitos funcionales

Datos

RF-1.– La aplicación debe permitir utilizar diferentes conjuntos de datos.

Características

RF-2.– La aplicación debe emplear las características de localización y de comercio.

Recomendación

RF-3.– La aplicación debe trabajar tanto con el recomendador como con otros algoritmos.

RF-4.– Los sistemas de recomendación deben generar un ránking en base al *rating* predicho.

RF-5.– La aplicación debe utilizar los datos de test para la recomendación.

RF-6.– La aplicación debe utilizar métricas de evaluación para saber cuánto de buena es la recomendación.

Visualización

RF-7.– La aplicación debe consentir realizar recomendaciones dada una ciudad y una categoría de tienda.

RF-8.– La aplicación debe mostrar un formulario donde el usuario pueda elegir sus preferencias.

RF-9.– La aplicación debe permitir ver las recomendaciones en un mapa.

RF-10.– La aplicación debe consentir ver al mismo tiempo la recomendación del recomendador y la de otro algoritmo para así poder comparar.

RF-11.– La aplicación debe posibilitar cambiar lo que se muestra en el mapa.

Requisitos no funcionales

Usabilidad:

RNF-1.– La aplicación debe tener una interfaz simple, es decir, sólo las opciones necesarias para resolver el problema.

RNF-2.– La aplicación debe ser *responsive*, permitiendo adaptarse a todo tipo de pantalla.

Eficiencia:

RNF-3.– La aplicación debe trabajar con los datos cargados en memoria para agilizar su ejecución.

RNF-4.– El tiempo de ejecución del modelo no debe superar las 12h.

RNF-5.– La memoria utilizada debe ser inferior a la mitad de la que dispone el dispositivo.

Desarrollo:

RNF-6.– El lenguaje de programación debe ser Python para que sea multiplataforma.

RNF-7.– Las pruebas deben realizarse en cuadernos de Jupyter para facilitar su lectura.

Mantenibilidad:

RNF-8.– Los sesgos, las características latentes y las características de localización y de comercio ya calculados deben almacenarse en disco para futuras ejecuciones con la misma configuración.

Seguridad:

RNF-9.– Los datos de los usuarios utilizados deben ser anónimos para cumplir la Ley de Protección de Datos.

3.2. Implementación

En esta sección se explica en orden y con todo tipo de detalles la implementación que se ha llevado a cabo para realizar el proyecto, desde el procesado de los conjuntos de datos originales hasta la visualización de las recomendaciones en la aplicación web, pasando por el cálculo de las características y el desarrollo del recomendador.

3.2.1. Procesado de datos

Los conjuntos de datos que se han utilizado son los proporcionados por Foursquare. [1] Posteriormente, estos han sido procesados para obtenerlos con una estructura determinada con la que trabajar.

[1] Foursquare Dataset: https://sites.google.com/site/yangdingqi/home/foursquare-dataset

Los conjuntos de datos originales son:

POIs: el archivo contiene los distintos puntos de interés registrados por Foursquare. Como se puede apreciar en la figura 3.4, cada POI está representado por un identificador, las coordenadas de latitud y longitud donde se encuentran, el tipo de tienda que es y el código del país al que pertenece.

Venue ID	Latitude	Longitude	Venue category name	Country code
3fd66200f964a52000e71ee3	40.73	-74	Jazz Club	US
3fd66200f964a52000e81ee3	40.76	-73.98	Gym	US
3fd66200f964a52000ea1ee3	40.73	-74	Indian Restaurant	US
3fd66200f964a52000ee1ee3	39.93	-75.16	Sandwich Place	US
3fd66200f964a52000f11ee3	40.65	-74	Bowling Alley	US

Figura 3.4: Muestra de datos de los POIs.

***Checkins*:** conjunto de datos con los registros que han hecho los diferentes usuarios en los diferentes puntos de interés. La figura 3.5 presenta una muestra de los datos.

User ID	Venue ID	UTC time	Timezone offset in minutes
50756	4f5e3a72e4b053fd6a4313f6	Tue Apr 03 18:00:06 +0000 2012	240
190571	4b4b87b5f964a5204a9f26e3	Tue Apr 03 18:00:07 +0000 2012	180
221021	4a85b1b3f964a520eefe1fe3	Tue Apr 03 18:00:08 +0000 2012	-240
66981	4b4606f2f964a520751426e3	Tue Apr 03 18:00:08 +0000 2012	-300
21010	4c2b4e8a9a559c74832f0de2	Tue Apr 03 18:00:09 +0000 2012	240

Figura 3.5: Muestra de datos de los *checkins*.

Ciudades: el fichero contiene las ciudades registradas por Foursquare. Por cada ciudad, como se puede observar en la figura 3.6, se tiene su nombre, sus coordenadas de latitud y longitud del centro de la ciudad, el país al que pertenece y el tipo de ciudad que es, es decir, si es la capital de un país, de una provincia, etc.

City name	Latitude of city center	Longitude of city center	Country code	Country name	City type
New York	40.71	-73.91	US	United States	Other
Harrisburg	40.27	-76.9	US	United States	Provincial capital
Trenton	40.22	-74.78	US	United States	Provincial capital
Philadelphia	39.93	-75.22	US	United States	Other
Boston	42.37	-71.1	US	United States	Provincial capital

Figura 3.6: Muestra de datos de las ciudades.

Categorías: el archivo almacena información de los tipos de tiendas que hay registrados, como se puede apreciar en la figura 3.7, desde algo más específico como es el nivel 4 a algo más general como es el nivel 1.

Level1_name	Level2_name	Level3_name	Level4_name
Food	Asian Restaurants	Japanese Restaurants	Donburi Restaurants
Food	Asian Restaurants	Japanese Restaurants	Japanese Curry Restaurants
Shops & Services	Food & Drink Shops	Cheese Shops	
Shops & Services	Food & Drink Shops	Fish Markets	
Shops & Services	Food & Drink Shops	Food Services	

Figura 3.7: Muestra de datos de las categorías de tienda.

Una vez conocemos el formato de los conjuntos de datos originales, los procesamos para obtener la estructura de datos deseada. En primer lugar, en la tabla 3.1 se muestra la diferencia de datos dependiendo del año del que se obtengan. Por ello, concatenamos los conjuntos de datos de cada año y borramos los duplicados para tener un mayor número de datos con los que trabajar.

Conjuntos de datos	2015	2019
Checkins	**33 263 633**	22 809 624
POIs	3 680 126	**11 180 160**

Tabla 3.1: Diferencia de datos. En negrita los conjuntos con más datos.

Posteriormente, agrupamos los *checkins* de los usuarios según el identificador de cada punto de interés. Así obtenemos el número de registros que ha tenido cada tienda.

En la figura 3.7 se puede apreciar que Foursquare clasifica las categorías de tienda en distintos niveles de profundidad. Por ello, para no añadir una mayor complejidad al problema, transformamos las categorías más específicas (nivel 3 o 4) en una categoría más general (nivel 2). La decisión de no utilizar el nivel 1 se debe a que es un nivel tan general, con 10 categorías como ”Food” o ”Professional & Other Places”, que no aporta apenas información.

Finalmente, filtramos los datos obtenidos según la ciudad a la que pertenece cada punto de interés. Además, cribamos los puntos de interés con una gran distancia al centro de la ciudad, ya que se han observado casos como por ejemplo la ciudad de Madrid con puntos de interés a 400km. Seguramente esto se debe a que Foursquare no tiene registradas ciudades del norte de España.

Las ciudades seleccionadas para este trabajo han sido Nueva York, Tokio y Madrid. En la figura 3.8 se muestra un ejemplo de la estructura de datos obtenida para la ciudad de Nueva York.

Venue ID	Latitude	Longitude	Venue category name	Checkins	Distance to city centre
3fd66200f964a52008e81ee3	40.76	-73.97	Dessert Shop	139	7.85
3fd66200f964a52008e91ee3	40.74	-73.99	Cafe	87	7.61
3fd66200f964a52008eb1ee3	40.78	-73.95	Bar	30	9.25
3fd66200f964a52000e81ee3	40.76	-73.98	Athletic & Sport	24	8.16
3fd66200f964a52001e91ee3	40.73	-73.95	Asian Restaurant	0	4.88

Figura 3.8: Muestra de la estructura de datos una vez procesados.

