Universidad Autónoma de Madrid

Escuela Politécnica Superior

22|23

Trabajo fin de grado

Estudio de técnicas de aprendizaje profundo aplicadas a recomendación contextual

Fernando Huidobro Albendea

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 profundo aplicadas a recomendación contextual

Autor: Fernando Huidobro Albendea
Tutor: Alejandro Bellogín Kouki

junio 2023

Fernando Huidobro Albendea
Estudio de técnicas de aprendizaje profundo aplicadas a recomendación contextual

Fernando Huidobro Albendea
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Agradecimientos

En primer lugar me gustaría agradecer a mi tutor Alejandro Bellogín por concerme la oportunidad de realizar este Trabajo de Fin de Grado y por su continua orientación, dedicación y esfuerzo a lo largo de todo el proceso, donde su experiencia, conocimiento y apoyo ha sido fundamental para el desarrollo de este proyecto. También agradecérselo a todo los profesores que he tenido durante mi etapa en la Escuela Politécnica Superior por su contribución en mi formación académica y necesaria para poder ejecutar este trabajo.

Por último, pero no menos importante, expresar mi agradecimiento a mis padres, amigos, compañeros de grado y familia. Su continuo apoyo ha sido fundamental a lo largo de esta etapa universitaria. Han sido mi respaldo constante, no solo en términos académicos, si no también en mi crecimiento personal.

Resumen

En la era digital actual, los sistemas de recomendación se han convertido en una herramienta esencial para muchas empresas que buscan mejorar la experiencia de sus usuarios y aumentar la satisfacción del cliente. Estos sistemas utilizan algoritmos inteligentes para analizar el comportamiento de los usuarios y proporcionar recomendaciones personalizadas de productos o servicios. Sin embargo, pocas empresas tienen en cuenta el contexto de sus usuarios al hacer recomendaciones, a pesar de que son ampliamente utilizados.

El presente trabajo aborda el estudio de técnicas de aprendizaje profundo aplicadas a la recomendación contextual. En este estudio, se exploraron diferentes algoritmos de aprendizaje profundo y su rendimiento en la generación de recomendaciones personalizadas en diferentes contextos. Se destacó en la importancia de considerar el contexto para mejorar la eficacia y precisión de los sistemas de recomendación. Mediante el uso de la biblioteca DeepCARSKit, se realizaron experimentos y evaluaciones de los algoritmos en distintos conjuntos de datos y de escenarios. Se observó que algunos contextos eran relevantes y aportaban información valiosa para la generación de recomendaciones, mientras que otros podían generar ruido y afectar negativamente la precisión. Este hallazgo destaca la importancia de seleccionar cuidadosamente el algoritmo adecuado y realizar pruebas continuas para mejorar el desempeño de los sistemas de recomendación. Además, se identificaron posibles áreas de mejora, como aumentar el número de usuarios en la muestra y realizar una mayor cantidad de experimentos. Se concluye que el estudio de técnicas de aprendizaje profundo aplicadas a la recomendación contextual es fundamental para desarrollar algoritmos de recomendación más precisos y relevantes en un entorno en constante evolución.

Palabras clave

Sistemas de recomendación, aprendizaje profundo, recomendación contextual, aplicación web, DeepCARSKit

Abstract

In the current digital era, recommender systems have become an essential tool for many companies looking to improve their user experience and increase customer satisfaction. These systems use intelligent algorithms to analyze user behavior and provide personalized product or service recommendations. However, few companies take into account the context of their users when making recommendations, despite the fact that they are widely used.

This work addresses the study of deep learning techniques applied to contextual recommendation. In this study, different deep learning algorithms and their performance in generating personalized recommendations in different contexts were explored. The importance of considering the context to improve the efficiency and precision of recommender systems is highlighted. Through the use of the DeepCARSKit library, experiments and evaluations of the algorithms were carried out in different data sets and scenarios. Some contexts were shown to be relevant by providing valuable information for the generation of recommendations, while others could generate noise and negatively affect accuracy. This highlights the importance of carefully selecting the right algorithm and continuous testing to improve the performance of recommender systems. In addition, possible areas for improvement were identified, such as increasing the number of users in the sample and carrying out a larger number of experiments. It is concluded that the study of deep learning techniques applied to contextual recommendation is essential to develop more accurate and relevant recommendation algorithms in a constantly evolving environment.

Keywords

Recommender systems, deep learning, contextual recommendation, web application, DeepCARSKit

Índice

1 Introducción **1**

1.1 Motivación del proyecto 1

1.2 Objetivos 2

1.3 Estructura del trabajo 2

2 Estudio del Arte **3**

2.1 Sistemas de recomendación 3

2.1.1 Recomendación contextual y aprendizaje profundo 4

2.1.2 Aplicaciones de la recomendación de restaurantes y Yelp 4

2.1.3 Evaluación de modelos de recomendación 5

2.1.4 Librerías 6

2.2 Django 8

2.2.1 Django vs Flask 8

2.2.2 ORM Django 9

3 Diseño, Análisis e Implementación **11**

3.1 Diseño 11

3.1.1 Estructura 11

3.1.2 Ciclo de vida 12

3.2 Ánalisis 12

3.2.1 Requisitos funcionales 13

3.2.2 Requisitos no funcionales 14

3.3 Recopilación y preprocesamiento de datos 15

3.4 Modelos de recomendación 16

3.5 Aplicación Web 17

3.5.1 Arquitectura 17

3.5.2 Funcionalidades 19

4 Pruebas y resultados **23**

4.1 Entorno de pruebas 23

4.2 Estadísticas de los datos 24

4.3 Experimentos 26

4.3.1 Comparativa de Algoritmos de Recomendación: Análisis de NeuCF frente a otros enfoques con TripAdvisor 26

4.3.2 Comparativa de Algoritmos de Recomendación: Análisis de NeuCF frente a otros enfoques con Yelp 28
4.3.3 Análisis de los mejores resultados 28
4.3.4 Comparativa de Resultados con Diferentes Contextos 32

5 Conclusiones y trabajo futuro **37**
5.1 Conclusiones 37
5.2 Trabajo Futuro 38

Bibliografía **40**

Acronyms **41**

Apéndices **43**

A Ficheros adicionales **45**

Listas

Lista de algoritmos

Lista de códigos

A.1 Fichero requirements parte 1. 45
A.2 Fichero requirements parte 2. 46
A.3 Fichero requirements parte 3. 47
A.4 Fichero requirements parte 4. 48

Lista de cuadros

Lista de ecuaciones

Lista de figuras

3.1 Estructura del proyecto 12
3.2 Ciclo de vida 13
3.3 Diagrama Patrón MVT 17
3.4 Diagrama Funcionamiento Django 18
3.5 Diagrama de clases 19
3.6 Diagrama de secuencia 20
3.7 Home 22

4.1 Ciudades 24
4.2 Comparación Número de Reseñas 25
4.3 Reseñas de Negocios 26
4.4 Comparativa Algoritmos Recomendación en TripAdvisor 27
4.5 Comparativa Algoritmos Recomendación en Yelp 29
4.6 Comparativa AUC, RMSE y MAE de los algoritmos NeuCMFii y NeuCMF0i 30
4.7 Comparativa rendimiento algoritmo NeuCMFii 31

4.8 Contextos vs No categorias 33
4.9 Sin 2 contextos 34
4.10 Sin Día y Sin Parking 34
4.11 Sin Hora y Popular 35

Lista de tablas

4.1 Tabla características técnicas equipo de pruebas 24

Lista de cuadros

1

Introducción

1.1. Motivación del proyecto

En la actualidad, existen numerosas aplicaciones como Netflix, Amazon Prime, Spotify y YouTube, entre muchas otras, que utilizan sistemas de recomendación para ofrecer sugerencias personalizadas a los usuarios. Estas aplicaciones buscan continuamente nuevas formas de generar recomendaciones que mantengan a los usuarios comprometidos y tengan una experiencia atractiva y útil para ellos mismos.

La oportunidad de mejorar significativamente la experiencia del usuario al brindar recomendaciones más precisas y relevantes se puede lograr mediante la combinación de técnicas de aprendizaje profundo y recomendación contextual. Esto puede aumentar la satisfacción del usuario y fomentar la participación continua con la aplicación, lo que mejora las recomendaciones.

A medida que los sistemas de recomendación basados en aprendizaje profundo han avanzado en los últimos años, los modelos de recomendación conscientes del contexto basados en el filtrado colaborativo tradicional (por ejemplo, CF basado en *K-Nearest Neighbors (KNN)* , factorización matricial) resultaron ser obsoletos, es por ello que surgió DeepCARSKit. Se trata de una biblioteca basada en aprendizaje profundo y compatible con Python y PyTorch, diseñada específicamente para recomendaciones contextuales. Es una actualización sobre CARSKit, la primera librería de código abierto para recomendaciones contextuales y que se lanzó en 2015; sin embargo, desde 2019 no se han realizado actualizaciones relacionadas con esa librería, por lo que el desarrollo de DeepCARSKit tiene mucho interés y potencial futuro [1].

En este proyecto, se realizará un estudio sobre las diferentes técnicas de aprendizaje profundo aplicadas a la recomendación contextual de restaurantes. Se considerarán diferentes contextos relevantes para la recomendación, lo que permitirá generar recomendaciones más personalizadas para cada usuario. También se desarrollará una aplicación web, a través de la cual los usuarios podrán disfrutar de recomendaciones de restaurantes adaptadas a sus preferencias y circunstancias específicas.

1.2. Objetivos

El objetivo principal de este trabajo consiste en el estudio de técnicas de aprendizaje profundo aplicadas a recomendación contextual, con un enfoque específico en la recomendación de restaurantes. En particular, se explorarán y analizarán dos técnicas destacadas: *Factorization Machines (FM)* y *Neural Collaborative Filtering (NeuCF)*.

Además, se llevará a cabo el desarrollo de una aplicación web interactiva, que permitirá a los usuarios visualizar y experimentar las recomendaciones generadas por estos modelos. Esta aplicación web servirá como una plataforma intuitiva y fácil de usar, donde los usuarios podrán explorar diferentes escenarios y contextos, y observar cómo varían las recomendaciones en función de estos factores.

1.3. Estructura del trabajo

Este documento está estructurado de la siguiente manera:

Capítulo 1. Introducción Se presenta la motivación detrás del proyecto, se establecen los objetivos a alcanzar y la estructura del documento.

Capítulo 2. Estado del arte Se explica el marco teórico del proyecto y se exploran los conceptos clave necesarios para comprenderlo.

Capítulo 3. Análisis, diseño e implementación. Se describe el análisis realizado para abordar el problema, las decisiones de diseño tomadas y la implementación del sistema final.

Capítulo 4. Pruebas y resultados. Se explican los experimentos realizados. También se presentan y analizan los resultados obtenidos a partir de las pruebas realizadas.

Capítulo 5. Conclusiones y trabajo futuro. Se presentan las conclusiones obtenidas del trabajo realizado, así como las posibles futuras ampliaciones del proyecto.

2

Estudio del Arte

En este capítulo se explorarán diversos conceptos clvaes para el entendimiento de este trabajo. Se explorarán los sistemas de recomendación, con especial atención a la recomendación contextual y su relación con el aprendizaje profundo. Se mencionarán también las distintas aplicaciones de restaurantes actuales, haciendo hincapié en Yelp. Además, se explicará cómo se lleva a cabo la evaluación de modelos de recomendación para comprobar su eficacia y rendimiento. Asimismo, se presentarán algunas librerías relevantes en el proyecto, en concreto, se abordará el framework de Django, profundizando en su *Object Relational Mappings (ORM)*.

2.1. Sistemas de recomendación

Los sistemas de recomendación son un conjunto de herramientas cuyo propósito es ayudar al usuario a tomar decisiones sobre qué debe consumir, y para ello le muestra una serie de sugerencias personalizadas para él [2]. El uso de estos sistemas ha crecido notablemente en los últimos años debido a la gran información que hay en la red sobre los usuarios y también al gran incremento de la competencia en muchos sectores comerciales.

El funcionamiento de los sistemas de recomendación ha evolucionado debido al Aprendizaje Automático (*Machine Learning*) y, por ello, a día de hoy, existen diferentes tipos, que dependen de variables principales, las cuales condicionarán su funcionamiento y que podemos clasificar según el enfoque utilizado para generar las recomendaciones [3]. Podemos distinguir entre sistemas basados en popularidad, que aconsejan aquellos “ítems” que son más populares, en base a una variable principal que suele ser el número de ventas, o número de visitas, sistemas basados en contenido, que recomiendan “ítems” basándose en el historial del usuario, y sistemas basados en filtrado colaborativo, que analizan el comportamiento de todos los usuarios con el objetivo de encontrar un patrón común entre ellos para realizar recomendaciones.

También existen otras técnicas de recomendación, como los sistemas híbridos, que combinan diferentes técnicas para generar las recomendaciones [2], o los sistemas de recomendación basados en Aprendizaje Profundo (*Deep Learning*), los cuales utilizan redes neuronales para establecer relaciones

entre producto y usuario [4]. Esta clase de algoritmos es en la que nos centraremos en este trabajo, por lo que hablaremos en más detalle de ellos posteriormente.

2.1.1. Recomendación contextual y aprendizaje profundo

En este trabajo, nos hemos centrado en las técnicas relacionadas con la recomendación contextual, las cuales utilizan como variable principal un contexto. Este contexto puede ser la hora del día, el día de la semana, la popularidad del restaurante, entre otros, y se busca encontrar patrones que permitan obtener recomendaciones más precisas y personalizadas para el usuario cuando se encuentre en un contexto determinado. En concreto, estas técnicas representan un cambio en la manera de generar sugerencias, puesto que deja amoldarlas de forma más eficaz a las necesidades y preferencias individuales de cada usuario.

También se han utilizado técnicas de *Deep Learning (DL)*, y en concreto el uso de redes neuronales, ya que de esta manera se puede analizar gran cantidad de datos y encontrar patrones, en vez de basarse en simples correlaciones. Además, esta técnica permite una constante actualización del modelo de recomendación, ya que se adapta a las modificaciones de los datos y de las preferencias del usuario, y permite un mayor análisis detallado de los datos del usuario y del contexto. Algunas técnicas recientes que han demostrado buenos resultados son *Neural Collaborative Filtering (NeuCF)* y *Factorization Machines (FM)*:

NeuCF: Es un tipo de modelo de aprendizaje profundo que se puede utilizar para predecir la puntuación de un elemento por parte de un usuario. El modelo combina técnicas de redes neuronales con el filtrado colaborativo para mejorar la precisión de las recomendaciones personalizadas y aprendiendo las relaciones entre las características de los elementos y los usuarios, así como las relaciones entre los usuarios y otros usuarios [5].