Una vez tenemos los datos preparados para trabajar con ellos, calculamos la matriz de valoraciones con la que el recomendador lleva a cabo el entrenamiento. Para obtener dicha matriz de valoraciones normalizamos los *checkins* como se ha explicado en la subsección 2.2.1. Así, a partir de la ecuación 3.1 otorgamos un rating a cada punto de interés. La figura 3.9 muestra un ejemplo de matriz de valoraciones.

$$rating = \frac{\log(num\_checkins)}{\log(limit)} \cdot 10 \tag{3.1}$$

Siendo $limit$ el número de *checkins* que marca la puntuación máxima, es decir, un 10. De esta manera los *ratings* de la matriz de valoraciones están en el intervalo [1, 10].

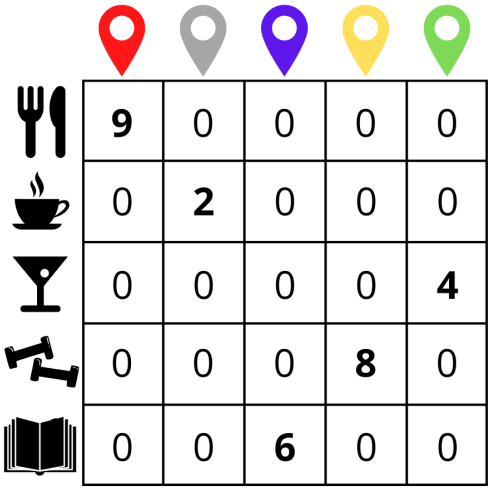


Figura 3.9: Ejemplo de matriz de valoraciones normalizando los *checkins*.

3.2.2. Cálculo de las características

Una vez procesados los conjuntos de datos, como se explica en la subsección 3.2.1, se calculan una serie de características las cuales son utilizadas por los modelos para entrenar y predecir sus recomendaciones. En este trabajo se han utilizado las mismas características de localización y de comercio que en el artículo original [26], las cuales se describen a continuación.

Características de localización

Las características de localización tratan de describir la ubicación de un punto de interés a partir de diferentes métricas, ya que un lugar puede ser bueno o malo para un negocio dependiendo de la zona. Algunas de estas métricas son:

Distancia al centro de la ciudad: para saber si la zona se encuentra cerca o lejos del centro, ya que, por lo general, las zonas cercanas al centro de la ciudad tienen mayor influencia.

$$DC = \frac{1}{\log(d)} \tag{3.2}$$

Siendo d la distancia al centro de la ciudad y DC el efecto que tiene dicha distancia en la popularidad de una tienda.

Accesibilidad del tráfico: para conocer, por ejemplo, la cercanía a paradas de transporte público, puesto que a mayor accesibilidad más clientes pueden visitar el establecimiento.

$$AT = \sum_{t \in T} \frac{\log_2[n(t,r)+1]}{\log_2[d(t)]} \tag{3.3}$$

Donde t es un tipo del conjunto de transportes T, $n(t,r)$ es el número de facilidades para acceder al transporte t en la zona con radio r (por ejemplo, una parada de autobús) y $d(t)$ es la distancia mínima, siguiendo con el ejemplo, a la parada de autobús.

Diversidad de tiendas: para percibir si es una zona explotada o no por una gran variedad de establecimientos. En caso afirmativo, las características de comercio serán las encargadas de ver si es positivo o negativo. Una gran diversidad de tiendas puede marcar una zona donde la demanda está servida. Sin embargo, una diversidad baja puede significar una zona residencial o en crecimiento.

$$DT = -\sum_{s \in S} [\frac{n(s,r)}{n(r)} \times \log \frac{n(s,r)}{n(r)}] \tag{3.4}$$

Siendo s el tipo del conjunto de tiendas S, $n(s,r)$ el número de tiendas de tipo s en la zona con radio r y $n(r)$ el número total de tiendas en la zona con radio r.

Popularidad: para calcular la densidad de personas que transitan por la zona, ya que cuanto mayor número de clientes en la zona, mayor probabilidad de que el negocio tenga éxito.

$$P = \log \sum_{s \in R} C(s) \tag{3.5}$$

Donde s es una tienda del conjunto de tiendas de la zona R y $C(s)$ es el número de *checkins* de la tienda s.

Características de comercio

Como hemos mencionado anteriormente, las características de comercio buscan predecir si las tiendas de la zona afectan de manera positiva o negativa a una nueva tienda. Por ejemplo, una biblioteca y una cafetería se influenciarían de manera positiva, ya que los clientes de un establecimiento muy posiblemente pasen por el otro.

Competitividad: para conocer si las tiendas de la zona son competencia o no de la nueva, puesto que si hay establecimientos de la misma categoría en la misma zona, tendrán que competir por los clientes.

$$C_s = \frac{n(s,r)}{n(r)} \tag{3.6}$$

Siendo s la tienda del punto de interés del que se están calculando las características, $n(s,r)$ el número de tiendas del tipo s en la zona con radio r y $n(r)$ el número total de tiendas en la zona con radio r.

Complementariedad: para saber si los establecimientos de la zona se complementan, es decir, se influencian de manera positiva o no con la nueva tienda.

$$a_{s \to s^*} = \frac{2 \cdot n(s,s^*)}{n \cdot (n+1)} \tag{3.7}$$

Donde $n(s,s^*)$ es el número de apariciones que tiene el par de tiendas s y s^* en las distintas zonas, n es el número total de tiendas y $a_{s \to s^*}$ mide la probabilidad de que la tienda s y s^* aparezcan en la misma zona. Se supone que si un par de tiendas tiene un alto número de apariciones es porque son complementarias.

3.2.3. Descripción del recomendador

El recomendador implementado utiliza la factorización de matrices para predecir las recomendaciones como se ha explicado a lo largo del documento. En esta subsección se va a explicar cómo se han desarrollado y adaptado las técnicas y los componentes empleados.

Factorización de matrices

En primer lugar, aplicamos la factorización matricial para modelar las relaciones de las localizaciones y los tipos de tiendas. Así, a partir de la técnica matemática SVD creamos representaciones de las localizaciones y las categorías de tienda mediante factores latentes de k dimensiones. De esta manera, como se puede observar en la ecuación 3.8, se puede predecir el *rating* que tendría una localización l con el tipo de tienda t.

$$\hat{r}(t,l) = T_t^T L_l \tag{3.8}$$

Siendo T_t el vector de factores latentes correspondiente al tipo de tienda t y L_l el vector de factores latentes correspondiente a la localización l.

Según se ha explicado en la subsección 2.1.2, los factores latentes se van ajustando durante el entrenamiento según el error del *rating* predicho y el *real*. Es decir, se va minimizando el error a partir del algoritmo de optimización descenso por gradiente y, por lo tanto, mejorando las predicciones.

El *rating* real, con el que se compara el *rating* predicho, lo adquirimos de la matriz de valoraciones obtenida durante el procesado de datos y mediante la normalización de los *checkins* (subsección 3.2.1).

Incorporación de características y sesgos

Al igual que en el artículo original [26], y basándonos en la factorización matricial básica, integramos las características de localización y de comercio calculadas en la subsección 3.2.2. Para ello es necesario incorporar también los sesgos dentro de la predicción. La ecuación 3.9 muestra un ejemplo de cómo se predice el *rating* de una localización con un tipo de tienda integrando tanto las características como los sesgos mencionados.

$$\hat{r}(t,l) = b + B(t,l)F(t,l) + P(t,l)C(t,l) + T_t^T L_l \tag{3.9}$$

Donde b es un sesgo global con un valor constante que coincide con la valoración media, ya que cuando se abre una tienda nueva hay un 50 % de probabilidad de que sea popular o no. $B(t,l)$ y $P(t,l)$ son los vectores de los sesgos para el tipo de tienda t y la localización l. Estos los utilizamos, como hemos mencionado anteriormente, para integrar las características de comercio $F(t,l)$ y las características de localización $C(t,l)$.

Al igual que los factores latentes, y según se ha explicado en la subsección 2.1.2, los sesgos se van ajustando durante el entrenamiento a partir del algoritmo de optimización descenso por gradiente.

En la tabla 3.2 se puede observar un resumen de las notaciones matemáticas utilizadas para llevar a cabo la implementación del recomendador. Como se puede apreciar, la notación del vector de características de localización y de comercio tiene tanto la localización l y el tipo de tienda t, aunque no coincide con el tamaño de su matriz de características (S$\times$4 y S$\times$2 respectivamente). Esto se debe a que las características se calculan a partir de la localización l y el tipo de tienda t, pero la matriz se indexa por la localización.

Aprendizaje de características latentes y sesgos para nuevas tiendas

Una vez entrenado el modelo, hay un inconveniente, y es que no se tiene ni las características latentes ni los sesgos para predecir el *rating* de las nuevas localizaciones para el tipo de tienda introducido. Para solucionar el problema se ha implementado el algoritmo 3.1, que muestra cómo predecir el ránking de localizaciones para el tipo de tienda indicado.

Símbolo	Tamaño	Descripción
L	S $\times$ K	Factores latentes de las localizaciones
T	M $\times$ K	Factores latentes de los tipos de tiendas
L_l	1 $\times$ K	Vector de factores latentes de la localización l
T_t	1 $\times$ K	Vector de factores latentes del tipo de tienda t
P	S $\times$ M $\times$ 4	Sesgos de las localizaciones
B	S $\times$ M $\times$ 2	Sesgos de los tipos de tiendas
$P(t,l)$	1 $\times$ 4	Vector de los sesgos de la localización l y el tipo de tienda t
$B(t,l)$	1 $\times$ 2	Vector de los sesgos del tipo de tienda t y la localización l
C	S $\times$ 4	Características de localización
F	S $\times$ 2	Características de comercio
$C(t,l)$	1 $\times$ 4	Vector de caract. de localización del tipo de tienda t y la localización l
$F(t,l)$	1 $\times$ 2	Vector de caract. de comercio del tipo de tienda t y la localización l

Tabla 3.2: Anotaciones matemáticas para el recomendador implementado.

input : Tipo de tienda t
output: Ránking de localizaciones según el rating predicho

$muestras \leftarrow$ POIs del entrenamiento con $rating > 0$ y tipo de tienda t;
for m *in* $muestras$ **do**
  $clase[m.rating] \leftarrow m$;
end
foreach l *in* $localizaciones$ **do**
  $l[tipo\_tienda] \leftarrow t$;
  $C_l \leftarrow DC_l, AT_l, DT_l, P_l$;
  $F_l \leftarrow Compet_l, Complem_l$;
  for $r \leftarrow 1$ **to** 10 **do**
    $dist[r] \leftarrow \sum_{c \in clase[r]} \frac{dist\_euclidea(C_l,C_c)+dist\_euclidea(F_l,F_c)}{long(clase[r])}$;
  end
  $clase\_elegida \leftarrow clase[min(dist)]$;
  $nuevos\_parametros \leftarrow \frac{clase\_elegida.parametros}{long(clase\_elegida)}$;
  $rating\_predicho \leftarrow predecir\_rating(C_l, F_l, nuevos\_parametros)$;
  $ranking \leftarrow$ añadir la localización l con su $rating\_predicho$;
end

Algoritmo 3.1: Pseudocódigo para el aprendizaje de características latentes y sesgos para nuevas tiendas.

En primer lugar, se clasifican según su *rating* los POIs del entrenamiento con *rating* mayor que cero y tipo de tienda igual que el introducido. Posteriormente, por cada localización se obtienen sus características latentes y sus sesgos en base a los ya calculados de la clase con mayor similitud. Finalmente, se hace un ránking de las localizaciones y su *rating* predicho para el supuesto caso de que se montase el tipo de tienda indicado.

3.2.4. Visualización

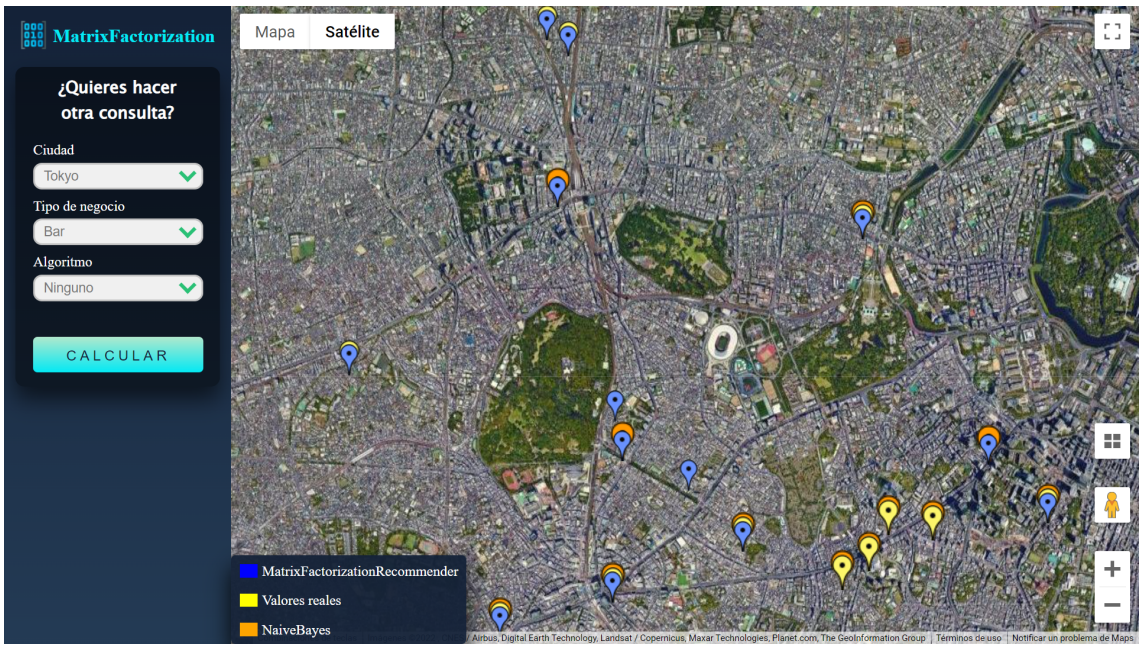
Con el objetivo de hacer más amigable la recomendación, se ha creado una maqueta de la aplicación web a partir de *Flask*. La decisión de utilizar el *framework Flask* se debe a que el trabajo ha sido implementado en *Python* y ya se conocía de antes. La parte visual de la *app* se ha llevado a cabo mediante los *templates Jinja2*, HyperText Markup Language (HTML) , Cascading Style Sheets (CSS) y JavaScript (JS) . JS ha sido utilizado para realizar peticiones a la Application Programming Interfaces (API) de *Google Maps* y así poder mostrar el mapa con las recomendaciones.

En primer lugar, como se puede observar en la figura 3.10, la aplicación web muestra la página principal con nuestro eslogan. Seguidamente, haciendo *scroll* y dejando de lado la página principal, se muestra un formulario como el de la figura 3.11 donde el usuario puede realizar su consulta.