FM: Estas técnicas son utilizadas para el enfoque del filtrado colaborativo en los sistemas de recomendación, y en el caso de la recomendación contextual, para descomponer la matriz de preferencias de factores latentes, y calcular la similitud del usuario y elemento, para modelar su relación y obtener mejores recomendaciones. Dentro de esta técnica, también se utiliza una variante de ella, llamada DeepFM, que combina los puntos fuertes de las *FM* y las redes neuronales convolucionales [6].

2.1.2. Aplicaciones de la recomendación de restaurantes y Yelp

Gracias a las tecnologías que existen actualmente, y sobre todo Internet, muchas personas ya recurren a ellas para descubrir nuevos lugares que visitar, pero también nuevos restaurantes donde comer, y no acudir a los sitios locales de siempre. A raíz de esto, han aparecido muchas aplicaciones y plataformas para la recomendación de restaurantes. Una de las más conocidas es Yelp, una plataforma en línea que permite a sus usuarios descubrir y revisar negocios locales, incluyendo restaurantes.

Con Yelp, los clientes puede leer las diferentes reseñas de restaurantes y las puntuaciones sobre ellos de otros usuarios. Obviamente, también pueden dejar sus reseñas y puntuaciones de un restaurante que han visitado. Además, esta plataforma se basa en el historial previo del propio cliente para recomendarle otros restaurantes que podrían gustarle [7]. Este tipo de aplicaciones facilita a los clientes descubrir nuevos sitios para comer y tener más información sobre ellos, para tomar una decisión a la hora de escoger el negocio.

En la actualidad existen otras aplicaciones de recomendación de restaurantes; estas incluyen TripAdvisor [8] que permite a los usuarios ver reseñas y recomendaciones sobre los negocios, y OpenTable, que es una plataforma de gestión de restaurantes que ofrece reservas online, pedidos para llevar, ver menús, herramientas de marketing, recompensas de fidelización de clientes y más [9]. Otra muy popular es Google Maps, la cual permite entre otras muchas funcionalidades el encontrar restaurantes cercanos a tu ubicación, y que posee una gran base de datos. Sin embargo, una limitación común de este tipo de aplicaciones es que, al ser empresas privadas, es difícil obtener datos completos y actualizados, ni saber cómo funcionan estos algoritmos por dentro.

Se puede evaluar el éxito de estos modelos utilizando una variedad de enfoques y métricas, lo que les permite tomar decisiones a la hora de la implementación. En la siguiente sección describimos los métodos más habituales para ello.

2.1.3. Evaluación de modelos de recomendación

La evaluación de los modelos de recomendación es un componente crucial para determinar su eficacia y precisión en el proceso de realizar recomendaciones. Nuestros modelos utilizan diferentes métricas, de las cuales vamos a destacar algunas de ellas; para una definición exhaustiva, aconsejamos revisar [10].

Precision (P): La precisión es una métrica que se emplea frecuentemente en evaluaciones de modelos de recomendación. Esta métrica evalúa la exactitud de las recomendaciones del modelo en comparación con las preferencias reales del usuario.

Recall (R): La exhaustividad es otra métrica que evalúa cómo de bien el sistema puede proporcionar recomendaciones relevantes para la totalidad del conjunto de datos.

Area Under the ROC Curve (AUC): Otra métrica que han utilizado nuestros modelos es *AUC*. Esta métrica se basa en la curva de ROC que calcula nuestra biblioteca, y en base a ello nos da un resultado. Si el valor es 1 o cercano a él, nuestro modelo sería válido, en cambio, si es 0 o próximo a él, nuestro modelo no sería válido. Este uso de la curva de ROC y el *AUC* es conocido como método ROC-AUC.

Mean Average Precision (MAP): Como su nombre indica, es una medida de la precisión media de una lista clasificada de elementos.

Normalized Discounted Cumulative Gain (NDCG): Es una medida de la calidad de una lista clasificada de elementos, teniendo en cuenta la relevancia de los elementos y sus posiciones en la lista.

F1-score: Es una media armónica del recall y precision.

Root Mean Square Deviation (RMSE): Mide la diferencia entre las puntuaciones pronosticadas y las reales.

Mean Absolute Error (MAE): Calcula la diferencia absoluta media entre las puntuaciones previstas y las reales.

Mean Reciprocal Rank (MRR): Evalua cualquier proceso que produzca una lista de posibles respuestas a una muestra de consultas, ordenadas por probabilidad de acierto.

Recall en la posición n (R@n): Mide la fracción de todos los elementos relevantes que se recuperaron en los primeros n elementos del ranking.

Precisión en la posición n (P@n): Mide la fracción de elementos relevantes entre los primeros n elemento previstos.

2.1.4. Librerías

En este trabajo se han usado varias librerías según el campo de aplicación. Empezamos describiendo las relacionadas con aprendizaje automático, como Scikit-learn, TensorFlow, PyTorch, Keras, entre otras, que se pueden utilizar para desarrollar sistemas de recomendación. La elección de estas librerías dependerá del problema al que se quiera enfrentar, del conjunto de datos y de las capacidades que se disponga, por ejemplo, se sabe que TensorFlow y PyTorch son eficientes para tratar conjuntos de datos grandes, mientras que la arquitectura de Keras es más concisa, legible y fácil de comprender [11].

DeepCarsKit

DeepCarsKit es una biblioteca de código abierto para recomendaciones profundas conscientes del contexto, construida en Python, PyTorch y RecBole [1]. La biblioteca incluye una serie de algoritmos y modelos de aprendizaje profundo que pueden utilizarse para diferentes problemas de recomendación, como la recomendación de películas, de noticias, artículos en una tienda a través de la web, o en contenido personalizado en redes sociales. Al ejecutar la biblioteca se puede obtener información sobre qué algoritmos tienen el mejor rendimiento y seleccionar el más adecuado para su caso de uso.

Una de las características a destacar es su arquitectura modular y escalable: se puede combinar y ajustar los diferentes componentes según las necesidades, lo cual es muy útil si tan solo se necesita una serie de componentes para una tarea específica. Además, se pueden configurar los parámetros para hacer que la biblioteca sea compatible con diferentes tipos de hardware y evitar problemas de ren-

dimiento. Por ejemplo, pueden ajustar la cantidad de memoria que se utiliza durante el entrenamiento o el número de GPUs que se utilizan.

Para entrenar y evaluar los modelos de aprendizaje automático, la biblioteca proporciona una serie de herramientas para la recopilación y etiquetado de datos. Por defecto, vienen con un conjunto de campos que se utilizan para los datos de entrada, pero se pueden modificar en un archivo llamado config.yaml. Este archivo también permite modificar ciertos valores para el modelo y el conjunto de datos. Destacar que DeepCarsKit no tiene una base de datos integrada, en su lugar utiliza un archivo .inter, donde almacena los diferentes campos de los datos a usar. El usuario debe crear este archivo antes de poder utilizar la biblioteca. En su favor, viene con dos conjuntos de datos de ejemplo para que el usuario pueda entender cómo funciona y utilizarlos como referencia para crear su propio conjunto de datos.

Existen otras librerías de código abierto para la investigación de recomendaciones, como Surprise o LightFM, pero no en el área de recomendaciones conscientes del contexto usando aprendizaje profundo, y es por ello que la hemos elegido.

Además, es bastante reciente (su última actualización es de Febrero 2023) y se puede ayudar a mejorarla, por lo que estará en constante evolución y mejora. Pensamos que puede llegar a ser una opción bastante popular para aquellos que busquen una solución de recomendación con conceptos contextuales.

TensorFlow

TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, desarrollada por el equipo de Google Brain y lanzada en 2015 como software libre y de código abierto [12]. Ofrece una gran variedad de herramientas para entrenar y crear modelos de *Deep Learning (DL)*, como redes neuronales simples, o incluso modelos complejos de varias capas.

Una de sus principales ventajas es su escalabilidad, ya que está diseñada para manejar grandes conjuntos de datos y puede ser utilizada en todo tipo de hardware, desde portátiles hasta grandes clusters de servidores. Añadir que ofrece una gran cantidad de funciones y operaciones que permiten la propia creación de modelos de *DL* totalmente personalizados.

Es compatible con populares lenguajes de programación, como Java, C++, Python, lo que hace que sea más fácil integrar la biblioteca en aplicaciones existentes.

PyTorch

PyTorch es una biblioteca de aprendizaje automático de código abierto basada en la biblioteca de Torch, utilizado para aplicaciones como visión artificial y procesamiento de lenguajes naturales [13].

También es usada para desarrollar sistemas de recomendación y otros tipos de aplicaciones de

aprendizaje automático. Una de sus características más destacada es que posee una arquitectura dinámica de grafos computacionales, lo que significa que se puede definir y modificar el grafo de cómputo en tiempo de ejecución, lo que permite una mayor flexibilidad en la construcción y entrenamiento de los modelos.

PyTorch dispone de herramientas para visualizar los datos y modelos, lo que permite a los programadores un fácil seguimiento de los modelos y datos a medida que se van entrenando y así validar su rendimiento.

2.2. Django

En esta sección se discute la elección del framework web Django para la implementación del modelo y su despliegue en una API REST. Se exponen las ventajas de Django frente a otros frameworks web como Flask, y se explica en detalle el *Object Relational Mappings (ORM)* de Django.

2.2.1. Django vs Flask

En este proyecto, hemos decidido usar Django, un framework web de Python, utilizado para desarrollar aplicaciones web escalables y seguras, para la implementación del modelo y su despliegue en API REST. Aunque es una biblioteca más pesada que otras, como FastApi o Flask, ofrece más herramientas para futuras mejoras y cambios.

También tomamos ventaja al estar escrito en Python, ya que este lenguaje ofrece una amplia variedad de librerías que facilitan el desarrollo de modelos y algoritmos de recomendación.

En cuanto a la interfaz gráfica, Django cuenta con numerosas herramientas para el desarrollo de páginas web, lo que nos permite crear una interfaz atractiva y sencilla para que los usuarios se sientan cómodos al usarlo. Para ello se usan plantillas, escritas en HTML y CSS, para darle una estructura y un diseño claro, que contienen etiquetas que permiten la inserción dinámica de contenido. Estas plantillas son procesadas por Django en tiempo de ejecución y se utilizan para generar páginas dinámicas. Además, Django cuenta con un sistema de herencia de plantillas que permite definir una estructura inicial para todas las páginas de la aplicación, y a partir de ellas extenderlas particularmente. Esto permite que haya una interfaz uniforme y coherente en toda la aplicación.

Django tiene constante soporte y actualizaciones, ya que es usado por una amplia parte de la comunidad de desarrolladores. Según Github Stars, Django fue el framework web más popular en 2022 con 65.687 estrellas, por delante de Flask y FastApi, y actualmente tiene 71.428 estrellas [14]. Django garantiza una seguridad y estabilidad de la aplicación en un futuro. También conviene destacar que tiene una arquitectura que es una variante del patrón de diseño MVC, en la cual la plantilla actúa como la capa “vista”, y se encarga de definir la presentación de los datos en la interfaz de usuario, y

esta permite una fácil separación de la base de datos, de la lógica y presentación [15].

En cuanto a la elección entre Django y Flask, debemos considerar que Flask es un framework minimalista y flexible, por lo que sería mejor si nuestro proyecto fuera pequeño y simple, pero en nuestro caso, ya que la idea es evolucionarlo a una mayor escala, Django es mejor debido a que ofrece una estructura más sólida y completa. Otra ventaja es que Django cuenta con un sistema de *ORM*, que permite que sea más fácil y rápido desarrollar aplicaciones con bases de datos grandes y complejas, por no olvidar que cuenta con una gran comunidad activa y con una gran cantidad de documentación para su desarrollo. Un *ORM* es una técnica de programación que permite mapear objetos de un sistema orientado a objetos a tablas de una base de datos relacional, y utilizarla como motor de persistencia [16].

2.2.2. ORM Django

El *ORM* de Django es una de sus características más destacadas, ya que permite interactuar con la base de datos relacional utilizando objetos Python, sin tener que escribir SQL.

Se abstrae de las operaciones en la base de datos, ya que en su lugar, se definen clases de objetos Python, que heredan de la clase 'django.db.models.Model', por lo que el programador no tiene que crear tablas o manipular datos a nivel SQL. Además, el *ORM* de Django es compatible con varias bases de datos, como PostgreSQL, SQLite, Oracle y MySQL, lo que lo hace muy flexible.

Otra característica es que permite realizar migraciones de forma simple. Las migraciones son cambios en la estructura de la base de datos que se realizan a través de código, y no en la base de datos manualmente. También permite crearlas y aplicarlas de manera segura y controlada, lo que facilita el mantenimiento y la evolución de la base de datos.

Los modelos de Django son muy flexibles, lo que permite que la base de datos defina diferentes tipos de campos, como números enteros, cadenas, fechas, valores booleanos y otros. Además, el *ORM* de Django permite que se puedan establecer relaciones entre los modelos, es decir, relaciones entre las tablas de datos. También permiten establecer restricciones sobre los datos que se pueden almacenar en la base de datos así como definir reglas de validación personalizadas para los campos, por lo que es una manera de realizar las consultas de manera eficiente. Además permite filtrar, ordenar y agrupar los datos de forma sencilla si se quiere realizar alguna consulta más compleja. En concreto, destacar la función Model.objects.all.filter(), y Model.objects.all.get().

3

Diseño, Análisis e Implementación

En este capítulo se abordará el diseño, análisis e implementación del proyecto. Se describirá el diseño del software central, su estructura general y el ciclo de vida elegido. También se identificarán los requisitos funcionales y no funcionales del sistema. Se analizará el preprocesamiento y la recopilación de datos, se presentarán los modelos de recomendación y se explicará su implementación. Por último, se hablará de la aplicación web, su arquitectura y funcionalidades implementadas.