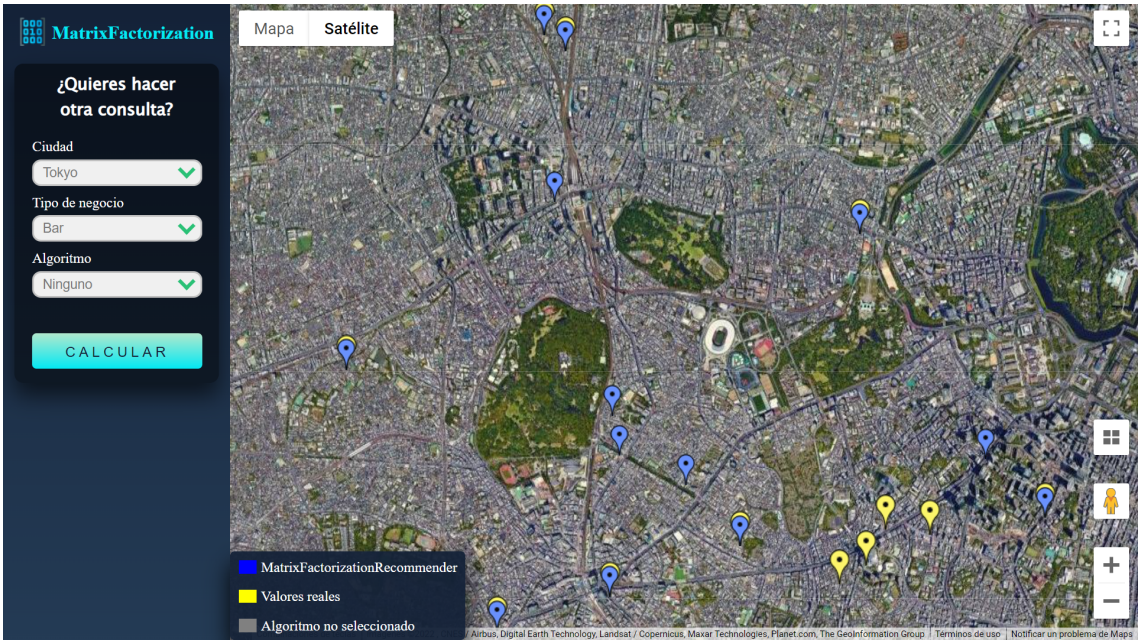
Finalmente, en la figura 3.12 se puede apreciar la recomendación del recomendador implementado con y sin la recomendación del algoritmo elegido para comparar resultados.

Figura 3.10: Página principal de la aplicación web.

Figura 3.11: Formulario para solicitar la recomendación deseada.



(a) Visualización de la recomendación con el algoritmo elegido para comparar.



(b) Visualización de la recomendación sin el algoritmo elegido para comparar.

Figura 3.12: Visualización de la recomendación.

4

Pruebas y resultados

En este capítulo se estudia el rendimiento del recomendador implementado en comparación con otros modelos. Para ello, en primer lugar, se analiza el entorno de pruebas con el que se ha trabajado. Posteriormente, se explican algunos asuntos a tener en cuenta sobre los datos. Antes de realizar las pruebas, se describe cómo se ha realizado el ajuste de hiperparámetros de los diferentes modelos. Finalmente, se comentan las pruebas realizadas y se muestra un análisis de los resultados.

4.1. Entorno de pruebas

Las diferentes pruebas realizadas en el trabajo se han hecho en el entorno de Google Colab Pro con las especificaciones que se pueden observar en la tabla 4.1. Algunos de los incovenientes que tiene dicho entorno es que se comparten los recursos, aunque se tenga la versión Pro, y las sesiones tienen un tiempo limitado. Esto último nos influye desfavorablemente, ya que nuestras pruebas tiene un alto coste en tiempo de ejecución. A pesar de ello, se ha decidido utilizar el entorno de Google Colab Pro porque tiene mejores especificaciones que el entorno local. Por ello, se han tenido que hacer algunas adaptaciones de los datos como se explica en la subsección 4.2.

Recursos	Características
CPU	Intel(R) Xeon(R)
GPU	NVIDIA Tesla K80/T4/P100
RAM	13GB
S.O.	Ubuntu 18.04.5 LTS
Python	3.7.13

Tabla 4.1: Especificaciones del entorno de pruebas Google Colab Pro.

4.2. Adaptación de los datos

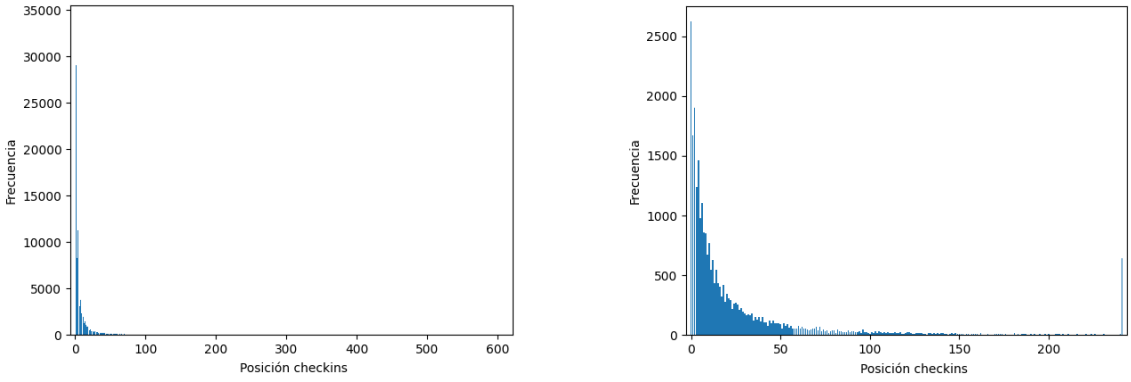
Tras el procesado de los datos para poder trabajar con ellos, explicado en la subsección 3.2.1, cabe desarrollar la adaptación que se ha hecho de los mismos para solucionar algunos inconvenientes con la recomendación o con el entorno de pruebas.

Antes de entrar en materia, es preciso señalar que dado que nuestro trabajo se centra en la recomendación de lugares comerciales en una ciudad, se ha decidido elegir tres ciudades del mundo para examinar el rendimiento del recomendador implementado. Estas ciudades son Nueva York, Tokio y Madrid, y se han elegido porque se encuentran entre el top ciudades con mayor número de puntos de interés. Además, puede ser interesante comparar los resultados de Nueva York, por un lado, con Madrid para ver cómo se comporta el recomendador con ciudades con un menor número de datos. Por otro lado, con Tokio, ya que al ser una sociedad diferente a la occidental seguro que su población tiene patrones diferentes. La tabla 4.2 muestra el número de POIs y *checkins* que tiene cada ciudad.

Ciudad	POIs	Checkins
Nueva York	95 932	765 183
Tokio	174 463	2 440 946
Madrid	82 149	288 004

Tabla 4.2: Datos de las ciudades elegidas.

Volviendo al tema principal de la sección, uno de los problemas a la hora de realizar la recomendación es la diferencia de *checkins* que tienen unas tiendas y otras. Para evitar los sesgos de escasez y popularidad, mencionados en la subsección 2.1.4, se ha marcado un límite inferior y superior de *checkins*, es decir, se han eliminado los POIs con un número de *checkins* menor al límite inferior y se ha restringido el número máximo de *checkins*. La figura 4.1 muestra la distribución del número de *checkins* antes y después de aplicar dichos límites.



(a) Distribución original.

(b) Distribución aplicando límites.

Figura 4.1: Distribución original del número de *checkins* 4.1(a) y aplicando límites 4.1(b).

Como hemos mencionado en la sección 4.1, uno de los inconvenientes del entorno de pruebas es el tiempo limitado de las sesiones. Por ello, para poder realizar pruebas con el conjunto de datos de Tokio se ha tenido que coger una muestra de POIs y no el conjunto completo. No obstante, como se puede observar en la tabla 4.3, la muestra tiene un tamaño superior al conjunto de datos de Nueva York o Madrid, por lo que es lo suficientemente grande para efectuar pruebas y poder sacar conclusiones de ellas.

Ciudad	POIs	Checkins
Nueva York	13 558	532 085
Tokio	20 000	812 177
Madrid	4 434	151 410

Tabla 4.3: Datos de las ciudades elegidas tras su adaptación.

Una vez se tienen los datos con los que realizar las pruebas, se dividen en entrenamiento y test. El modelo lleva a cabo su entrenamiento a partir de los datos de entrenamiento y se estudia su rendimiento mediante los datos de test.