3.1. Diseño

En la siguiente sección se presentará el diseño de la aplicación, donde se describirá su estructura y los módulos que la componen. También se explicará el ciclo de vida seleccionado para su desarrollo, detallando su implementación.

3.1.1. Estructura

El sistema se encuentra dividido en tres módulos principales, cada uno de ellos realiza una función específica en el proyecto. En la figura 3.1 podemos observar la estructura general de esta división, resaltando la conexión entre los módulos.

Preprocesamiento de datos

Este módulo se encarga de realizar todas las tareas relacionadas con la extracción, filtrado y limpieza de los datos obtenidos a partir del dataset de Yelp. Es esencial para garantizar la certeza de los datos utilizados en el proyecto. Aquí se encuentran todos los archivos necesarios para llevar a cabo estas operaciones.

Modelos de recomendación

Este segundo módulo se centra en la implementación de los algoritmos de recomendación propuestos por la biblioteca DeepCARSKit, así como su entrenamiento y validación de cada uno de ellos.

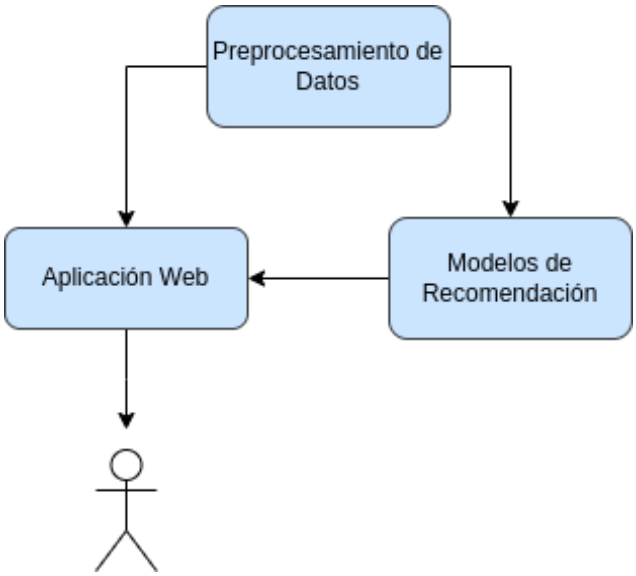

Figura 3.1: Esctructura del proyecto.

Utiliza los datos ya preprocesados para construir y entrenar los modelos de recomendación (*FM* y *NeuCF*, ya explicados en la sección 2.1.1). Además se encuentran todos los archivos necesarios para su correcta configuración, como el archivo 'config.yaml', donde se pueden variar los parámetros de ajuste para los modelos.

Aplicación Web

En este último módulo, se encuentran los resultados de las recomendaciones para que los usuarios interactúen con ella y poder mostrárselo a través de la interfaz. Además, incluye la implementación de las vistas, modelos y plantillas del framework Django.

3.1.2. Ciclo de vida

Para el desarrollo de la aplicación, se ha optado por seguir un ciclo de vida en cascada, ya que se trata de un sistema simple, donde los requisitos han estado bien definidos desde el primer momento. Estos requisitos no han sufrido cambios en ninguna etapa del desarrollo, por lo que nos asegura que se ha escogido un ciclo de vida que se ajusta perfectamente a la naturaleza del proyecto. En la figura 3.2 se muestran las diferentes etapas que contiene este tipo de ciclo de vida.

3.2. Ánalisis

En esta sección, se detallarán los requisitos necesarios que el sistema debe cumplir para satisfacer las necesidades de los usuarios. Estos requisitos se han clasificado en dos categorías: requisitos

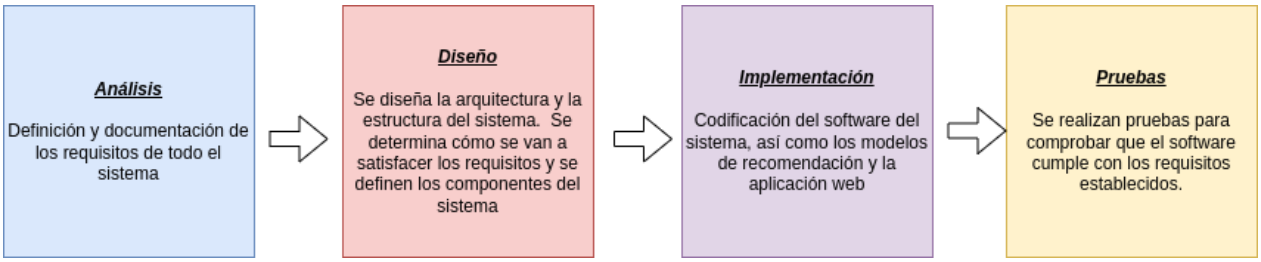

Figura 3.2: Ciclo de vida.

funcionales y requisitos no funcionales, y a su vez, los requisitos funcionales se han dividido según los módulos del sistema.

3.2.1. Requisitos funcionales

Preprocesamiento de Datos

RF-1.– El sistema debe ser capaz de extraer los datos proporcionados por Yelp.

RF-2.– El sistema debe hacer una limpieza de los datos extraídos para eliminar datos eliminados o incompletos.

RF-3.– El sistema debe hacer una selección de los datos que vaya a utilizar en la aplicación, en concreto: usuarios con más de 800 reseñas hechas y negocios que pertenezcan a la ciudad de Tampa.

RF-4.– El sistema debe hacer una transformación de los datos a un formato adecuado para su uso en los modelos de recomendación.

RF-5.– El sistema debe hacer un archivo .inter que contenga los diferentes contextos, que son:

user_id:token, business_id:token, rating:float, day:token, hour:token, categories:token, popular:token, parking:token, contexts:token, uc_id:token.

Modelos de recomendación

RF-6.– Los modelos de recomendación deben implementar algoritmos de recomendación, como *FM* y *NeuCF*, para generar las recomendaciones.

RF-7.– Los modelos de recomendación deben ser entrenados utilizando los datos preprocesados para aprender los patrones y preferencias del usuario.

RF-8.– Los modelos de recomendación deben ser capaces de generar las recomendaciones basándose en contextos, como el día, la hora, si se dispone de parking, entre otros.

Aplicación Web

RF-9.– La aplicación web debe proporcionar una interfaz intuitiva y sencilla de utilizar, con una navegación clara y accesible para el usuario.

RF-10.– La aplicación web debe permitir al usuario realizar recomendaciones seleccionando un negocio y un usuario.

RF-11.– La aplicación web debe mostrar las recomendaciones generadas (incluidos los contextos) por los modelos de recomendación de manera clara y sencilla.

RF-12.– La aplicación web debe mostrar a los usuarios la información detallada de los diferentes usuarios, negocios y reseñas.

RF-13.– La aplicación web debe mostrar a los usuarios una lista con los diferentes usuarios y negocios disponibles para hacer las recomendaciones.

3.2.2. Requisitos no funcionales

RNF-1.– El sistema debe ser eficiente a la hora de manejar los datos, optimizando el uso de los recursos y minimizando el tiempo de procesamiento.

RNF-2.– El sistema debe asegurarse que los datos preprocesados no deben tener errores y garantizar la integridad de los datos.

RNF-3.–Los modelos de recomendación deben ser eficientes, sobre todo en términos de tiempo de respuesta, y a la hora de realizar puntuaciones.

RNF-4.– Los modelos de recomendación deben proporcionar recomendaciones precisas para que sean relevantes y útiles para los usuarios.

RNF-5.– Los modelos de recomendación deben permitir la incorporación y evaluación de diferentes algoritmos de recomendación, además de realizar ajustes en los modelos existentes.

RNF-6.– La aplicación web debe tener una interfaz atractiva visualmente, proporcionando una buena experiencia al usuario

RNF-7.– La aplicación web debe ser rápida en términos de tiempo de carga y respuesta, minimizando la espera del usuario.

3.3. Recopilación y preprocesamiento de datos

Para llevar a cabo la aplicación, se ha utilizado una base de datos facilitada por Yelp, pero al ser tan extensa, con miles de registros y múltiples campos, se ha optado por realizar un proceso de filtrado y preprocesamiento de datos. Sin embargo, es importante tener en cuenta que en este caso, los datos utilizados se limitaron a negocios ubicados en Norteamérica. Los datos más concretos antes y después del filtrado se muestran en el siguiente capítulo (ver figura 4.2).

En primer lugar, se ha descargado la base de datos de Yelp desde su sitio web oficial, la cual constaba de una carpeta con diferentes archivos JSON, en los que cada uno representaban diferentes tablas. Cada uno de estos archivos contenían información y campos correspondientes a los diferentes objetos de la base de datos.

Una vez obtenidos los archivos JSON, había que volcarlos en la base de datos, para ello, se creó un archivo Python, en el que procesaba cada línea del archivo JSON como un objeto individual y lo agregaba a la base de datos. Sin embargo, debido al elevado número de líneas, agregarlos de uno en uno no era eficiente en términos de tiempo, ya que esto significaba muchas peticiones a la base de datos. Para solucionar este problema se optó por agregar los objetos en lotes, creando listas de los objetos, y agregándolos en bloques a la base de datos.

Como se ha mencionado anteriormente, dado que había varios archivos JSON en la carpeta, este proceso tuvo que repetirse por cada uno de ellos. Un problema adicional que surgió fue debido al tamaño de los archivos, ya que cada archivo JSON tuvo que dividirse en archivos más pequeños para evitar que el ordenador se bloqueara al intentar abrir ficheros tan grandes.

Posteriormente, se decidió disminuir nuestra base de datos, ya que al ser tan grande, posiblemente sería algo lento en cuanto a ejecución. Se comprobaron qué ciudades eran las que tenían mayor número de reseñas, y Tampa (Florida), fue la elegida, por número de reseñas de los usuarios. De esta manera, nos asegurábamos tener información de usuarios experimentados en la base de datos, ya que cumplían que debían tener 800 o más reseñas cada uno.

Una vez filtrados los datos, se procedió a crear el archivo .inter, con los campos de acuerdo a las especificaciones del RF-5. Tanto “user_id” como “business_id”, eran tokens que contenían los identificadores del usuario y negocio respectivamente. Por su parte, “rating” representaba la puntuación que el usuario le había dado a ese negocio. Los campos “day” y “hour” tokenizados para facilitar su representación: si el día correspondía a un día entre semana, tomaba el valor “WD”, y si era un día de fin de semana “WE”; respecto a la hora, si era posterior a las 17:00, tomaba el valor de “Din” (en referencia a “Dinner”), y si era anterior, “Lch” (en referencia a “Lunch”).

Se realizó una función de hashing para las categorías, con el fin de reducir el tamaño del archivo. Además, se guardó en un archivo aparte un diccionario que relacionaba el nombre de la categoría con un respectivo número (su correspondiente hash), por si se necesitaba en un futuro.

El campo “parking” y “popular” también fueron tokenizados. Si el negocio disponía de parking propio, se utilizaba el token “Gar” (indicando “garage”), y en caso contrario “NoGar” (indicando “No garage”). En cuanto a la popularidad del negocio, se determinó que un negocio era considerado popular si su número de reseñas se encontraba dentro del top 20 % (debía ser mayor de 418), por lo que utilizaba el token “Pop” si el negocio era popular, y “NoPop” si no lo era.

Por último, los campos “contexts” y “uc_id” fueron tokenizados. El primero consistía en la agrupación de los contextos (día-hora-categorías-popularidad-parking) porque así lo requiere la librería. El segundo campo representaba la unión de todos los campos separados por "_",igualmente, debido a ser un requisito de la librería. Se hicieron pruebas para intentar eliminar algunos de estos campos y así reducir el tamaño del fichero generado, pero la librería dejaba de funcionar.

3.4. Modelos de recomendación

En este apartado hablaremos más en detalle cómo se ha relacionado el módulo de los modelos de recomendación con el resto de la aplicación.

Este módulo funciona como una capa de servicios externos que desempeña un papel importante en la aplicación al proporcionar las recomendaciones contextuales proporcionadas para los usuarios. Además, para integrarlo en la aplicación, se implementaron llamadas a funciones específicas que devuelven puntuaciones de recomendación y ofrecen un resultado al considerar una serie de contextos.

Para obtener los resultados de los modelos de recomendación de la biblioteca DeepCARSKit, se requirió configurar varios archivos. El archivo principal fue config.yaml, en el cual se debía especificar la ruta al archivo .inter que contenía los datos de entrenamiento en la variable "dataset". Además, se ajustaron diferentes parámetros según el algoritmo utilizado. En el caso de NeuCMFii, se establecieron los siguientes valores: mf_embedding_size: 64 (tamaño de la incrustación de factorización matricial), mlp_embedding_size: 64 (tamaño de la incrustación del perceptrón multicapa), mlp_hidden_size: [128, 64, 32] (tamaño oculto del perceptrón multicapa), learning_rate: 0.01 (tasa de aprendizaje) y dropout_prob: 0.1 (probabilidad de eliminación). También existían parámetros generales que se podían ajustar según las necesidades de los datos, tales como: epochs: 50 (épocas), train_batch_size: 500 (tamaño del lote de entrenamiento) y eval_batch_size: 4960 (tamaño del lote de evaluación).

Una vez configurado este archivo, simplemente se ejecutaba la biblioteca utilizando el comando "python run.py". Este archivo se encargaba de realizar los entrenamientos y las validaciones necesarias. Los resultados de rendimiento se almacenaban en la carpeta "logs", donde se podían consultar algunas de las métricas mencionadas anteriormente en la sección 2.1.3 Además, se guardaba un conjunto de datos y el modelo para utilizarlo posteriormente en un archivo junto con la biblioteca RecBole (que es en la que se basa DeepCARSKit. De cara a la integración con la aplicación web, al ingresar una serie de usuarios y negocios, RecBole proporcionaba las puntuaciones de las recomendaciones al

variar los contextos. Basándonos en estas puntuaciones, en la aplicación web se muestran recomendaciones de los contextos específicos ordenadas en base a las selecciones del usuario y del negocio realizadas.

3.5. Aplicación Web

Para esta aplicación se ha utilizado Django como framework de desarrollo y una base de datos, adaptada del dataset de Yelp. Django sigue un patrón de diseño llamado *Model View Template (MVT)*, similar al popular patrón *Model View Controller (MVT)*, con la diferencia de que la parte del controlador es atendida por el propio Django.

3.5.1. Arquitectura

La arquitectura de nuestra aplicación web sigue el patrón *MVT*, se puede apreciar en la figura 3.3 y consta de las siguientes partes:

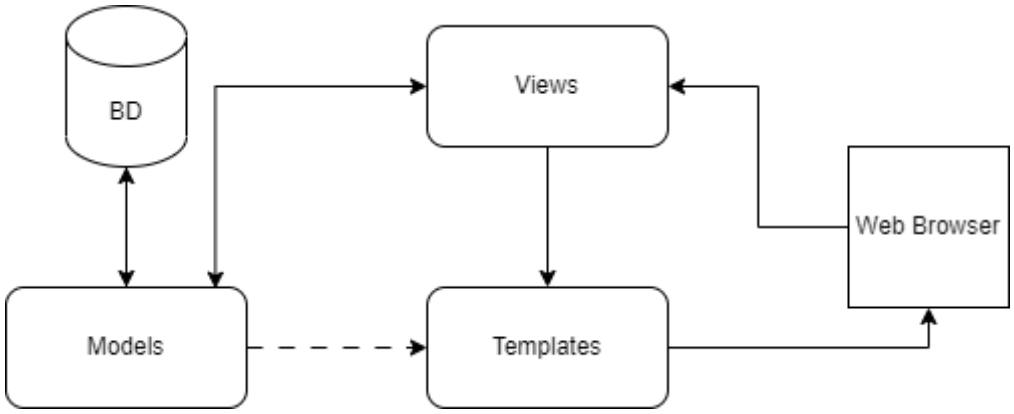

Figura 3.3: Diagrama Patrón MVT [17].

→ La M hace referencia a “Model”, que es la capa de acceso a la base de datos. En esta capa está contenida toda la información sobre los datos, es decir, las relaciones que hay en ellos, cómo se acceden, cómo validarlos y cómo se relacionan.

→ La T hace referencia a “Template”, que es la capa de presentación. Esta contiene todo lo relacionado con las decisiones que se toman sobre la presentación, como puede ser qué es lo que se va a mostrar en la página web.

→ La V hace referencia a “View”, que es la capa de la lógica de negocio. En esta capa está contenida todo sobre la lógica que accede al propio modelo, y la delega a la plantilla apropiada. Establece una conexión entre las otras dos capas, entre los modelos y las plantillas.

El funcionamiento de este patrón sería el siguiente, tomando como referencia la figura 3.4: Primero,

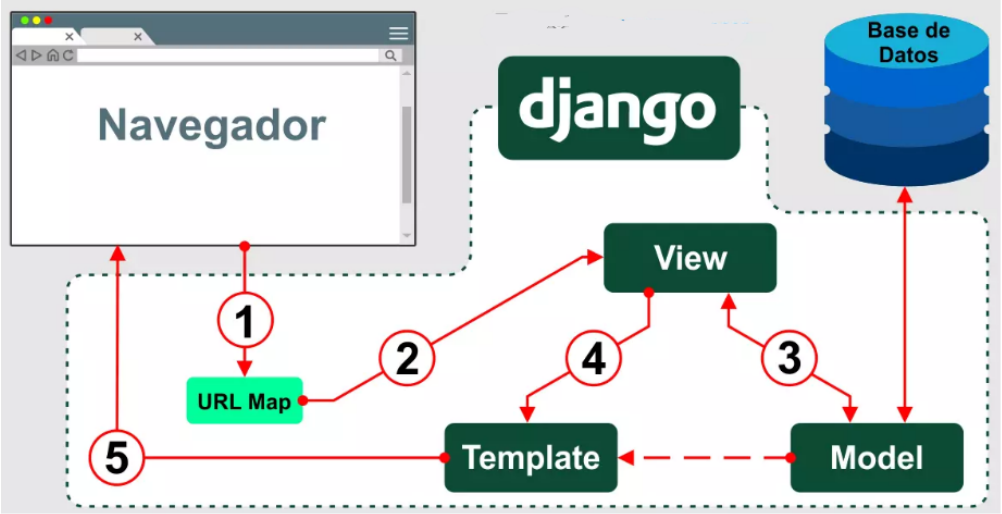

Figura 3.4: Diagrama Funcionamiento Django [18].

el usuario escribiría una dirección en el navegador web o bien haría click en un enlace (1). De esta manera, se accede al mapa de las URLs, en el cual contiene una serie de rutas que están asociadas a una determinada view (2). En caso de que la vista necesite algún dato de algún modelo, se solicitará a la capa "Model", y esta consultará a la base de datos y los devolverá. Una vez que se tengan los datos, la capa "View" los enviará a la capa "Template" (4), que es la que tiene la lógica para la correcta presentación de los datos. Por último, esta capa "Template", será la que envíe al navegador la página con la representación correspondiente, para que este lo muestre al usuario que realizó la solicitud.

Base de datos

Para la base de datos de la aplicación, como se ha mencionado en la sección 3.3 se ha contado con los conjuntos de datos que proporciona Yelp, en el siguiente enlace: `https://www.yelp.com/dataset`. Se pueden diferenciar diferentes clases, las cuales están definidas cada una en diferentes archivos json. Estas clases son:

- Business: Es la que contiene toda la información respecto al negocio, principalmente restaurantes. Los campos más importantes de esta clase son: Business_id, el identificador del negocio; name, el nombre del negocio; city y state, que es la ciudad y estado en el que se encuentra el negocio respectivamente; Stars: la puntuación de estrellas que tiene el negocio; Review_count: el número de reseñas que se le han hecho al negocio; categories, attributes y hours, que son las distintas categorias atributos y horarios de aperturas que tiene el negocio.
- Review: Esta clase contiene toda la información respecto a una reseña que realiza un usuario sobre un negocio. Los campos más importantes son: review_id, identificador único de una reseña; user_id: identificador único del usuario que realiza la reseña; business_id: identificador único del negocio al que se le hace la reseña; text: el texto de la reseña, en el cual el cliente da su opinión sobre el negocio; stars: la puntuación que le da el usuario al negocio.
- User: Esta clase contiene toda la información respecto a los usuarios. Sus campos más importantes son: user_id: identificador único de un usuario; name: nombre del usuario; review_count: número de reseñas que ha realizado el usuario; average_stars: media de las puntuaciones de las reseñas que ha realizado el usuario.

También, hay otras clases como Checkin, Tip y Photo, que vienen en el dataset, pero se ha optado por descartarlas ya que no proporcionaban información relevante para la aplicación, y ocupaban un espacio innecesario. Finalmente, tendríamos un diagrama de clases como el de la figura 3.5.

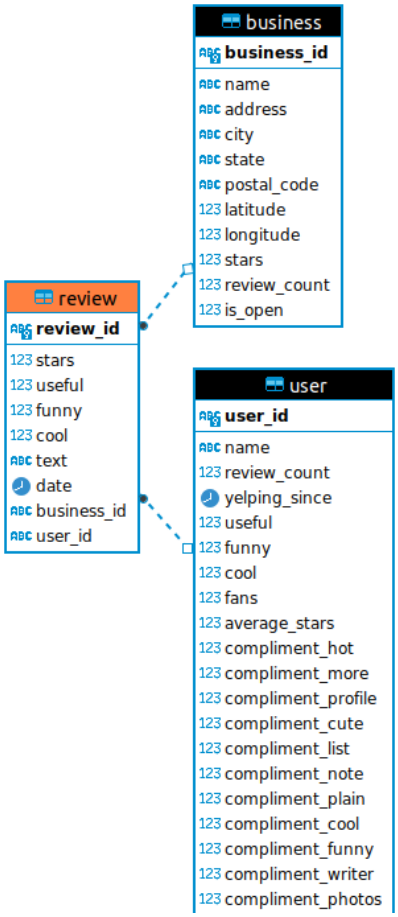

Figura 3.5: Diagrama de clases.

3.5.2. Funcionalidades

En este apartado, hablaremos sobre el diseño de la interfaz de nuestra aplicación web y de las distintas funcionalidades que existen para que el usuario tenga una experiencia intuitiva e interactiva. En la figura 3.6 podemos ver un diagrama de secuencia sobre la funcionalidad de la aplicación web.

Al iniciar la aplicación, los usuarios se encuentran en la página de inicio (*home*), donde el usuario se encuentra con dos menús desplegables. Estos menús permiten seleccionar un usuario y un negocio. Una vez que se han seleccionado ambos, pueden hacer click en el botón “Recomiéndame un restaurante”, para ser redirigidos a nueva página con la recomendación correspondiente. Se puede ver esta página en la figura 3.7.

En esta nueva página, “resultado”, se verá el resultado de la recomendación, en el cual se puede comprobar si dispone de parking, las diferentes categorías a las que pertenece y lo más importante, la

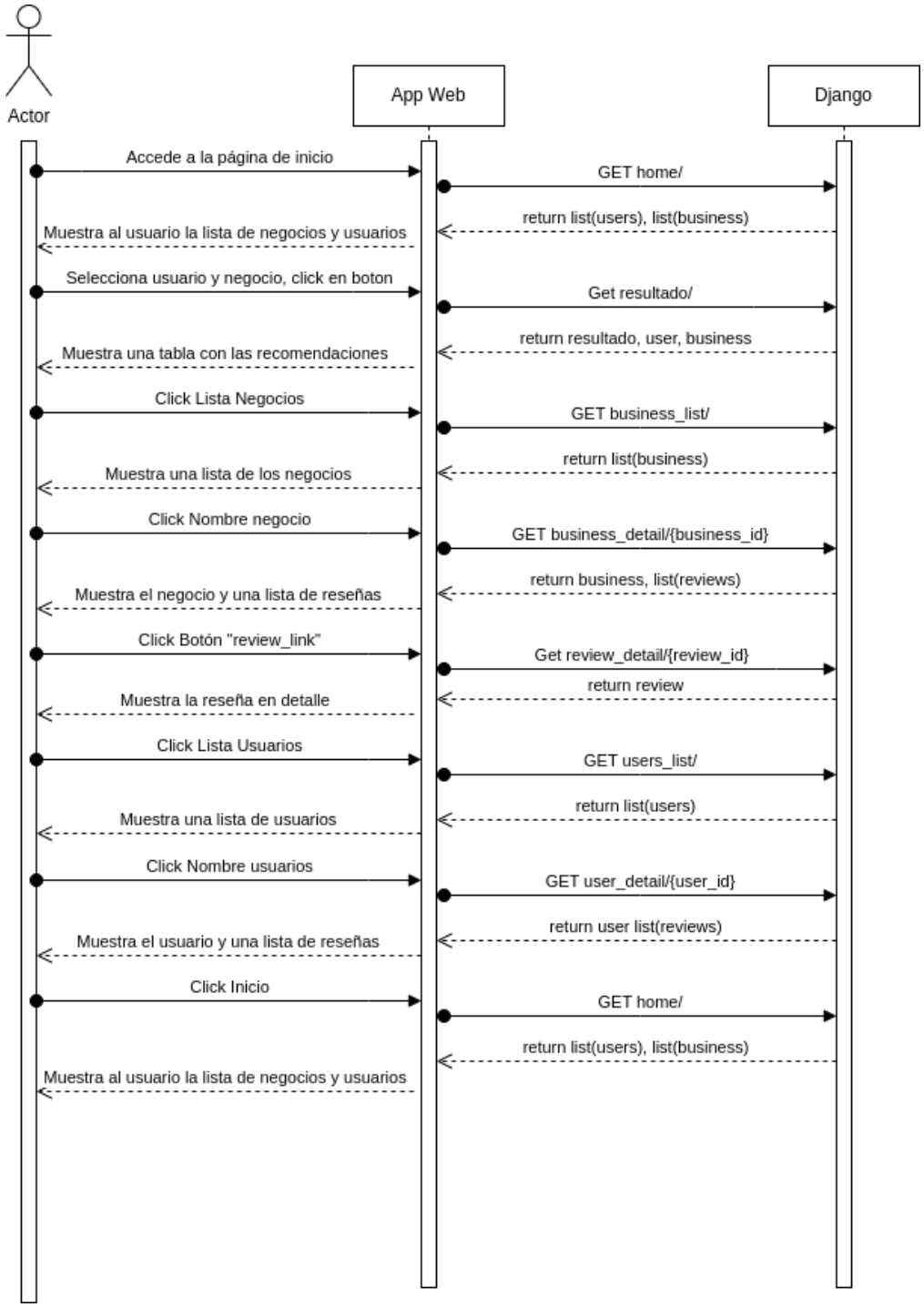

Figura 3.6: Diagrama de secuencia que describe la funcionalidad de la aplicación web.

hora y el día recomendados para acudir a él, ya sea para una cena o una comida, entre semana o durante el fin de semana. Además, se ofrecen otras recomendaciones según las puntuaciones generadas por la biblioteca DeepCARSKit.

En la aplicación, los usuarios también tienen acceso a un menú superior que les permiten navegar a diferentes páginas. Estas páginas incluyen la página de inicio, la lista de negocios y la lista de usuarios.

En la página de la lista de negocios, mostramos los diferentes negocios que el usuario puede seleccionar desde el inicio, junto con información relevante, como la dirección, el código postal, las estrellas que tiene y el número de reseñas. Si se desea obtener información más detallada sobre un negocio en particular, bastará con hacer click en él, y serán redirigidos a la página 'business_detail', que describiremos más adelante.

La página de la lista de usuarios se muestran los diferentes usuarios que se pueden seleccionar desde el inicio para hacer la recomendación. Junto a cada usuario se muestra el número de seguidores y el promedio de estrellas que tiene a la hora de realizar reseñas. Al igual que en la lista de negocios, si se hace click sobre el nombre del usuario, serán redirigidos a la página "user_detail", donde encontrarán información más detallada sobre usuario en particular.

En la página "business_detail", se muestran todos los detalles del negocio, como el número de reseñas, las estrellas, su dirección, y las coordenadas de latitud y longitud. Además, se incluye una tabla "Reseñas", que contiene todas las reseñas que han realizado los usuarios sobre ese negocio en concreto. La tabla tiene diferentes columnas como un enlace a la reseña, el nombre del usuario, la puntuación, la fecha y el texto de la reseña. Si se desea más información de la reseña, tan solo debe hacerse click, en el botón "review_link" de la fila correspondiente a la reseña, que le reenviará a una nueva página llamada "review_detail". También pueden hacer click en el nombre del usuario para acceder a la página "user_detail" y obtener más información sobre el usuario que realizó la reseña.