Las pruebas de rendimiento se han realizado sobre tres conjuntos formados por cinco tipos de tiendas, los cuales se han seleccionado en base al número de tiendas de cada tipo que hay en las diferentes ciudades. Los conjuntos seleccionados son:

Primer conjunto: Bar, Cafe, Dessert Shop, Food & Drink Shop, Fast Food Restaurant.

Segundo conjunto: Office, Bank, Bar, Cafe, Fast Food Restaurant.

Tercer conjunto: Athletic & Sport, Hotel, Clothing Store, Food & Drink Shop, Bar.

A partir de dichos conjuntos los algoritmos deben ser capaces de crear un ránking con las localizaciones que recomiendan para montar cada tipo de tienda. Respecto a esto, cabe destacar la adaptación que se ha realizado con la recomendación de los algoritmos de *Scikit-learn* utilizados para las pruebas. Para no añadir una mayor complejidad a la pruebas, y como para recomendar localizaciones se tiene que recomendar dos parámetros, es decir, la latitud y la longitud, se ha obtenido un ránking de tipos de tiendas por cada localización y en base a dichos ránkings se ha generado el ránking de localizaciones para cada tipo de tienda.

4.3. Ajuste de hiperparámetros

Antes de realizar los experimentos es conveniente llevar a cabo el ajuste de hiperparámetros de los diferentes modelos. De esta manera, se saca un provecho superior de los recomendadores obteniendo así mejores predicciones.

Las pruebas para configurar los modelos se efectuan para el conjunto de datos de Nueva York, ya que es el conjunto con características promedio entre las diferentes ciudades elegidas. Además, se utiliza el primer conjunto de tipos de tiendas mencionado en la sección anterior 4.2. La división de datos entre entrenamiento y test es de un 70 % y 30 % respectivamente. No obstante, aunque durante el entrenamiento se utilizan todos los datos, en el test se emplean los datos de test que pertenecen al conjunto de tipos de tiendas elegido, es decir, el primero.

Las métricas usadas durante la evaluación son las explicadas en la sección 2.3. Estas nos permiten analizar el rendimiento de los diferentes modelos.

El primer modelo en ser estudiado es el recomendador implementado, el cual se compone de cuatro hiperparámetros: el número de épocas, el número de dimensiones latentes, la tasa de aprendizaje y un parámetro regularizador. El valor óptimo de dichos hiperparámetros ha sido elegido en base a los valores en los que se reducía el MAE y RMSE sin afectar a la predicción, es decir, aquellos valores con los que no se producía sobreentrenamiento.

La figura 4.2 y la tabla 4.4 muestran tanto la evolución del error como el valor de las diferentes métricas de evaluación para los diferentes números de épocas. En base a los resultados, el valor elegido es 500 épocas, ya que el error se ve reducido al mínimo y la predicción no se ve afectada. No obstante, como la predicción no se ve afectada, también tendría sentido utilizar 100 épocas, porque así el algoritmo sería más rápido.

Número de épocas	R-Precision	MAP	MRR
100	0.22	0.23	0.37
200	0.22	0.23	0.37
300	0.22	0.23	0.47
400	0.22	0.23	0.47
500	0.22	0.23	0.37

Tabla 4.4: Métricas de evaluación para los diferentes números de épocas.

El siguiente hiperparámetro a evaluar es el número de dimensiones latentes. Como se puede observar en la figura 4.3, para el tamaño 20 el error crece ligeramente (recordar que RMSE crece de manera más bruca debido a que penaliza más los errores) y la predicción se encuentra en el punto máximo en el que comienza a mantenerse.

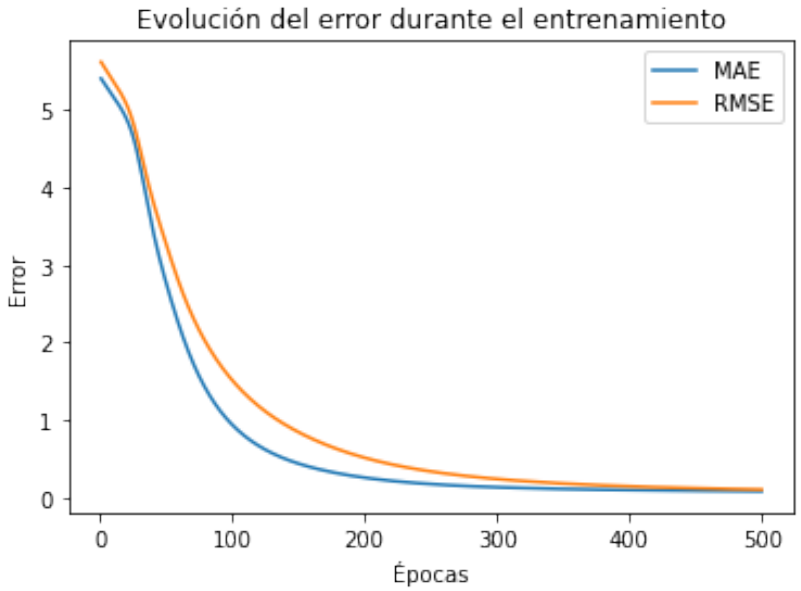


Figura 4.2: Evolución del error para los diferentes números de épocas.

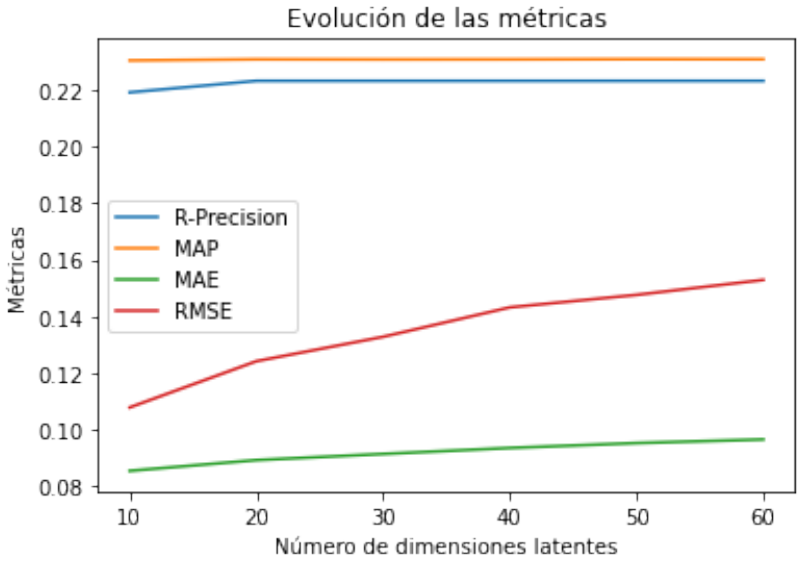


Figura 4.3: Evolución de las métricas para las diferentes dimensiones latentes.

Posteriormente, como se puede apreciar en la tabla 4.5(a), se examinan diferentes números racionales para la tasa de aprendizaje. De acuerdo con los resultados obtenidos, se considera que el mejor es 0.01, ya que se reduce de manera significativa el error, aunque la predicción es muy similar a la de 0.001. A partir de este valor, con el objetivo de ajustar más el hiperparámetro, se han hecho más pruebas cambiándolo ligeramente. Por lo tanto, en base a los resultados de la tabla 4.5(b), se ha seleccionado el valor 0.03, porque se da mayor importancia a los valores de la predicción que a los del error. Un valor bajo del error puede ser bueno o no, es decir, puede ser que se esté produciendo sobreentrenamiento, por lo que hay que buscar el equilibrio entre los valores de la predicción y del error.

Tasa	R-Precision	MAP	MAE	RMSE
1.0000	0.1681	0.2313	inf	inf
0.1000	0.1750	**0.2318**	inf	inf
0.0100	**0.2234**	0.2308	**0.0300**	**0.0324**
0.0010	**0.2234**	0.2309	0.0887	0.1214
0.0001	0.2032	0.2231	2.8620	3.3778

(a) Resultados de las primeras pruebas con diferentes números racionales.

Tasa	R-Precision	MAP	MAE	RMSE
0.0100	**0.2234**	0.2307	0.0299	0.0323
0.0200	**0.2234**	0.2308	0.0230	0.0252
0.0300	**0.2234**	**0.2311**	0.0204	0.0224
0.0400	**0.2234**	0.2305	0.0193	0.0213
0.0500	**0.2234**	0.2307	**0.0188**	**0.0210**

(b) Resultados de las segundas pruebas variando el mejor valor de las primeras.

Tabla 4.5: Resultados de las pruebas con distintos valores de tasa de aprendizaje. En negrita se encuentran los mejores resultados, es decir, para la predicción los más altos y para el error los más bajos.

Siguiendo la misma técnica utilizada para ajustar el valor de la tasa de aprendizaje, se obtienen para el parámetro regularizador los resultados de la tabla 4.6. A partir de ellos, se elige el valor 0.1, puesto que para *R-Precision* se obtiene el valor máximo, MAP tiene un valor similar al resto y el error se ve reducido. La decisión de dar tanta importancia a la métrica *R-Precision* viene dada porque, en el caso de utilizar el recomendador en producción, nos interesa que acierte el máximo de localizaciones dentro del top del ránking.

Parámetro	R-Precision	MAP	MAE	RMSE
1.0000	**0.2277**	0.2301	0.9156	1.0004
0.1000	**0.2277**	0.2305	0.0911	0.1001
0.0100	0.2234	0.2308	0.0125	0.0137
0.0010	0.2234	0.2309	0.0029	0.0032
0.0001	0.2234	**0.2310**	**0.0004**	**0.0005**

(a) Resultados de las primeras pruebas con diferentes números racionales.

Parámetro	R-Precision	MAP	MAE	RMSE
0.1000	**0.2277**	0.2305	**0.0908**	**0.0994**
0.2000	**0.2277**	0.2303	0.1835	0.2021
0.3000	**0.2277**	0.2302	0.2774	0.3051
0.4000	**0.2277**	**0.2311**	0.3685	0.4051
0.5000	**0.2277**	**0.2311**	0.4602	0.5049

(b) Resultados de las segundas pruebas variando el mejor valor de las primeras.

Tabla 4.6: Resultados de las pruebas con distintos valores del parámetro regularizador. En negrita se encuentran los mejores resultados, es decir, para la predicción los más altos y para el error los más bajos.

Por último, realizamos diferentes pruebas con los hiperparámetros de la tabla 4.7. El objetivo es configurar los modelos de *Scikit-learn* con los que posteriormente realizar la comparación de rendimiento. El criterio utilizado para seleccionar un valor u otro ha sido el utilizado hasta ahora. La tabla 4.8 muestra la configuración final elegida para realizar los experimentos.

Los algoritmos de *Scikit-learn* escogidos para realizar dichas pruebas de rendimiento son los más populares. Estos son Support Vector Machine (SVM) , Random Forest, Regresión logística, Naive Bayes, K-Nearest Neighbors (KNN), Perceptrón multicapa y predicción aleatoria. Además, cabe señalar que algunos de estos algoritmos coinciden con los que se usan en el TFM [25] y el artículo [26], aunque en este trabajo se han implementado e integrado desde cero para la nueva tarea de recomendación.

Modelo	Parámetros
SVM	kernel=['linear', 'poly', 'rbf', 'sigmoid'], gamma=['scale', 'auto'], C=[0.01, 0.1, 0.5, 1, 3, 6], class_weight=['balanced', None], decision_function_shape=['ovo', 'ovr']
Random Forest	n_estimators=range(50, 801, 50), criterion=['gini', 'entropy'], max_depth=[range(1, 50, 2), None], max_features=['sqrt', 'log2', None], bootstrap=[False, True], warm_start=[False, True]
Regresión logística	multi_class=['auto', 'ovr', 'multinomial'], max_iter=range(100, 2001, 100), warm_start=[True, False], penalty=['elasticnet', 'l1', 'l2', 'none'], solver=['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
KNN	n_neighbors=range(1, 51), weights=['uniform', 'distance'], p=[1, 2, 3]
Perceptrón multicapa	hidden_layer_sizes=(range(50, 1001, 50), range(50, 1001, 50)), activation=['identity', 'logistic', 'tanh', 'relu'], solver=['lbfgs', 'sgd', 'adam'], learning_rate_init=[1, 0.1, 0.01, 0.001, 0.0001], shuffle=[True, False] max_iter=range(100, 2001, 100), warm_start=[True, False]
Aleatorio	strategy=['stratified', 'uniform']

Tabla 4.7: Posibles valores de los hiperparámetros de los modelos.

Modelo	Parámetros
SVM	kernel='sigmoid', gamma='auto', C=0.5, class_weight='balanced', decision_function_shape='ovr'
Random Forest	n_estimators=300, criterion='entropy', max_depth=3, max_features='log2', bootstrap=False, warm_start=False
Regresión logística	multi_class='auto', max_iter=100, warm_start=False, penalty='none', solver='lbfgs'
KNN	n_neighbors=7, weights='distance', p=2
Perceptrón multicapa	hidden_layer_sizes=(800, 50), activation='logistic', solver='adam', learning_rate_init=0.001, shuffle=False max_iter=1100, warm_start=True
Aleatorio	strategy='uniform'

Tabla 4.8: Elección final de los hiperparámetros de los modelos.

4.4. Experimentos y análisis de los resultados

Una vez ajustados los hiperparámetros de los modelos, se llevan a cabo las pruebas de rendimiento para las tres ciudades y los tres conjuntos de tipos de tiendas. El objetivo es comparar el recomendador implementado con otros algoritmos populares. A continuación en la tabla 4.9 se muestran los resultados obtenidos para las pruebas de las tres ciudades con el primer conjunto de tipos de tiendas. El resto de pruebas con los conjuntos de tipos de tiendas restantes se encuentran en el apéndice A.

Ciudad	Modelo	R-Precision	MAP	MRR	Tiempo de ejecución (seg.)
Nueva York	**Factorización de matrices**	0.24	0.22	0.42	12 372.31
	SVM	**0.27**	0.24	0.39	9 932.84
	Random Forest	0.23	**0.26**	0.47	9 808.13
	Regresión logística	0.22	0.22	0.44	9 771.42
	Naive Bayes	0.25	0.24	0.56	9 754.07
	KNN	0.16	0.21	**0.63**	9 755.51
	Perceptrón multicapa	0.23	0.23	0.45	10 529.22
	Aleatorio	0.15	0.22	0.38	**9 753.69**
Tokio	**Factorización de matrices**	0.22	0.21	0.34	27 284.38
	SVM	**0.24**	**0.24**	0.35	23 778.35
	Random Forest	0.23	**0.24**	0.14	23 182.84
	Regresión logística	0.21	0.23	0.45	23 134.42
	Naive Bayes	0.19	0.21	0.19	**23 114.70**
	KNN	0.20	0.21	0.39	23 114.84
	Perceptrón multicapa	0.20	0.20	**0.52**	24 045.01
	Aleatorio	0.19	0.21	0.31	23 114.82
Madrid	**Factorización de matrices**	0.20	0.27	0.52	1 229.66
	SVM	0.24	0.28	0.49	788.35
	Random Forest	0.27	0.28	**0.54**	777.26
	Regresión logística	**0.29**	0.28	0.49	772.84
	Naive Bayes	0.21	0.24	0.34	767.49
	KNN	0.19	0.22	0.36	767.47
	Perceptrón multicapa	0.26	**0.30**	0.42	1 050.42
	Aleatorio	0.18	0.23	0.27	**767.46**

Tabla 4.9: Resultados de las pruebas de rendimiento con el primer conjunto de tipos de tiendas. En negrita se encuentran los mejores resultados, es decir, para la predicción los más altos y para el tiempo de ejecución los más bajos.