En la página "user_detail", se muestran todos los detalles del usuario, incluyendo la información previamente mencionada que se encuentra en la tabla de la lista de los usuarios. Además, se incluye una tabla que muestra los diferentes elogios que el usuario ha realizado a los negocios, como elogios buenos. tiernos, sencillos, entre otros. Al igual que en la página "business_detail", también se muestra una tabla "Reseñas", que muestra las distintas reseñas que ha realizado el usuario sobre distintos negocios.

Por último, en la página "review_detail", se muestra de manera detallada y ordenada la información de una reseña específica. Proporciona una visualización más amplia de los detalles de la reseña en comparación con la tabla de reseñas que se encuentra en las páginas de detalle del negocio y del usuario. Al igual que en las páginas anteriores, si los usuarios hacen click en el nombre del usuario o del negocio, serán redirigidos a la página en detalle respectivamente.

Figura 3.7: Home de la aplicación Web.

4

Pruebas y resultados

En este capítulo se abordarán las pruebas y los resultados obtenidos durante el desarrollo de este proyecto. Se proporcionará una descripción detallada del entorno de pruebas utilizado, se analizarán las estadísticas de los datos utilizados en los experimentos. Finalmente, se presentarán los experimentos que se han realizado.

4.1. Entorno de pruebas

A la hora de realizar las pruebas en el proyecto, al igual que en el desarrollo, se ha optado por un entorno local, donde todas las características están detalladas en la tabla 4.1. Mencionar que inicialmente se optó por un sistema operativo Windows, pero a la hora de trabajar con Django y otras librerías y programas, se encontró mayor comodidad y familiaridad al utilizar Ubuntu, por lo que se decidió cambiar al sistema operativo Linux.

Se creó un entorno virtual para el uso de diferentes librerías y versiones, lo que permitió aislar el proyecto y sus dependencias del entorno global del S.O. Surgieron algunas incompatibilidades, por lo que hubo que utilizar versiones específicas de las librerías que inicialmente nos proporcionaba el "requirements.txt" de la biblioteca de DeepCarsKit. Se puede ver en detalle este archivo en el Apéndice A.

Es importante destacar que se utilizó un entorno basado en Python para llevar a cabo todas las pruebas y el desarrollo del proyecto. Esto nos permitió aprovechar las ventajas y características del lenguaje de programación Python, así como utilizar numerosas bibliotecas y herramientas disponibles.

Después de considerar todos los aspectos mencionados y realizar diversas mediciones de tiempo, se llegó a la conclusión de que el equipo descrito en la tabla 4.1 sería suficiente para realizar nuestro proyecto. Aunque este equipo presenta ciertas limitaciones, ya que al ser un ordenador de uso personal que no podía estar encendido de forma continua, y no disponer de gran memoria RAM ni procesador para realizar grandes operaciones simultáneamente, se consideró que podría cumplir con los requisitos mínimos necesarios para realizar las pruebas y obtener buenos resultados.

Componente	Características
CPU	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
GPU	Intel Corporation UHD Graphics 620
RAM	16 GB
S.O.	Ubuntu 22.04.2 LTS
Python Enviroment	Python 3.8.17

Tabla 4.1: Tabla características técnicas equipo de pruebas.

4.2. Estadísticas de los datos

En este apartado hemos analizado los diferentes datos que hemos preprocesado anteriormente. En primer lugar, comprobamos el número de negocios que hay por cada ciudad, y después, seleccionamos una ciudad para la cual vamos a realizar el resto de análisis. En este proceso, filtramos el número de reseñas que tienen los usuarios en relación con esa ciudad, buscando un umbral que indique que esos usuarios son usuarios experimentados, y por lo tanto, las reseñas son más confiables.

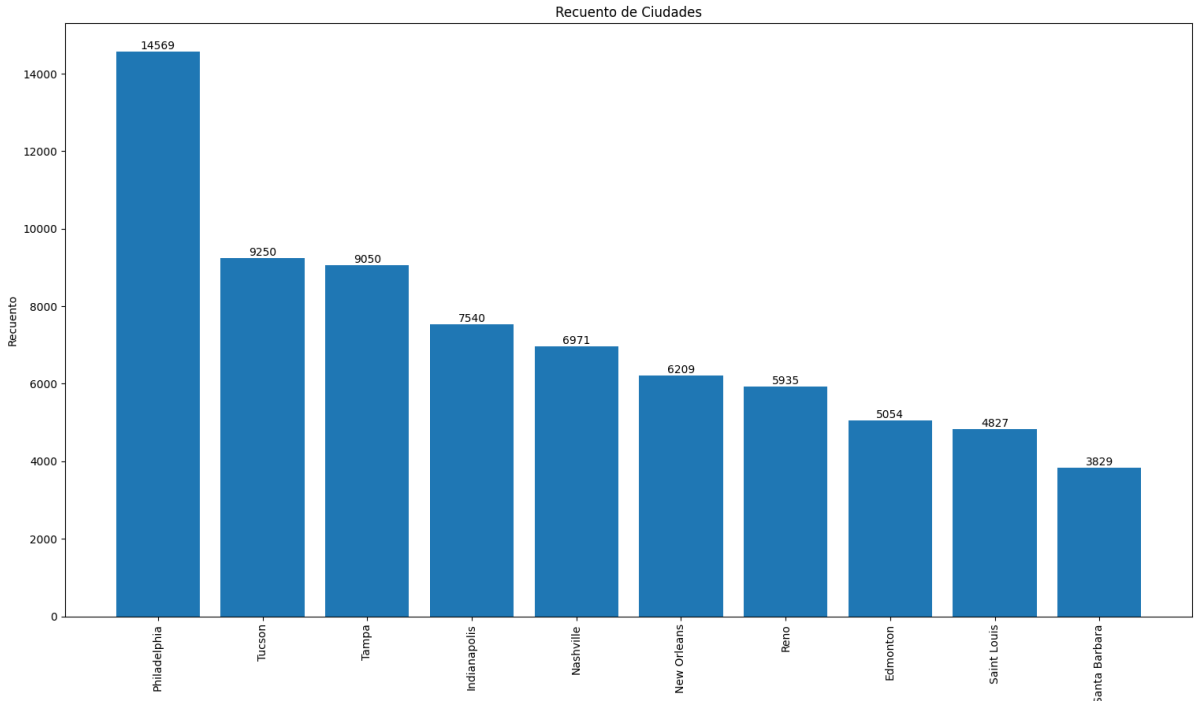

Figura 4.1: Estadísticas de las ciudades del conjunto de datos.

Al observar la figura 4.1, vemos que hay una primera ciudad que tiene un número considerablemente mayor de negocios en comparación con los demás. Sin embargo, en nuestro proyecto decidimos optar por la tercera ciudad debido a consideraciones relacionadas con el tamaño de los archivos y la configuración de nuestros modelos a la hora de entrenarlos. Al reducir este tamaño, podemos hacer mayor uso de diferentes contextos y comparar las diferencias entre ellos.

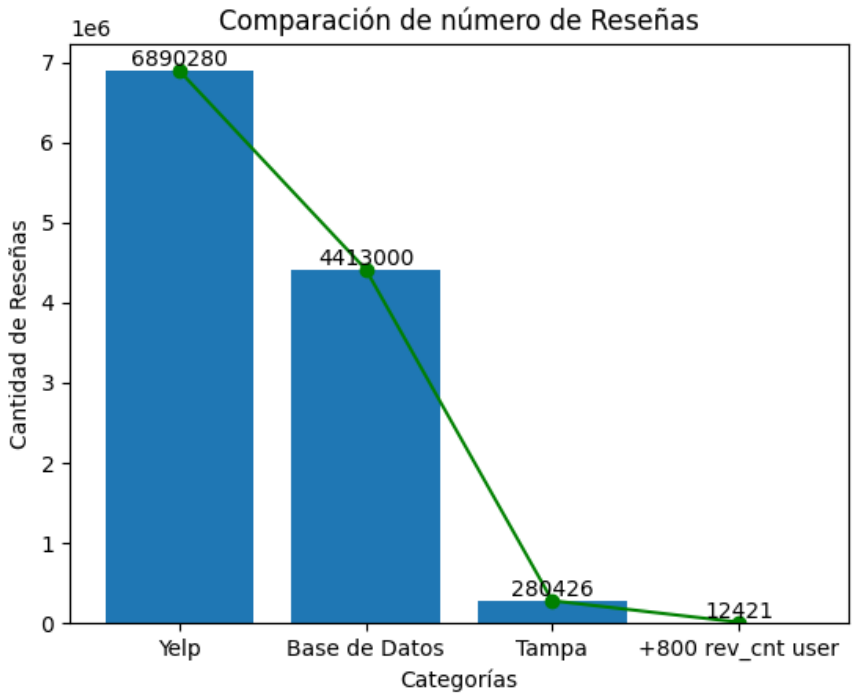

Figura 4.2: Comparación Número de Reseñas.

Ahora analizamos la Figura 4.2, donde se muestra una gráfica con el número de reseñas realizadas por los usuarios. En este caso, establecemos el umbral de más de 800 reseñas, lo que nos permitió reducir la cantidad de datos en comparación con los datos iniciales de manera significativa. Vemos que inicialmente Yelp nos proporciona 6,8 millones de reseñas, luego en nuestra base datos guardamos 4,4 millones, donde observamos una bajada en el número de datos debido a la existencia de datos nulos o inconsistentes, que no añadimos a la base de datos. Después, filtramos con reseñas cuyos negocios se encuentran en la ciudad de Tampa, y hay 280k de reseñas, y finalmente tenemos 12.421 reseñas cuyo negocio está en Tampa y donde el usuario que ha realizado la reseña tiene más de 800 reseñas realizadas.

Finalmente, para determinar si un negocio es popular, su número de reseñas debe ser superior al del 80 % frente a otros negocios. La figura 4.3 muestra un histograma donde se puede ver la situación de los negocios con respecto al número de reseñas que tienen en los datos, y dónde se encuentra el umbral seleccionado. La mayoría de los negocios tienen un número de reseñas entre 0 y 50, lo que indica que la mayoría de estos negocios tienen un “review_count” bajo. El número de negocios disminuye lentamente a medida que aumenta el “review_count”, lo que indica que hay menos negocios con un alto número de reseñas. También se aprecia una línea horizontal marcada con el valor 418, que representa el límite para el top 20 % de los negocios con mayor “review_count” en Tampa. Esta línea nos permite identificar qué negocios se encuentran dentro de este grupo “popular”. Por último, podemos ver que hay un pequeño grupo de negocios con “review_count” significativamente alto, superando

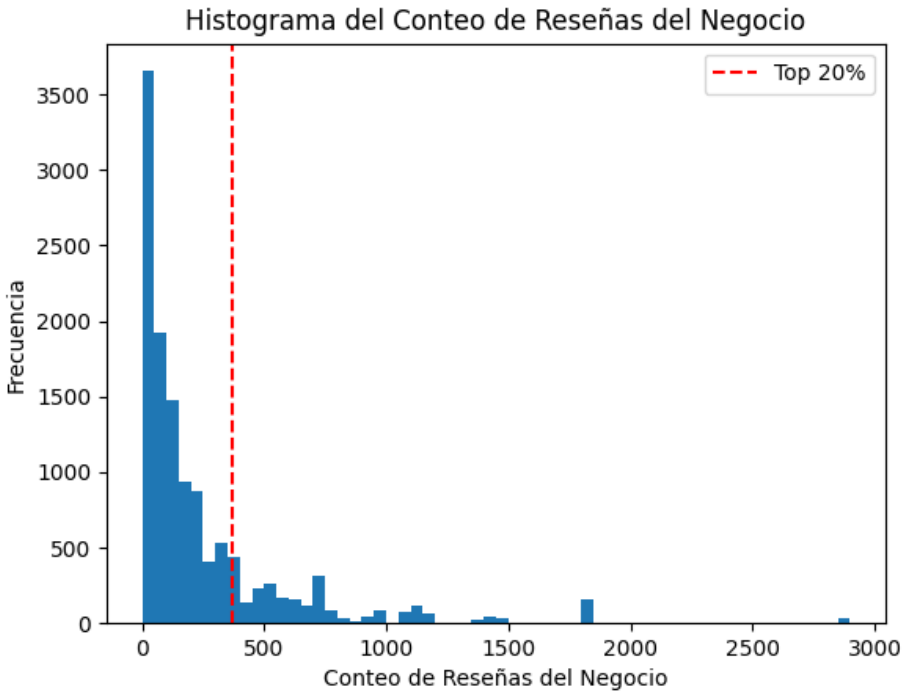

Figura 4.3: Reseñas de Negocios.

el valor de 418, por lo que los negocios que se encuentren en este grupo, se consideran los más populares de la ciudad.

4.3. Experimentos

Los experimentos realizados para evaluar el rendimiento y la efectividad del sistema de recomendación implementado se presentarán en esta sección. Se realizará una comparativa de algoritmos de recomendación, centrándonos específicamente en el análisis de *NeuCF*. Se analizarán los resultados obtenidos y se compararán los resultados considerando diferentes contextos.

4.3.1. Comparativa de Algoritmos de Recomendación: Análisis de NeuCF frente a otros enfoques con TripAdvisor

Al analizar los resultados obtenidos por los diferentes algoritmos que disponemos de la biblioteca de DeepCarsKit, podemos observar diferencias en términos de rendimiento. Estos algoritmos son: DeepFM, FM, NeuCMF0i, NeuCMF0w, NeuCMFi0, NeuCMFii, NeuCMFw0 y NeuCMFww. Mencionar que estos resultados han sido obtenidos utilizando el dataset de TripAdvisor, facilitado por la propia biblioteca DeepCARSKit, para saber qué algoritmo priorizar en nuestros experimentos con el dataset

de Yelp.

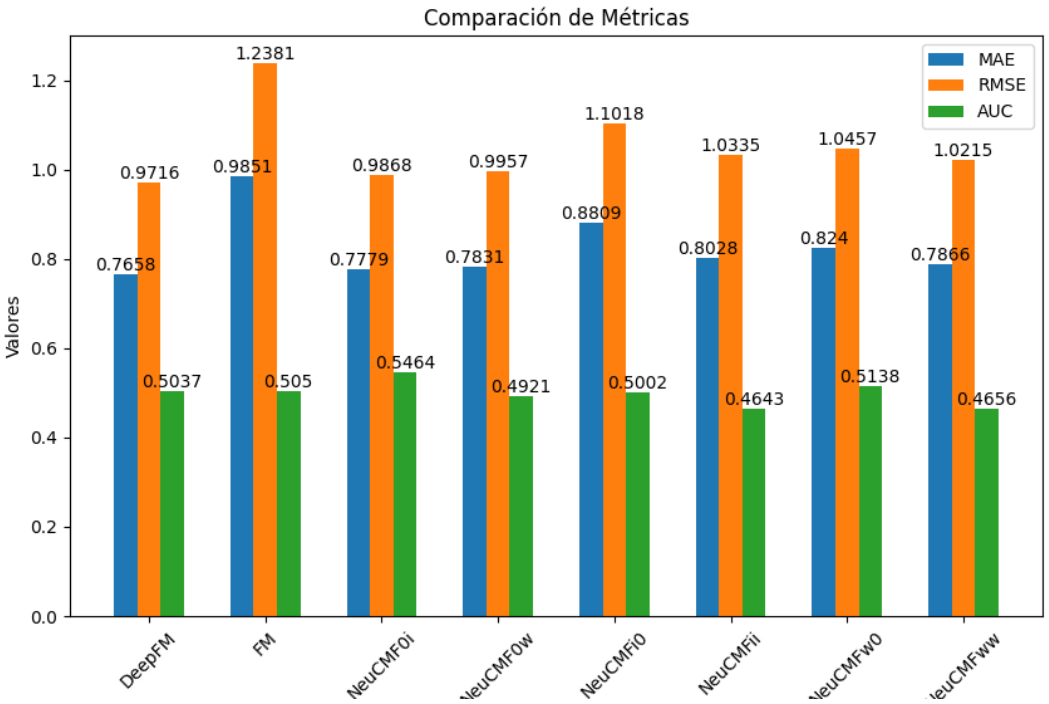

Figura 4.4: Comparativa de algoritmos de recomendación en TripAdvisor, incluyendo métricas de error (MAE, RMSE, donde es mejor los valores menores) y de precisión (AUC, valores altos son mejores).

Los resultados de la figura 4.4 muestran que el algoritmo DeepFM tuvo el mejor desempeño con un valor de 0,7658 en términos de la *MAE*. Esto demuestra que, en comparación con los valores reales de las reseñas, las predicciones generadas por este algoritmo tienen una discrepancia promedio menor. Sin embargo, los algoritmos NeuCMF0w y NeuCMF0i obtuvieron *MAE* de 0,9851 y 0,7866, lo que indica un rendimiento ligeramente más bajo.

El algoritmo DeepFM volvió a tener un mejor desempeño con un valor de 0,9716 en *RMSE* lo que indica una menor dispersión de las predicciones en relación con los valores reales. Además, los algoritmos NeuCMF0i y NeuCMF0w tuvieron un desempeño razonablemente bueno con *RMSE* de 0.9868 y 0.9957.

Se utilizó *AUC* para evaluar la capacidad de clasificación. Esta vez, el algoritmo NeuCMF0i logró el valor de *AUC* más alto con 0.5464. Esto indica una mayor capacidad de discriminación cuando se trata de clasificar las reseñas. Sin embargo, debe tenerse en cuenta que otros algoritmos también lograron valores de *AUC* superiores a 0,5, lo que indica una cierta capacidad predictiva; aunque no hay que olvidar que, en rigor, valores menores a 0,5 indican predicciones aleatorias.

También hay que tener en cuenta factores adicionales a los resultados de rendimiento de las diferentes métricas, como el tiempo de ejecución de los algoritmos. Estos tuvieron valores bastante

similares, entre 21 y 23 segundos, por lo que no es un factor de gran relevancia. El único valor que es diferente al resto es el del algoritmo FM, que tardó 17,22 segundos. NeuCMF0i fue el que requirió más tiempo, con un tiempo de ejecución de 23.96 segundos. Dado que las ejecuciones presentadas en este análisis son solo una muestra, se recomienda realizar múltiples ejecuciones y evaluar la desviación para obtener resultados más confiables.

En resumen, los resultados muestran que tanto el algoritmo NeuCMF0i como el algoritmo DeepFM tuvieron un tiempo de ejecución razonablemente bueno y mejoraron en *MAE* y *RMSE*. Sin embargo, NeuCMF0i superó a DeepFM en *AUC*. Consideramos que NeuCMF0i es el algoritmo óptimo porque ofrece un rendimiento similar en *RMSE* y *MAE* en comparación con DeepFM, pero significativamente mejor en *AUC*. Estos resultados, sin embargo, plantean una discrepancia con los publicados por la biblioteca DeepCARSKit, ya que el autor indica que NeuCMFii es el algoritmo más efectivo, y también tiene un buen rendimiento. Según las pruebas anteriores y las métricas evaluadas, llegamos a la conclusión de que NeuCMF0i es la mejor opción para este caso, lo cual puede tener sentido ya que la naturaleza de los datos y el tamaño de los mismos es muy diferente al utilizado por el autor de la librería.

4.3.2. Comparativa de Algoritmos de Recomendación: Análisis de NeuCF frente a otros enfoques con Yelp

En este apartado vamos a volver a analizar como en la sección 4.3.1 los diferentes algoritmos que proporciona la biblioteca DeepCARSKit, pero esta vez con el dataset de Yelp. En la figura 4.5 podemos observar qué resultados hemos obtenido con esta nueva configuración.

Por lo general, el algoritmo “NeuCMF0i” vuelve a mostrar un mejor rendimiento destacado en las métricas evaluadas (*MAE*, *RMSE* y *AUC*), lo que indica que podría ser una buena opción para realizar recomendaciones precisas en el conjunto de datos de Yelp. Este algoritmo ha obtenido el valor más alto para *AUC* (0.3241), y los valores más bajos para *RMSE* (1.1011) y *MAE* (0.8502).

Tanto en esta sección como en la anterior 4.3.1, “NeuCMF0i” ha obtenido los mejores resultados para las diferentes métricas de rendimiento, por lo que será nuestro algoritmo a usar en nuestro proyecto.

4.3.3. Análisis de los mejores resultados

En este análisis, se utilizaron una variedad de métricas para evaluar el rendimiento del sistema de recomendación de nuestro proyecto. La *P*, el *R*, F1, *NDCG*, *MRR* y *MAP* son las métricas utilizadas, las cuales brindan una visión exhaustiva de la calidad y relevancia de las recomendaciones que el sistema ha hecho.

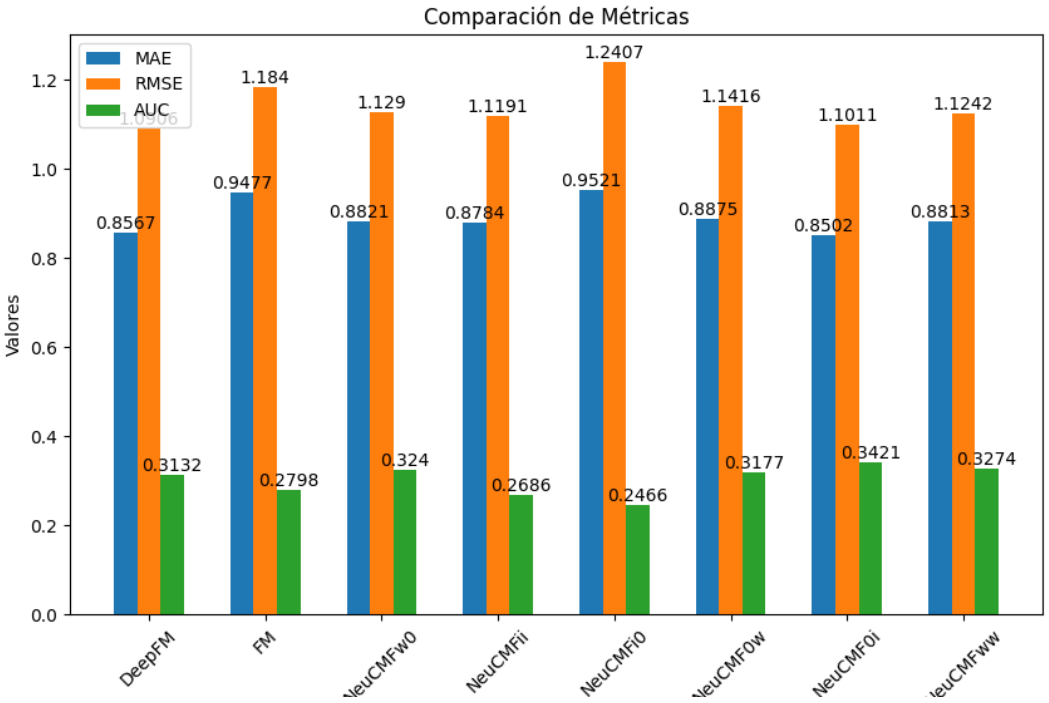

Figura 4.5: Comparativa de algoritmos de recomendación en Yelp, incluyendo métricas de error (MAE, RMSE, donde es mejor los valores menores) y de precisión (AUC, valores altos son mejores).

Según lo descrito en las secciones 4.3.1 y 4.3.2, inicialmente se consideró que el algoritmo más adecuado para utilizar sería “NeuCMF0i” debido a su desempeño en las métricas mencionadas (*MAE*, *RMSE* y *AUC*). Sin embargo, al analizar los resultados de la figura 4.6, se observó que este algoritmo no mostraba buenos resultados en las métricas utilizadas en la evaluación, esto se debe a que estas nuevas métricas no estaban relacionadas con las métricas previamente seleccionadas como óptimas.

En consecuencia, tras considerar los nuevos resultados, se ha decidido utilizar el algoritmo “NeuCMFii” como la mejor opción. Este algoritmo demostró tener mejores resultados en las métricas utilizadas, lo que indica un mejor rendimiento.

Es importante destacar que la elección del algoritmo óptimo puede variar según las métricas utilizadas y las necesidades específicas del sistema de recomendación, por lo que en este caso, se ha tomado la decisión de utilizar “NeuCMFii”.

En la figura 4.7, podemos observar una gráfica de histograma para visualizar los valores de diferentes métricas de evaluación, tanto para métricas@10 como para métricas@20. Los valores promedio para las métricas de *P* (precision@10 y precision@20) son 0.0176 y 0.0111, respectivamente. Estos valores muestran las recomendaciones relevantes dentro de los primeros 10 y 20 elementos recomendados. Los valores son relativamente bajos en este caso, lo que indica que las recomendaciones no son muy precisas, pero hay que tener en cuenta que es un problema muy difícil, por lo que acertar no es trivial.

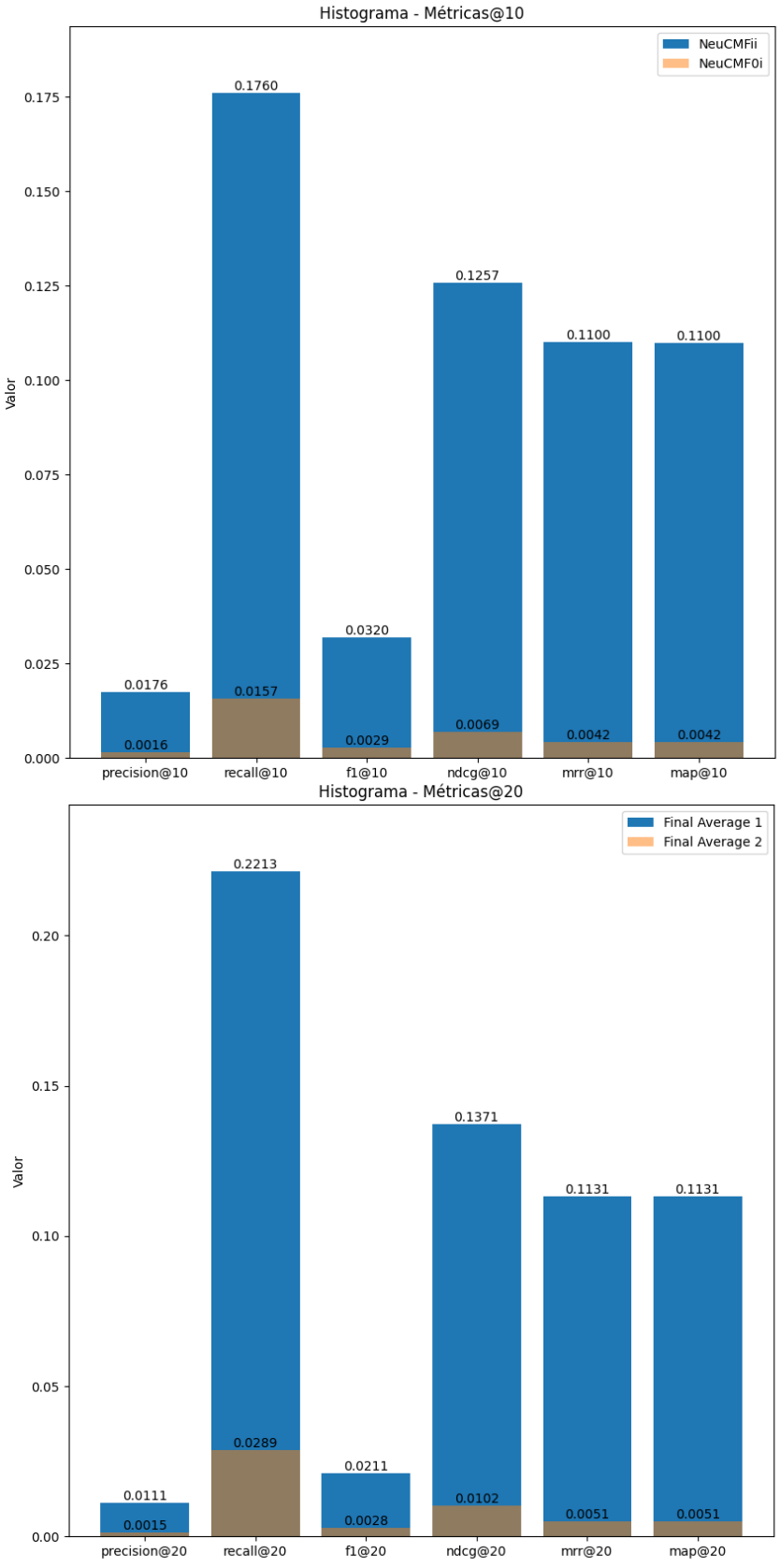

Figura 4.6: Comparativa AUC, RMSE y MAE de los algoritmos NeuCMFii y NeuCMF0i, para dos tamaños del ranking.

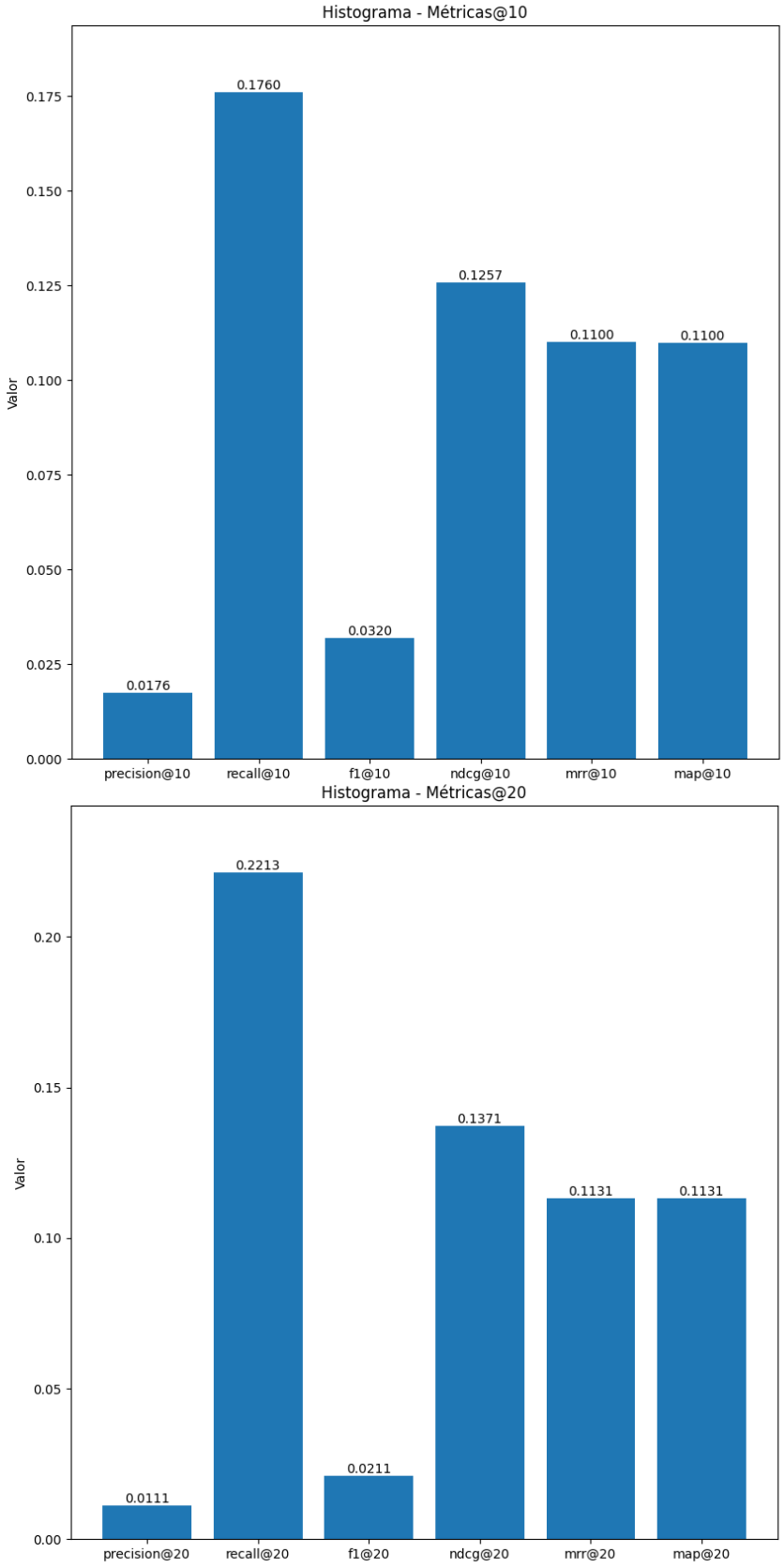

Figura 4.7: Comparativa rendimiento algoritmo NeuCMFii.

Las métricas de *R* (recall@10 y recall@20) tienen valores promedio de 0.176 y 0.221, respectivamente. Estos valores muestran la proporción de elementos relevantes que se recuperan dentro de los primeros 10 y 20 elementos recomendados. A pesar de los valores obtenidos, el sistema de recomendación ha logrado recuperar un porcentaje moderado de elementos relevantes.

Para las métricas F1 (f1@10 y f1@20), los valores promedio son 0.032 y 0.0211, respectivamente. El valor F1 proporciona una visión general del rendimiento del sistema al combinar la precisión y el recall en una sola medida. Los valores obtenidos son relativamente bajos, lo que sugiere que no hay un correcto equilibrio entre la precisión y el recall en las recomendaciones.

Las métricas *NDCG* (ndcg@10 y ndcg@20), tienen valores promedio de 0.1257 y 0.1371, respectivamente. La calidad y relevancia de las recomendaciones clasificadas son evaluadas por el NDCG. Los valores obtenidos muestran que las recomendaciones tienen un nivel de calidad moderado.

En cuanto a las métricas *MRR* (mrr@10 y mrr@20), los valores promedio son 0.110 y 0.1131, respectivamente. La posición promedio del primer elemento relevante en la lista de recomendaciones se muestra por *MRR*. El primer elemento relevante se encuentra en una posición moderada, según los valores obtenidos.

Los valores promedio para las métricas *MAP* (map@10 y map@20) son 0.11 y 0.1131, respectivamente. La precisión promedio de las recomendaciones relevantes en el rango especificado se mide mediante MAP (Mean Average Precision). Los resultados muestran una precisión moderada en las recomendaciones relevantes.

En general, los valores promedio de las métricas indican que el sistema de recomendación funciona de manera moderada. Sin embargo, para proporcionar recomendaciones más precisas y relevantes, puede ser necesario mejorar la precisión, el recall y el equilibrio entre ellos.

4.3.4. Comparativa de Resultados con Diferentes Contextos

En este apartado, analizaremos los resultados obtenidos para diferentes conjuntos de datos. En las figuras 4.8, 4.9, 4.10 y 4.11 se observan variaciones significativas en las métricas de *R* (rc) y *P* (pr). Entre los diferentes conjuntos evaluados, 'result_noPop' muestra un rendimiento notablemente superior, con una precisión@10 de 0.0270 y una precisión@20 de 0.0207, así como un recall@10 de 0.2686 y un recall@20 de 0.4117. Estos valores indican que este conjunto de resultados proporciona recomendaciones más precisas y abarca un mayor número de elementos relevantes en comparación con el resto. Por otro lado, 'result_noCat_noGar' muestra el rendimiento más bajo en términos de precisión y recall, con valores de 0.0085 para precision@10, 0.0084 para precision@20, 0.0520 para recall@10 y 0.0977 para recall@20.

Los resultados indican que la exclusión de algunos contextos, como las categorías y el parking,

afecta negativamente el funcionamiento del sistema de recomendación. Parece que estos contextos son esenciales para hacer recomendaciones precisas y relevantes. Por otro lado, se observa que el contexto “popular” afecta negativamente a las recomendaciones, ya que su exclusión conduce a una mejora en el rendimiento del sistema. Estos resultados resaltan la importancia de considerar cuidadosamente los contextos relevantes y eliminar aquellos que puedan causar ruido o dañar la calidad de las recomendaciones.

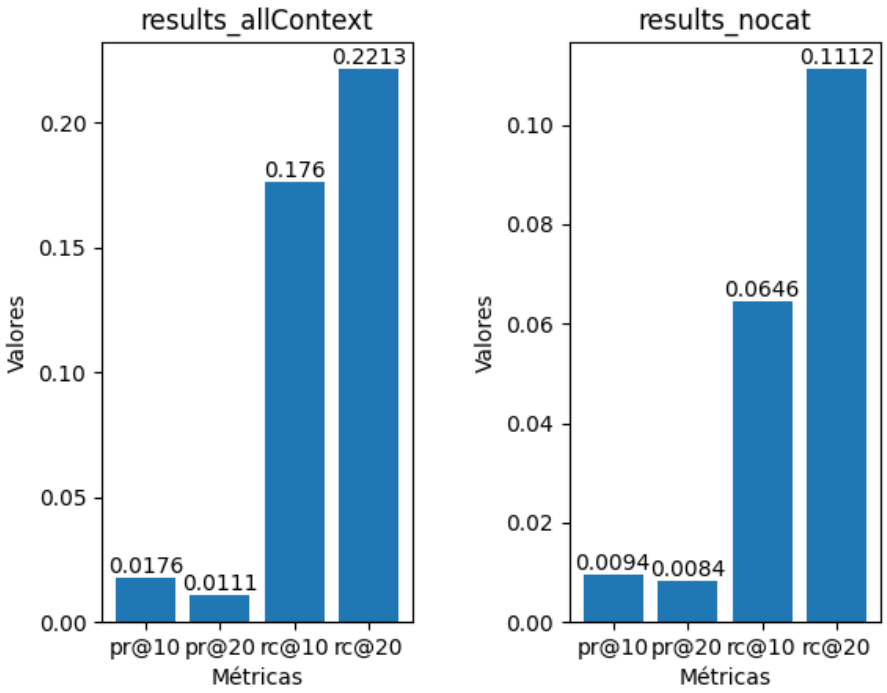

Figura 4.8: Todos los contextos vs No categorías.

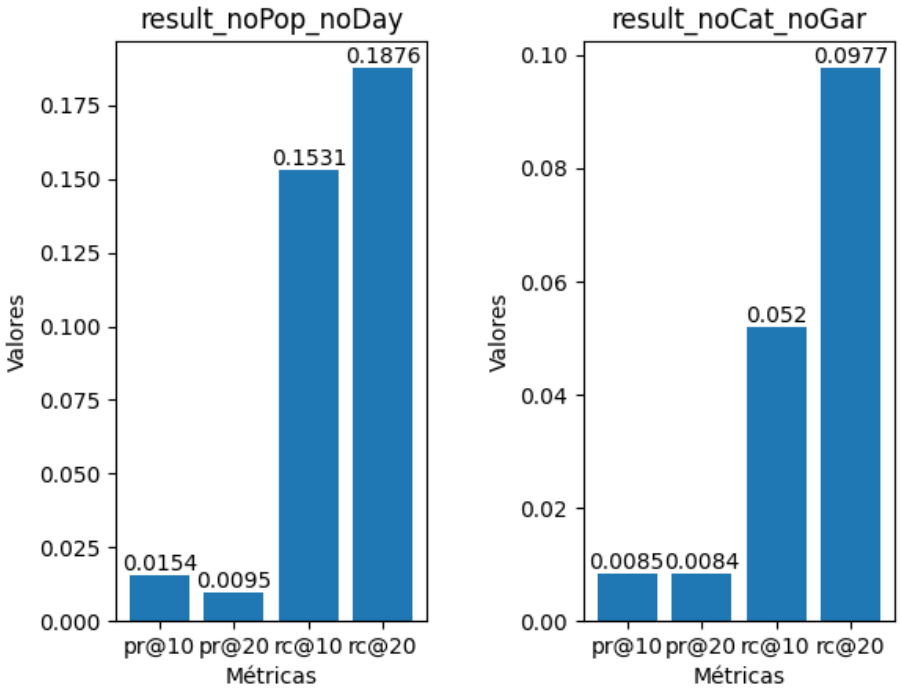

Figura 4.9: Sin 2 contextos.

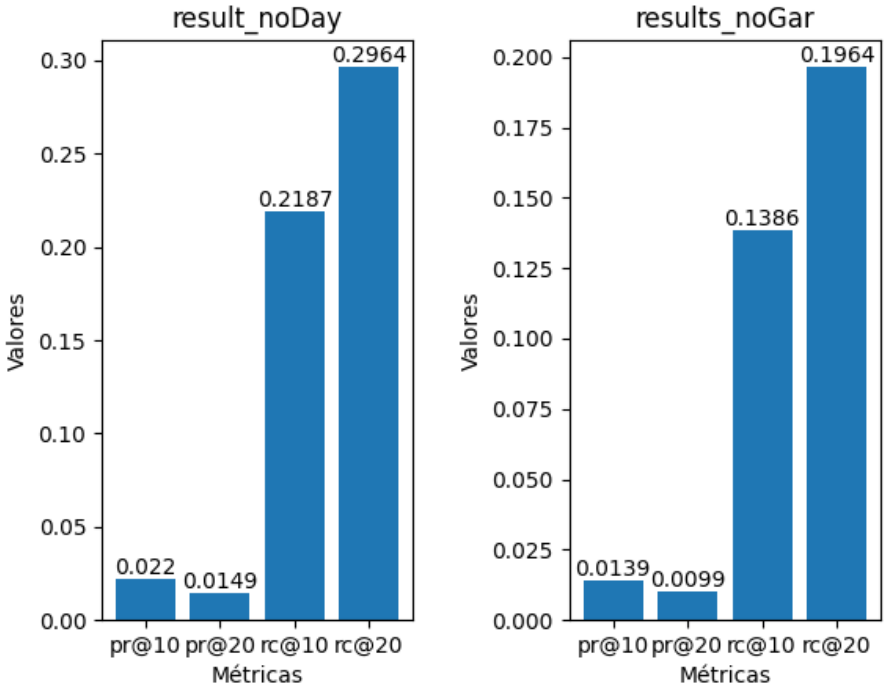

Figura 4.10: Sin Día y Sin Parking.

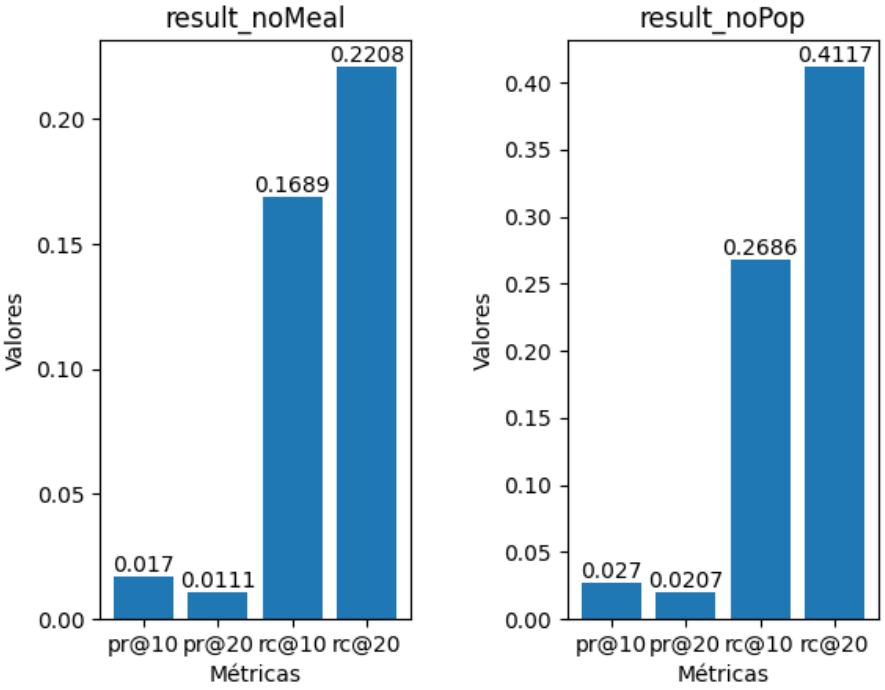

Figura 4.11: Sin Hora y Popular.

5

Conclusiones y trabajo futuro

En esta sección, se presentarán las conclusiones alcanzadas durante la realización de este proyecto, así como las perspectivas de un trabajo futuro para su ampliación.

5.1. Conclusiones

Los sistemas de recomendación se están volviendo cada vez más importantes en nuestra vida diaria. Aprovechamos esta tecnología en cada interacción con servicios o aplicaciones en línea para seleccionar los productos y servicios que se nos ofrecen. Este proyecto se ha centrado en estudiar cómo las técnicas de aprendizaje profundo se pueden aplicar a los sistemas de recomendación contextual, y específicamente en los de restaurantes.

Este proyecto ha permitido explorar y evaluar diferentes algoritmos de aprendizaje profundo utilizando la biblioteca DeepCARSKit. Se ha podido determinar el rendimiento de estos algoritmos en una variedad de tipos de datos y contextos a través de este análisis. Se ha observado que algunos contextos son relevantes y útiles para los sistemas de recomendación, mientras que otros no aportan información importante y pueden causar ruido. Estos resultados destacan la importancia de considerar el contexto en el diseño e implementación de los sistemas de recomendación, así como la selección adecuada del algoritmo utilizado.

Además, destacar la necesidad de llevar a cabo pruebas y experimentos continuos para evaluar el rendimiento de los algoritmos en diversos escenarios y mejorar su precisión. Estas continuas prácticas de evaluación y experimentación son esenciales para mejorar el desempeño de los sistemas de recomendación y garantizar que las recomendaciones generadas sean relevantes y útiles para los usuarios.

Durante el desarrollo de la aplicación web de este proyecto, se aprovechó lo visto en el Grado a la hora de implementar un servicio REST utilizando Django, y se profundizó en temas relacionados con el rendimiento de la propia aplicación y su gestión. Para asegurarse de que los usuarios de la aplicación tengan una experiencia fluida y eficiente, la optimización del rendimiento se convirtió en un componente fundamental.

5.2. Trabajo Futuro

Durante el desarrollo de este proyecto, se han identificado varias áreas y posibles mejoras que podrían realizarse en futuras expansiones de este TFG. Sin embargo, no fue posible explorarlas en profundidad debido a las limitaciones de recursos.

Una de las mejoras destacadas consiste en aumentar el conjunto de datos a la hora de entrenar y validar nuestros modelos de recomendación, ya que esto permitiría un análisis más preciso de los resultados. Además, se podrían realizar más experimentos para realizar un estudio completo y detallado, en concreto, con más algoritmos o probando con contextos diferentes (algo que también depende de los datos que se tengan disponibles).

Otra posible mejora para la aplicación web implementada en Django sería migrarla a la nube, lo que permitiría ampliar su alcance y tener un mayor número de usuarios. Al trasladarla a la nube, se facilitaría el acceso desde cualquier dispositivo y lugar, lo que aumentaría la comodidad y la flexibilidad para los usuarios. Además, se podría implementar un sistema de registro y autenticación de usuarios que permitiría a los usuarios crear sus perfiles, iniciar sesión y escribir sus propias reseñas sobre los negocios. Esto mejoraría la base de datos y proporcionaría retroalimentación adicional para que las recomendaciones fueran más precisas. Surgirían nuevas oportunidades para brindar a los usuarios una experiencia personalizada y enriquecedora que se abrirían como resultado de esta expansión del sistema a la nube y la incorporación de características de interacción de usuarios.

Bibliografía

[1] Y. Zheng, "DeepCARSKit: A deep learning based context-aware recommendation library," *Software Impacts*, vol. 13, p. 100292, Aug. 2022.

[2] F. Ricci, L. Rokach, and B. Shapira, "Recommender systems: Techniques, applications, and challenges," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 1–35, Springer US, 2022.

[3] "Sistemas de recomendación | qué son, tipos y ejemplos." https://www.grapheverywhere.com/sistemas-de-recomendacion-que-son-tipos-y-ejemplos/. Visitado última vez en Junio 2023.

[4] Y. Koren, S. Rendle, and R. M. Bell, "Advances in collaborative filtering," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 91–142, Springer US, 2022.

[5] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, (Republic and Canton of Geneva, CHE), p. 173–182, International World Wide Web Conferences Steering Committee, 2017.

[6] S. Rendle, "Factorization machines," in *2010 IEEE International Conference on Data Mining*, pp. 995–1000, 2010.

[7] V. Martín, "¿qué es yelp y en qué se diferencia de otras plataformas?." https://victormartinp.com/que-es-yelp-y-en-que-se-diferencia-de-otras-plataformas 2013. Visitado última vez en Junio 2023.

[8] M. Academy, "Tripadvisor para restaurantes." https://www.monouso.es/academy/tripadvisor-para-restaurantes/. Visitado última vez en Junio 2023.

[9] GetApp, "Opentable." https://www.getapp.com.mx/software/2044651/opentable-for-restaurants. Visitado última vez en Junio 2023.

[10] A. Gunawardana, G. Shani, and S. Yogev, "Evaluating recommender systems," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 547–601, Springer US, 2022.

[11] J. Terra, "Keras vs tensorflow vs pytorch: Key differences among deep learning." https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article, Junio 2023. Visitado última vez en Junio 2023.

[12] Wikipedia, "Tensorflow." https://es.wikipedia.org/wiki/TensorFlow, Octubre 2021. Visitado última vez en Junio 2023.

[13] Wikipedia, "Pytorch." https://es.wikipedia.org/wiki/PyTorch, Diciembre 2022. Visitado última vez en Junio 2023.

[14] Github, "Web application frameworks." https://github.com/showcases/web-application-frameworks, Marzo 2017. Visitado última vez en Junio 2023.

[15] uniwebsidad, “El patrón de diseño mtv.” https://uniwebsidad.com/libros/django-1-0/capitulo-5/el-patron-de-diseno-mtv. Visitado última vez en Junio 2023.

[16] Wikipedia, “Asignación objeto-relacional.” https://es.wikipedia.org/wiki/Asignaci%C3%B3n_objeto-relacional, Marzo 2022. Visitado última vez en Junio 2023.

[17] Héctor, “Patrón mvt: Modelo-vista-template.” https://docs.hektorprofe.net/django/web-personal/patron-mvt-modelo-vista-template/, Octubre 2018. Visitado última vez en Junio 2023.

[18] “Que es el patrón mvt (model-template-view).” https://espifreelancer.com/mtv-django.html, 2018. Visitado última vez en Junio 2023.

Acrónimos

AUC Area Under the ROC Curve.

DL Deep Learning.

FM Factorization Machines.

KNN K-Nearest Neighbors.

MAE Mean Absolute Error.

MAP Mean Average Precision.

MRR Mean Reciprocal Rank.

MVT Model View Template.

NDCG Normalized Discounted Cumulative Gain.

NeuCF Neural Collaborative Filtering.

ORM Object Relational Mappings.

P Precision.

R Recall.

RMSE Root Mean Square Deviation.

Apéndices

A

Ficheros adicionales

En esta sección se adjunta el archivo “requirements.txt”, que contiene las versiones de las distintas bibliotecas empleadas en el entorno virtual, necesario para la realización del proyecto.

Código A.1: Fichero requirements (parte 1).

```
absl-py==1.4.0
anyio==3.6.2
app==0.0.1
argon2-cffi==21.3.0
argon2-cffi-bindings==21.2.0
arrow==1.2.3
asgiref==3.6.0
asttokens==2.2.1
attrs==22.2.0
backcall==0.2.0
backports.zoneinfo==0.2.1
beautifulsoup4==4.11.2
bleach==6.0.0
cachetools==5.3.0
certifi==2022.12.7
cffi==1.15.1
charset-normalizer==3.1.0
cloudpickle==2.2.1
colorama==0.4.4
colorlog==4.7.2
comm==0.1.2
contourpy==1.0.7
coverage==7.2.1
crispy-bootstrap4==2022.1
cryptography==39.0.2
cuda-python==12.1.0
cycler==0.11.0
Cython==0.29.34
debugpy==1.6.6
decorator==5.1.1
```

Código A.2: Fichero requirements (parte 2).

```
defusedxml==0.7.1
dj-database-url==1.2.0
dj-static==0.0.6
Django==4.1.7
django-allauth==0.52.0
django-bootstrap4==3.0.1
django-crispy-forms==2.0
django-extensions==3.2.1
django_debug_toolbar==3.8.1
et-xmlfile==1.1.0
executing==1.2.0
Faker==17.6.0
fastjsonschema==2.16.3
flake8==6.0.0
fonttools==4.39.0
fqdn==1.5.1
future==0.18.3
google-auth==2.16.2
google-auth-oauthlib==0.4.6
grpcio==1.51.3
gunicorn==20.1.0
hyperopt==0.2.7
idna==3.4
image==1.5.33
importlib-metadata==6.0.0
importlib-resources==5.12.0
ipykernel==6.21.3
ipython==8.11.0
ipython-genutils==0.2.0
ipywidgets==8.0.4
isoduration==20.11.0
jedi==0.18.2
Jinja2==3.1.2
joblib==1.2.0
jsonpointer==2.3
jsonschema==4.17.3
jupyter==1.0.0
jupyter-console==6.6.3
jupyter-events==0.6.3
jupyter_client==8.0.3
jupyter_core==5.2.0
jupyter_server==2.4.0
jupyter_server_terminals==0.4.4
jupyterlab-pygments==0.2.2
jupyterlab-widgets==3.0.5
kiwisolver==1.4.4
Markdown==3.4.1
MarkupSafe==2.1.2
matplotlib==3.6.0
matplotlib-inline==0.1.6
```

Código A.3: Fichero requirements (parte 3).

```
mccabe==0.7.0
mistune==2.0.5
nbclassic==0.5.3
nbclient==0.7.2
nbconvert==7.2.9
nbformat==5.7.3
nest-asyncio==1.5.6
networkx==3.0
notebook==6.5.3
notebook_shim==0.2.2
numpy==1.19.5
nvidia-cublas-cu11==11.10.3.66
nvidia-cuda-nvrtc-cu11==11.7.99
nvidia-cuda-runtime-cu11==11.7.99
nvidia-cudnn-cu11==8.5.0.96
oauthlib==3.2.2
openpyxl==3.1.2
packaging==23.0
pandas==1.4.4
pandocfilters==1.5.0
parso==0.8.3
pexpect==4.8.0
pickleshare==0.7.5
Pillow==9.4.0
pkgutil_resolve_name==1.3.10
platformdirs==3.1.0
prometheus-client==0.16.0
prompt-toolkit==3.0.38
protobuf==4.22.0
psutil==5.9.4
psycopg2-binary==2.9.5
ptyprocess==0.7.0
pure-eval==0.2.2
py4j==0.10.9.7
pyasn1==0.4.8
pyasn1-modules==0.2.8
pycodestyle==2.10.0
pycparser==2.21
pyflakes==3.0.1
Pygments==2.14.0
PyJWT==2.6.0
pyparsing==3.0.9
PyQt5==5.15.9
PyQt5-Qt5==5.15.2
PyQt5-sip==12.11.1
pyrsistent==0.19.3
python-dateutil==2.8.2
python-json-logger==2.0.7
python3-openid==3.2.0
pytz==2022.7.1
```

Código A.4: Fichero requirements (parte 4).

```
PyYAML==6.0
pyzmq==25.0.0
qtconsole==5.4.0
QtPy==2.3.0
recbole==1.0.0
requests==2.28.2
requests-oauthlib==1.3.1
rfc3339-validator==0.1.4
rfc3986-validator==0.1.1
rsa==4.9
scikit-learn==0.24.2
scipy==1.6.0
seaborn==0.12.2
Send2Trash==1.8.0
simplejson==3.18.3
six==1.16.0
sniffio==1.3.0
soupsieve==2.4
sqlparse==0.4.3
stack-data==0.6.2
static3==0.7.0
tensorboard==2.12.0
tensorboard-data-server==0.7.0
tensorboard-plugin-wit==1.8.1
terminado==0.17.1
text-unidecode==1.3
threadpoolctl==3.1.0
tinycss2==1.2.1
torch==1.13.1
torch-geometric==2.2.0
tornado==6.2
tqdm==4.65.0
traitlets==5.9.0
typing_extensions==4.5.0
uri-template==1.2.0
urllib3==1.26.14
wcwidth==0.2.6
webcolors==1.12
webencodings==0.5.1
websocket-client==1.5.1
Werkzeug==2.2.3
whitenoise==6.4.0
widgetsnbextension==4.0.5
xgboost==1.7.4
zipp==3.15.0
```


UÁM
Universidad Autónoma
de Madrid