Como se puede observar en la tabla 4.9, el algoritmo implementado tiene un porcentaje de acierto entre el 20 y el 24 %. No obstante, hay pruebas como las de Madrid con el segundo conjunto de tipos de tiendas, es decir, las de la tabla A.1, que logran alcanzar un 28 % de acierto. Esto es algo a remarcar, porque, por lo general, para Madrid se suelen obtener peores resultados debido a que se tienen menos datos con los que trabajar. Teniendo en cuenta la complejidad del problema (algo que se puede apreciar en la tabla 4.9 al observar los valores tan bajos de acierto que se obtienen cuando se predice de manera aleatoria) son cifras razonablemente buenas. Además, a nuestro favor, comentar que, por lo general, los modelos de *Scikit-learn* utilizados logran métricas similares. En la misma línea, estudios con otras implementaciones han obtenido cifras parecidas [28]. Sin embargo, un punto en contra es su alto coste en tiempo de ejecución, por lo que se debería optimizar el algoritmo.

Comparando los resultados con los del TFM [25] o los del artículo original [26] de donde se ha extraído la idea de nuestro trabajo, y los cuales implementan el otro tipo de recomendación, es decir, en base a una localización recomiendan el tipo de tienda que montar, los resultados de nuestro recomendador son similares o ligeramente peores. A nuestro favor, cabe señalar la diferencia de complejidad que hay en cada problema, ya que no es lo mismo predecir qué tipo de tienda montar entre las 5 opciones que hay en los conjuntos utilizados, que predecir qué localizaciones son las mejores entre miles de opciones.

Por último, explicar que la predicción aleatoria es igual de lenta que el resto de algoritmos debido a que se está teniendo en cuenta el cálculo de las características de localización y de comercio. Además, cabe destacar, y que podría ser interesante estudiar como trabajo futuro, que hay situaciones en las que la predicción aleatoria no es siempre la peor, y, por si fuera poco, llega a obtener el mejor valor para MAP. Esto se puede observar en los resultados de las pruebas de la tabla A.2.

5

Conclusiones y trabajo futuro

5.1. Conclusiones

Los sistemas de recomendación son herramientas que están presentes en nuestro día a día. Cada vez que consumimos algún servicio hacemos uso indirectamente de esta herramienta. Por ello, debido a su gran potencial, están en constante evolución y se utilizan en diferentes ámbitos. En nuestro caso, el recomendador implementado está enfocado al mundo empresarial.

En este trabajo se ha desarrollado un sistema de recomendación que, dada una tienda a montar, recomienda las localizaciones que predice como mejores. Para la implementación del algoritmo nos hemos basado en el artículo original [26], el cual utiliza el filtrado colaborativo, y más concretamente la factorización de matrices, junto con una serie de características de localización y de comercio calculadas a partir de la información de cada POI obtenida de los LBS. Los conjuntos de datos utilizados tanto para el entrenamiento del modelo como para las pruebas con otros algoritmos han sido los proporcionados por Foursquare.

A partir de las pruebas de rendimiento realizadas se ha podido observar que el modelo es capaz de ajustar las características latentes y los sesgos, aunque los datos estén muy dispersos, alcanzando así métricas muy similares a otros algoritmos de *Scikit-learn* u otros estudios con diferentes implementaciones [28].

Por último, tener en cuenta que la eficacia del recomendador implementado se ha visto condicionada, por un lado, por el número de datos disponible de cada ciudad. Por otro lado, por el tiempo limitado de las sesiones del entorno de pruebas, lo que ha tenido como consecuencia la necesidad de minimizar el número de datos a usar de algunas ciudades.

Finalmente, me gustaría señalar que todo el procesado de datos, el código fuente del recomendador, las pruebas realizadas y la aplicación web se encuentran disponibles en el siguiente link. La decisión de utilizar Drive, en vez de GitHub, viene determinada por el almacenamiento de los conjuntos de datos y el uso de notebooks de Jupyter en el entorno de Google Colab Pro.

`https://drive.google.com/drive/folders/1390ek0lB1xREMmZ5StTYYtChT6bmyD3a?usp=sharing`

5.2. Trabajo futuro

A lo largo de esta sección se van a plantear diversos temas que, por falta de tiempo y recursos, no han sido posible tratar, pero que podría ser interesante su desarrollo o investigación. Estos temas podrían ser una ampliación de este Trabajo de Fin de Grado (TFG) .

En primer lugar, sería interesante realizar más pruebas. Por un lado, se podrían utilizar más conjuntos de tipos de tiendas, incluso ampliar el número de tipos de tiendas para así ver cómo se comporta el recomendador. Por otro lado, como sólo se han comparado ciudades de distintos continentes, podría ser interesante examinar ciudades del mismo continente o país. En la misma línea, se podría entrenar el modelo con una ciudad y observar su rendimiento en otras que tengan una cierta relación, por ejemplo, entrenar con Madrid y realizar las predicciones en Barcelona.

Uno de los problemas que hay en el campo de la recomendación son los pocos datos disponibles de manera pública. No obstante, sería interesante utilizar otros conjuntos de datos y, si el entorno lo permite, realizar pruebas sin límite de tamaño.

Como hemos apreciado en la sección de pruebas 4.4, el algoritmo implementado tiene un alto coste en tiempo de ejecución, por lo que sería conveniente optimizarlo. Dado que la mayor parte de este tiempo se emplea en el cálculo de las características, se podría paralelizar el trabajo calculando al mismo tiempo las características de cada POI, o por *batches* de POIs.

Otro tema que podría ser motivador sería estudiar cómo lograrían resolver el mismo problema los sistemas de recomendación basados en memoria o basados en contenido.

Por último, un nuevo reto podría ser mostrar el porqué se ha recomendado una localización, cuáles son las características por las que se ha tomado la decisión, de esta manera el experto podría decidir si tener en cuenta la recomendación o no. Incluso, otro reto podría ser ampliar el recomendador utilizando otro tipo de características, bien de localización y comercio u otras como pueden ser la iluminación o la contaminación. Por una parte, la contaminación lumínica podría complementarse con negocios nocturnos. Por otra parte, la zonas muy contaminadas podrían tratarse de manera positiva por ser zonas muy concurridas o de manera negativa debido a lo perjudicial que es la contaminación para la salud.

Bibliografía

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

[2] L. Candillier, K. Jack, F. Fessant, and F. Meyer, "State-of-the-art recommender systems," in *Collaborative and Social Information Retrieval and Access: Techniques for Improved User Modeling*, pp. 1–22, IGI Global, 2009.

[3] M. D. Ekstrand and J. A. Konstan, "Recommender systems notation: Proposed common notation for teaching and research," *CoRR*, vol. abs/1902.01348, 2019.

[4] L. Terveen and W. Hill, "Beyond recommender systems: Helping people help each other," *HCI in the New Millennium*, vol. 1, no. 2001, pp. 487–509, 2001.

[5] Y. Koren, R. M. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[6] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008* (Y. Li, B. Liu, and S. Sarawagi, eds.), pp. 426–434, ACM, 2008.

[7] H. Polat and W. Du, "Svd-based collaborative filtering with privacy," in *Proceedings of the 2005 ACM Symposium on Applied Computing (SAC), Santa Fe, New Mexico, USA, March 13-17, 2005* (H. Haddad, L. M. Liebrock, A. Omicini, and R. L. Wainwright, eds.), pp. 791–795, ACM, 2005.

[8] Y. Koren and R. M. Bell, "Advances in collaborative filtering," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 77–118, Springer, 2015.

[9] X. Ning, C. Desrosiers, and G. Karypis, "A comprehensive survey of neighborhood-based recommendation methods," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 37–76, Springer, 2015.

[10] C. Zeng, C. Xing, and L. Zhou, "Similarity measure and instance selection for collaborative filtering," in *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003* (G. Hencsey, B. White, Y. R. Chen, L. Kovács, and S. Lawrence, eds.), pp. 652–658, ACM, 2003.

[11] P. Lops, M. de Gemmis, and G. Semeraro, "Content-based recommender systems: State of the art and trends," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, eds.), pp. 73–105, Springer, 2011.

[12] A. Suglia, C. Greco, C. Musto, M. de Gemmis, P. Lops, and G. Semeraro, "A deep architecture for content-based recommendations exploiting recurrent neural networks," in *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization, UMAP 2017, Bratislava, Slovakia,*

July 09 - 12, 2017 (M. Bieliková, E. Herder, F. Cena, and M. C. Desmarais, eds.), pp. 202–211, ACM, 2017.

[13] R. D. Burke, "Hybrid recommender systems: Survey and experiments," *User Model. User Adapt. Interact.*, vol. 12, no. 4, pp. 331–370, 2002.

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

[15] S. D. Barman, M. Hasan, and F. Roy, "A genre-based item-item collaborative filtering: Facing the cold-start problem," in *Proceedings of the 8th International Conference on Software and Computer Applications, ICSCA '19, Penang, Malaysia, February 19-21, 2019*, pp. 258–262, ACM, 2019.

[16] S. Sedhain, S. Sanner, D. Braziunas, L. Xie, and J. Christensen, "Social collaborative filtering for cold-start recommendations," in *Eighth ACM Conference on Recommender Systems, RecSys '14, Foster City, Silicon Valley, CA, USA - October 06 - 10, 2014* (A. Kobsa, M. X. Zhou, M. Ester, and Y. Koren, eds.), pp. 345–348, ACM, 2014.

[17] R. Burke, M. P. O'Mahony, and N. J. Hurley, "Robust collaborative recommendation," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 961–995, Springer, 2015.

[18] E. Pariser, *The filter bubble: What the Internet is hiding from you.* Penguin UK, 2011.

[19] Z. A. Pardos and W. Jiang, "Combating the filter bubble: Designing for serendipity in a university course recommendation system," *CoRR*, vol. abs/1907.01591, 2019.

[20] A. Bellogín, P. Castells, and I. Cantador, "Statistical biases in information retrieval metrics for recommender systems," *Inf. Retr. J.*, vol. 20, no. 6, pp. 606–634, 2017.

[21] Y. Liu, C. Liu, X. Lu, M. Teng, H. Zhu, and H. Xiong, "Point-of-interest demand modeling with human mobility patterns," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pp. 947–955, ACM, 2017.

[22] "Openstreetmap." www.openstreetmap.org.

[23] P. Sánchez and A. Bellogín, "Point-of-interest recommender systems: A survey from an experimental perspective," *CoRR*, vol. abs/2106.10069, 2021.

[24] H. Katsumi, W. Yamada, and K. Ochiai, "Generic POI recommendation," in *UbiComp/ISWC '20: 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and 2020 ACM International Symposium on Wearable Computers, Virtual Event, Mexico, September 12-17, 2020* (M. Tentori, N. Weibel, K. V. Laerhoven, G. D. Abowd, and F. D. Salim, eds.), pp. 46–49, ACM, 2020.

[25] J. Gutiérrez Díaz *et al.*, "Recomendación de localizaciones de negocio usando aprendizaje automático," Master's thesis, 2021.

[26] Z. Yu, M. Tian, Z. Wang, B. Guo, and T. Mei, "Shop-type recommendation leveraging the data from social media and location-based services," *ACM Trans. Knowl. Discov. Data*, vol. 11, no. 1, pp. 1:1–1:21, 2016.

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

[28] J. Griesner, T. Abdessalem, and H. Naacke, "POI recommendation: Towards fused matrix factorization with geographical and temporal influences," in *Proceedings of the 9th ACM Conference on Recommender Systems, RecSys 2015, Vienna, Austria, September 16-20, 2015* (H. Werthner, M. Zanker, J. Golbeck, and G. Semeraro, eds.), pp. 301–304, ACM, 2015.

Acrónimos

AP Average Precision.

API Application Programming Interfaces.

CSS Cascading Style Sheets.

GPS Global Positioning System.

HTML HyperText Markup Language.

JS JavaScript.

KNN K-Nearest Neighbors.

LBS Location-Based Services.

MAE Mean Absolute Error.

MAP Mean Average Precision.

MRR Mean Reciprocal Rank.

POI Point Of Interest.

RMSE Root Mean Squared Error.

RR Reciprocal Rank.

SVD Singular Value Decomposition.

SVM Support Vector Machine.

TFG Trabajo de Fin de Grado.

TFM Trabajo de Fin de Máster.

Apéndices

A

Pruebas restantes

Ciudad	Modelo	R-Precision	MAP	MRR	Tiempo de ejecución (seg.)
Nueva York	**Factorización de matrices**	0.23	0.23	0.38	13 637.90
	SVM	0.22	0.24	0.44	12 132.58
	Random Forest	0.23	0.24	**0.64**	12 006.91
	Regresión logística	0.24	0.23	0.52	11 971.15
	Naive Bayes	**0.25**	**0.26**	0.51	11 953.96
	KNN	0.21	0.21	0.38	11 953.94
	Perceptrón multicapa	0.21	0.23	0.37	12 739.49
	Aleatorio	0.17	0.23	0.33	**11 953.68**
Tokio	**Factorización de matrices**	0.20	0.21	0.31	21 186.69
	SVM	0.24	0.24	0.31	20 286.18
	Random Forest	0.25	**0.25**	0.47	19 697.19
	Regresión logística	**0.27**	**0.25**	0.29	19 638.62
	Naive Bayes	0.21	0.23	0.34	19 629.19
	KNN	0.20	0.22	0.35	**19 629.07**
	Perceptrón multicapa	0.21	0.22	**0.48**	20 447.97
	Aleatorio	0.20	0.21	0.30	19 629.20
Madrid	**Factorización de matrices**	0.28	**0.33**	**0.71**	1 427.60
	SVM	0.28	0.29	0.42	1 308.21
	Random Forest	0.28	0.28	0.30	1 297.75
	Regresión logística	**0.30**	0.30	0.53	1 292.17
	Naive Bayes	0.21	0.24	0.43	1 287.00
	KNN	0.17	0.22	0.26	1 287.02
	Perceptrón multicapa	0.27	0.28	0.50	1 564.01
	Aleatorio	0.20	0.23	0.24	**1 286.91**

Tabla A.1: Resultados de las pruebas de rendimiento con el segundo conjunto de tipos de tiendas. En negrita se encuentran los mejores resultados, es decir, para la predicción los más altos y para el tiempo de ejecución los más bajos.

Ciudad	Modelo	R-Precision	MAP	MRR	Tiempo de ejecución (seg.)
Nueva York	**Factorización de matrices**	**0.23**	0.22	**0.48**	11 578.03
	SVM	0.16	0.17	0.13	10 531.94
	Random Forest	0.13	0.17	0.35	10 401.55
	Regresión logística	0.20	0.20	0.17	10 360.07
	Naive Bayes	0.16	0.19	0.32	**10 353.23**
	KNN	0.18	0.21	0.28	10 354.23
	Perceptrón multicapa	0.13	0.17	0.36	10 950.18
	Aleatorio	0.14	**0.23**	0.32	10 353.42
Tokio	**Factorización de matrices**	0.22	0.23	0.24	19 456.92
	SVM	0.20	0.21	0.35	18 907.40
	Random Forest	0.19	0.22	0.10	18 310.64
	Regresión logística	0.20	**0.25**	**0.38**	18 260.46
	Naive Bayes	**0.23**	**0.25**	0.34	18 242.00
	KNN	0.22	0.21	0.27	18 242.88
	Perceptrón multicapa	0.20	0.23	0.19	19 247.50
	Aleatorio	0.22	0.23	0.24	**18 241.54**
Madrid	**Factorización de matrices**	0.23	0.24	0.25	1 347.90
	SVM	**0.33**	0.29	0.53	1 225.04
	Random Forest	0.26	0.29	0.27	1 214.78
	Regresión logística	0.29	**0.30**	0.65	1 209.32
	Naive Bayes	0.31	0.29	0.24	1 204.05
	KNN	0.23	0.26	**0.66**	**1 203.96**
	Perceptrón multicapa	0.29	0.29	0.54	1 487.81
	Aleatorio	0.19	0.25	0.23	1 204.03

Tabla A.2: Resultados de las pruebas de rendimiento con el tercer conjunto de tipos de tiendas. En negrita se encuentran los mejores resultados, es decir, para la predicción los más altos y para el tiempo de ejecución los más bajos.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID