UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

19
20

Trabajo fin de grado

Estudio de métodos de detección de patrones de movimiento para sistemas de recomendación turísticos.

Sergio Torrijos López de la Manzanara

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

excelencia Campus Internacional UAM CSIC

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Estudio de métodos de detección de patrones de movimiento para sistemas de recomendación turísticos.

Autor: Sergio Torrijos López de la Manzanara
Tutor: Alejandro Bellogín Kouki

junio 2020

Sergio Torrijos López de la Manzanara
Estudio de métodos de detección de patrones de movimiento para sistemas de recomendación turísticos.

Sergio Torrijos López de la Manzanara
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi familia y amigos

". . . the only way to be truly satisfied is to do what you believe is great work. And the only way to do great work is to love what you do. If you haven't found it yet, keep looking. Don't settle. As with all matters of the heart, you'll know when you find it."

Steve Jobs

Agradecimientos

En primer lugar me gustaría expresar mi agradecimiento a Alejandro Bellogín por su compromiso con este proyecto, que se ha visto reflejado en su incansable esfuerzo y dedicación, brindándome su disponibilidad en cualquier momento. En definitiva, muchas gracias por todo.

Quiero destacar el apoyo de mi familia durante toda la carrera y en la travesía del desarrollo de este trabajo, ayudándome en los momentos más difíciles. Me tomo el honor de agradecer a mi hermana María, por habérselo leído con dedicación.

Me gustaría también resaltar a mis amigos y compañeros de trabajo, por entender mi estrés y ayudarme dándome consejos y siendo flexibles para que pudiese sacar adelante todo este trabajo.

Finalmente, quiero tener un especial recuerdo para mi compañero de prácticas a lo largo de toda la carrera, Sergio Salcedo, el cual ha estado como pilar básico durante toda esta y ha contribuido en gran parte a mis buenos resultados.

Resumen

El turismo representa una gran fuente de ingresos para muchas ciudades alrededor del mundo, y ha visto multiplicada su popularidad gracias a la facilidad con la que se pueden exponer en internet. En estos tiempos es poco habitual llegar a una nueva ciudad, o incluso a una ya conocida, sin tener una mínima planificación de dónde ir. Es por ello que muchas aplicaciones como FourSquare, dedican gran parte de sus esfuerzos a mostrar de la manera más precisa los lugares de interés más relacionados con los intereses del usuario y ofrecérselos cuando intente planificar una ruta en su viaje o un próximo destino.

Este Trabajo de Fin de Grado consiste en utilizar toda la información obtenida de aplicaciones como FourSquare o Yelp, donde los usuarios registran su acceso a una localización, y a partir de esta inferir cómo se desplaza el usuario y con qué otros usuarios comparte trayectorias. Esto nos permitirá encontrar usuarios comunes, no necesariamente conocidos, los cuales se utilizarán para proporcionar recomendaciones más precisas.

A lo largo del trabajo se ha estudiado una gran cantidad de algoritmos de análisis de trayectorias, como Flock, Convoy, ST-DBSCAN, Dynamic Time Warping o Hausdorff. Además, se ha propuesto un algoritmo base para tomarlo como referencia. Del mismo modo se han evaluado todos los rendimientos de los algoritmos adaptados al dataset utilizado para tratar de ver cuál encajará mejor en una relación de eficiencia y rendimiento. Todos los resultados obtenidos se han utilizado para crear un recomendador basado en vecinos próximos utilizando las salidas proporcionadas por los diferentes métodos de estudios de trayectorias, comparándolas con las salidas por otros recomendadores sin utilizar estas aproximaciones. Gracias a este estudio, se han publicado dos artículos de investigación cortos en sendas conferencias, reconociendo la novedad de estas aproximaciones.

Por último, este trabajo ha permitido abrir un campo de estudio aún muy novedoso dando pie a futuras investigaciones. Se ha comprobado que la aproximación de las recomendaciones mediante el estudio de trayectorias es un ámbito con un gran potencial y abrirá paso a mejores recomendaciones en el futuro.

Palabras clave

Sistema de recomendación, trayectoria, patrones de movimiento, algoritmo, vecinos próximos (KNN), FourSquare, dataset, POI.

Abstract

Tourism represents a great source of income for many cities around the world, and its popularity has multiplied thanks to the ease with which it can be exposed on the internet. In these times it is unusual to arrive to a new city, or even to an already known one, without a minimum planning of where to go. That is why many applications such as FourSquare devote a large part of their efforts to show the places of interest most related to the user's interests in the most precise way, and offering them when they try to plan a route in their trip or next destination.

This Bachelor Thesis consists of using all the information obtained from applications such as FourSquare or Yelp, where users register their access to a location, and from this, to infer how the user moves and with which other users they may share trajectories. This will lead us to find common, not necessarily known, users which will be used to provide more accurate recommendations.

Throughout the work, a large number of trajectory and co-movement analysis algorithms have been studied, such as Flock, Convoy, ST-DBSCAN, Dynamic Time Warping, and Hausdorff. Besides, a base algorithm has been proposed to take as a reference. At the same time, we have evaluated the performance of these algorithms adapted to the used dataset to analyze which one will fit better under an efficiency and performance tradeoff. All the results obtained have been used to create a nearest neighbors recommender using the outputs provided by the studied trajectory methods, comparing them with the outputs by other recommenders without using these approaches. Thanks to this study, two short research articles have been published in international conferences, acknowledging the novelty of these approaches.

Finally, this work has opened up a field of study that is still very novel, giving rise to future research. It has been proven that the approximation of the recommendations through the study of trajectories is an area with great potential and will open the way for better recommendations in the future.

Keywords

Recommender systems, trajectory, co-movement patterns, algorithm, nearest neighbors (KNN), FourSquare, dataset, POI.

Índice

Listas

Lista de ecuaciones

Lista de figuras

Lista de tablas

1 Introducción

Con el avance de internet, se ha reinventado la manera de hacer turismo. En la actualidad ya no viven únicamente del turismo aquellos lugares de interés, sino también aquellas empresas encargadas de poner en contacto a las personas con dicho punto de interés, y así recomendar nuevas localizaciones personalizadas al usuario. Algo muy común en las aplicaciones de turismo son los sistemas de recomendación, cada vez más presentes en nuestro día a día. Muchas veces nos encontramos con aplicaciones como FourSquare cuando buscamos nuevos sitios que visitar durante nuestro viaje, ofreciéndonos lugares que considera más afines a nosotros. Esto es llevado a cabo gracias a los sistemas de recomendación. Normalmente estos sistemas aportan recomendaciones en base a los gustos de personas conocidas o lugares previamente visitados, pero es un campo totalmente nuevo de estudio ofrecer recomendaciones en base a la similitud entre las trayectorias de los usuarios.

Los sistemas de filtrado colaborativo pueden considerarse como los primeros y más ampliamente implementados sistemas de recomendación [1], sugiriendo elementos interesantes a los usuarios según las preferencias de personas similares o relacionadas [2]. Gracias a esto, un usuario sabe que cuando va de turismo a una nueva ciudad puede mirar a través del móvil qué ruta podría planificarse para realizar durante su estancia. Dentro de este ámbito destacan aplicaciones como FourSquare, Tripadvisor o Yelp, muy populares hoy día. Por todo esto, el objetivo principal y la competición entre dichas compañías es ofrecer la mejor experiencia al usuario alcanzando las recomendaciones más personalizadas y precisas, y por lo tanto tratando de conocer mejor al usuario.

1.1. Motivación del proyecto

Debido al creciente número de usuarios registrados en las redes sociales basadas en ubicación (*Location Based Social Networks*, LBSN), donde los usuarios comparten los lugares o puntos de interés (POI, del inglés *Point Of Interest*) que visitan; la recomendación de puntos de interés se ha vuelto particularmente útil y varios enfoques específicos se han propuesto en los últimos años. En particular, estos enfoques tienden a incorporar propiedades inherentes a estos sistemas, como la información social, geográfica o temporal [3, 4].

Sin embargo, a pesar de existir una extensa literatura sobre métodos de minería de patrones de movimiento, rara vez se han utilizado para recomendación. En este trabajo presentaremos la posibilidad de integrar patrones de movimiento de usuarios para la recomendación. Para ello se calcularán usuarios similares a partir de las trayectorias para introducirlos en los sistemas de recomendación.

La literatura sobre minería de patrones de movimiento proporciona varios métodos para calcular similitudes entre trayectorias o en un contexto más general, mover objetos, por ejemplo, para agru-

parlos o identificarlos y analizar su comportamiento [5]. Ejemplos de aplicaciones y servicios podrían ser análisis de animales, recomendaciones sociales y planificación del tráfico, donde los datos llegarían de chips para telemetría animal, dispositivos portátiles o vehículos con GPS [6]. Muchos de estos problemas se resuelven en el ámbito de descubrir patrones de co-movimiento, que se refiere a encontrar grupos de objetos que viajan juntos por un cierto período de tiempo, explotando las dimensiones espacio-temporales. Existen varios métodos de patrones de co-movimiento, como Flock, Convoy, Swarm o ST-DBSCAN [6–8]. Todos ellos definen diferentes restricciones sobre lo que interpretan como un grupo o un co-movimiento.

1.2. Objetivos

Como se ha introducido anteriormente, el objetivo principal del trabajo es conseguir unas recomendaciones basadas en la similitud con usuarios afines, a modo de obtener unas recomendaciones de puntos de interés más precisas. Para ello, en primer lugar analizaremos distintos algoritmos de patrones de movimiento ya existentes en la literatura, y en un segundo lugar presentaremos algunos nuevos algoritmos diseñados para tratar de solucionar las limitaciones de tiempo al ejecutarlos.

1.3. Estructura del trabajo

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

- **Capítulo 1. Introducción:** Descripción actual de las implementaciones realizadas hasta la fecha en recomendación de lugares de interés, motivación del proyecto y objetivos del mismo.
- **Capítulo 2. Estado del arte:** Introducción a los patrones de movimiento, sistemas de recomendación, y los diferentes algoritmos.
- **Capítulo 3. Diseño e Implementación:** Explicación de los módulos en los que se ha diseñado el proyecto partiendo de la base teórica del estado del arte. Se justificarán también todas las decisiones de diseño e implementación tomadas, adjuntando diagramas explicativos.
- **Capítulo 4. Pruebas y resultados:** Se evaluará la validez de la propuesta e implementación mediante diversas pruebas y comparativas con otras aproximaciones al problema.
- **Capítulo 5. Conclusión y Trabajo Futuro:** Explicación de las conclusiones obtenidas tras la realización del proyecto y modificaciones propuestas para continuar la investigación en dicha área.

2

Estado del arte

En este capítulo se explicarán las diferentes técnicas de procesamiento de datos aplicadas, los patrones de movimiento más relevantes a la hora de buscar trayectorias espacio-temporales similares entre objetos, y su posterior aplicación a los sistemas de recomendación. Por último se presentarán los conceptos y términos básicos para entender la metodología que utilizaremos en nuestra evaluación.

2.1. Patrones de movimiento

El estudio de las trayectorias está en pleno auge desde el surgimiento de los dispositivos móviles, debido a la gran cantidad de datos que estos envían, lo cual permite grandes avances. En este trabajo definimos una trayectoria como el movimiento de un objeto de un punto A a un punto B en un determinado instante de tiempo.

Un patrón de movimiento se define como un grupo de objetos viajando juntos, normalmente agrupado por su proximidad espacial, durante un periodo de tiempo. Un patrón es representativo si el tamaño del grupo excede una longitud M (mínima cardinalidad del cluster espacial) y su duración una longitud K (mínimo número de ocurrencias en el cluster), donde K y M son parámetros introducidos por el usuario.

A la hora de buscar patrones de movimiento espacio-temporales similares entre objetos, uno de los aspectos fundamentales a tener en cuenta es el tiempo necesario para ejecutar el algoritmo, con el reto de conseguir tiempos bajos sin paralelizar con herramientas como Spark, como ya se ha hecho en [6].

A lo largo de este proyecto, se abordarán diversas técnicas como Flock, Convoy, ST-DBSCAN [6–8] y se propondrán soluciones mediante algoritmos de análisis de series temporales como DTW y Hausdorff [9–11]. Previo a esto, en las siguientes secciones se expondrán las bases para entender estos algoritmos.

Consideramos que dos objetos siguen una trayectoria similar cuando, fijando unos umbrales de tiempo, encontramos dos objetos que siguen unos patrones similares a la hora de desplazarse por diferentes localizaciones. Los patrones se ejecutan buscando la trayectoria común más larga, para tratar de maximizar la similitud entre objetos. Debido al alto coste computacional de estos, se definirán diversas formas de reducirlo al máximo. A lo largo de nuestro trabajo consideraremos que cada trayectoria tiene una longitud diferente y puede comenzar y finalizar en cualquier momento.

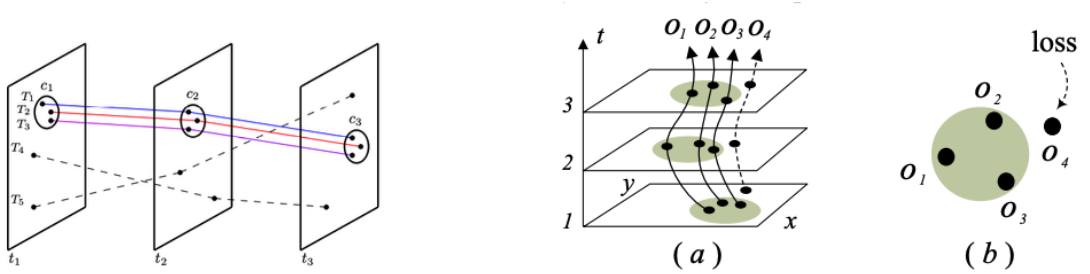
2.1.1. Algoritmos principales

Flock

Un *flock* [7] se entiende como un conjunto de puntos que se mueven juntos en un disco de tamaño fijo especificado, compartiendo un determinado patrón de movimiento entre sí, es decir, siguen una trayectoria similar. En este caso son puntos que viajan en coordenadas espacio-temporales. Este algoritmo procesa los datos en función de los parámetros:

- ε: Distancia entre los elementos móviles (radio).
- μ: Número mínimo de objetos móviles.
- δ: Intervalo de tiempo definido mínimo entre elementos.

Una condición que exige Flock es que los elementos encontrados deben compartir trayectorias sin saltos temporales, es decir, en el momento que ambos no visiten una localización común se acabará la trayectoria. Flock realiza la **agrupación de ítems en función de las distancias**. Para quedarse con la secuencia mayor de elementos comunes, la librería utilizada es LCM [12], eliminando la redundancia de subconjuntos de elementos ya incluidos en uno mayor. Su principal problema es el tiempo de ejecución, y es por ello por lo que se proponen algoritmos como SPARE [6].



(a) Trayectoria Flock [7] donde se muestran cómo se escogen los puntos para la trayectoria en función de si cumplen el criterio del radio del disco.

(b) Demostración de la pérdida de elementos ocasionada por Flock [13], donde se aprecia que el punto *O4* no entra en las restricciones de Flock pese a seguir una trayectoria similar.

Figura 2.1: Análisis de las trayectorias escogidas por Flock, marcando con círculos verdes los discos en los que Flock busca los vecinos, y cada uno de los cuadrados un instante de tiempo.

En la Figura 2.1(a) se muestra una trayectoria obtenida utilizando la técnica Flock. Aquí se pueden ver representados los 3 discos, uno para cada instante de tiempo, y cómo se eligen los objetos o se descartan dependiendo de si entran o no en las condiciones del disco. Así, se observa que el objeto T4 y T5 son descartados por no seguir las restricciones. El mayor problema de Flock, como podemos ver en la Figura 2.1(b), es que puede producirse una pérdida de datos que realmente sí seguían la trayectoria, por no adaptar el disco conforme a la densidad espacial de puntos.

Convoy

Convoy [6, 13] se presenta como alternativa a la rigidez de Flock, basada en el tamaño del disco provocando pérdidas como se ha argumentado en la Figura 2.1(b). Como alternativa a la rigidez en la búsqueda de vecinos y longitud de la trayectoria de Flock, Convoy permite encontrar patrones genéricos de cualquier forma y duración. Este concepto se basa en la **agrupación en función de**

la densidad espacial. Esta permite encontrar una mayor agrupación de patrones gracias a la relajación de algunas condiciones respecto a Flock. Dado un conjunto de trayectorias, O, un entero m, una distancia e y un tiempo de vida k, Convoy agrupa los elementos en al menos m objetos llamados *objetos densamente-conectados* con distancia e durante k puntos consecutivos. Por ello, se puede definir que dos elementos están densamente conectados si existe una secuencia de objetos que conecta los puntos y la distancia entre ellos no excede e.

A continuación, se definen algunos conceptos que se repetirán en los siguientes algoritmos [14]:

Density-reachable: Un punto p se dice que es alcanzable (*density-reachable*) desde un punto q si el punto p está a una distancia ε del punto q, y q tiene un número suficiente de puntos en sus vecinos a una distancia ε. Este concepto se puede ver referenciado en la Figura 2.3 (a).

Density-Connected: Dado un conjunto de puntos S, un punto $p \in S$ está densamente conectado a un punto $q \in S$ con respecto a e y m si existe un punto $x \in S$ tal que ambos p y q son alcanzables desde x. Se puede apreciar cómo los objetos están densamente conectados en la Figura 2.3 (b) y Figura 2.3 (c).

Core object: Un objeto core se refiere al punto al cual sus vecinos en un radio ε tienen que contener al menos un numero *minPts* mínimo de puntos de otros elementos, es decir, sus vecinos conectados deben satisfacer la condición de contener al menos *minPts*. Esto se puede ver reflejado en la Figura 2.3 (c).

Border-object: Un objeto p es un *border-object* si no es un *core-object* pero es *density-reachable* desde otro *core-object*.

Búsqueda Convoy: Dado un conjunto de trayectorias de N objetos, un umbral de distancia ε, un entero m y una longitud de tiempo k, Convoy devuelve todos los grupos de objetos posibles tal que cada grupo consista en un grupo máximo de puntos densamente conectados con respecto a e y m durante al menos k puntos consecutivos. Un ejemplo de esto puede verse en la Figura 2.2 donde con unos parámetros $m = 2$, $k = 3$, vemos que O_2 y O_3 pertenecen al mismo Convoy durante los puntos consecutivos de t_1 a t_3.

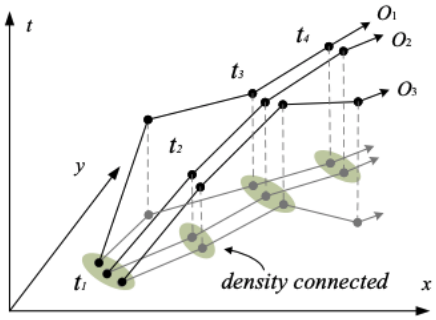


Figura 2.2: Ejemplo de Trayectoria Convoy [13], marcando en verde los discos que agrupan los puntos y las líneas marcando la trayectoria.

ST-DBSCAN

ST-DBSCAN [8, 14] es un método de agrupación espacio-temporal basado en densidad, al igual que Convoy, utilizando el concepto de que los puntos similares en una zona forman un cluster con una densidad alta. Los parámetros utilizados para definir la densidad son:

- ε: El radio espacial que delimita los puntos.
- *minEps*: Número mínimo de puntos para definir el cluster.
- *temporal_threshold (th)*: Ventana temporal que delimita los instantes de tiempo de los puntos.

Tomando el cluster como cada localización, se define la similitud de los objetos en función del número de ubicaciones que han compartido en un cierto margen de tiempo.

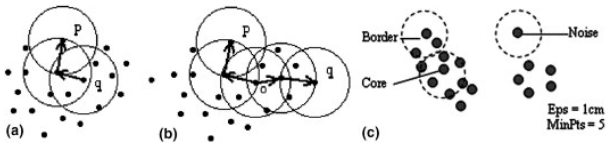


Figura 2.3: Ejemplo de cluster de puntos ST-DBSCAN [14], donde se aprecian los círculos delimitando el radio a buscar y cada uno de los puntos para definir cuáles entran en el criterio.

ST-DBSCAN comienza con el primer punto p en el conjunto de trayectorias y busca todos los puntos *density-reachable* desde p con respecto a ε. Si cumple el criterio de búsqueda delimitado por los parámetros y p es un *core-object* se crea un cluster, y el punto p y sus vecinos se asignan al cluster. Si p es un *border-object* al no tener puntos *density-reachable* desde p el algoritmo visita el siguiente punto del conjunto. Esto se repetirá de manera iterativa hasta que todos los puntos se hayan procesado.

Métodos de similitudes entre trayectorias

De manera alternativa a los métodos de patrones de co-movimiento, se pueden utilizar métricas para calcular la similitud entre las trayectorias. Asumiendo que los objetos son similares si sus trayectorias son similares, se puede calcular su similitud mediante funciones matemáticas como Hausdorff [15] y Dynamic Time Warping (DTW) [10, 11], variando la función de similitud a sus formas como coseno o euclídea.

Hausdorff:

Mide la distancia de dos subconjuntos compactos dentro de un espacio métrico. De manera informal, dos conjuntos están cerca según la distancia de Hausdorff si cada punto de cualquiera de los conjuntos está cerca de algún punto del otro conjunto. En este sentido, la distancia de Hausdorff sería la distancia más larga que nos podría obligar a viajar un adversario eligiendo un punto en uno de los dos conjuntos, desde donde deberíamos viajar al otro conjunto. Si dist(x,y) denota la distancia en M, podemos definir [9]:

$$d_H(X,Y) = \max\{d_1, d_2\}, \qquad \begin{cases} d_1 = \sup_{x\in X} \inf_{y\in Y} \operatorname{dist}(x,y) \\ d_2 = \sup_{y\in Y} \inf_{x\in X} \operatorname{dist}(x,y)\} \end{cases}$$

Figura 2.4: Fórmula de distancia Hausdorff [9].

Dynamic Time Warping (DTW) [10]:

DTW es un algoritmo de deformación no lineal que descompone un problema complejo de optimización global en problemas de optimización local utilizando el principio de programación dinámica. DTW tiene la ventaja de la flexibilidad de tiempo y logra mejores resultados que la distancia euclidiana al permitir flexibilidad para buscar el punto más conveniente con el que calcular la distancia, como

vemos en la Figura 2.5:

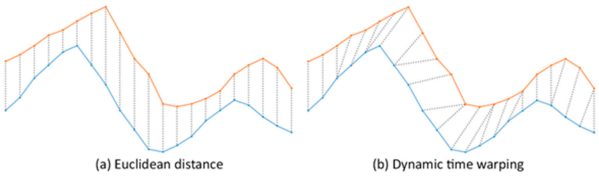


Figura 2.5: Comparación entre dos secuencias: (a) mientras la distancia euclídea es rígida en el tiempo, (b) Dynamic Time Warping (DTW) es flexible en el tiempo para tratar la posible distorsión de tiempo entre las secuencias [10].

2.1.2. Librerías para el cálculo de patrones de movimiento

A continuación se describirán algunas de las librerías de patrones de movimiento encontradas en la literatura a través de búsquedas en internet y repositorios públicos (GitHub), y se detallarán los motivos que nos han llevado a escoger las de nuestro estudio:

LCM

Librería escrita en C utilizada para ejecutar Flock. Nos permite encontrar el conjunto de patrones frecuentes máximos en tiempo lineal, buscando la secuencia más larga.

tungk/TrajectoryMining [16]

Proyecto en GitHub que contiene gran cantidad de patrones de movimiento en Java, como Flock, Convoy, Swarm, DBSCAN o Group. En una última etapa descartamos esta librería ya que estaba implementada en Java y nos encontramos con diversos problemas en su ejecución, por lo que preferimos buscar implementaciones en Python.

FPFlock [17]

Entre las escasas implementaciones de Flock en Python encontramos este repositorio que utilizaba algoritmos como LCM para encontrar las trayectorias comunes entre objetos más largas. Decidimos utilizarlo ya que ofrecía unos resultados competitivos pese a consumir muchos recursos.

Coherent Moving Cluster algorithm [18]

Una de las escasas implementaciones de Convoy encontradas en GitHub con lenguaje base Python. Ofrece también la implementación de la cual se partió en Java, y lo elegimos por ser una implementación básica de Convoy en Python utilizando DBSCAN. Tomada como punto de partida para, aplicando unas modificaciones, adaptarlo a nuestro problema.

py-st-dbscan [19]

Hay multitud de implementaciones de DBSCAN en código abierto como sklearn y la librería en GitHub py-st-dbscan. La ventaja de esta primera es principalmente el ser una librería proporcionada

por la reputada organización Scikit-learn, pero a pesar de ello no cumplía nuestros objetivos, ya que se buscaba analizar trayectorias espacio-temporales, y esta solo nos ofrecía análisis espacial, es decir, analizar trayectorias en base a su latitud y longitud. Por todo esto escogimos la librería py-st-dbscan, al ser de las pocas disponibles en GitHub que ofrecían análisis espacio-temporales.

jJimo/Trajectory-Analysis-and-Classification-in-Python-Pandas-and-Scikit-Learn- [20]

Librería implementada en Python donde se exponen diversos algoritmos para calcular la similitud entre trayectorias mediante los algoritmos *Fast Dynamic Time Warping (Fast-DTW)* y *Longest Common Subsequence*. De esta extrajimos la idea de utilizar DTW para realizar la búsqueda mucho más rápida entre vecinos.

Shathra/comparing-trajectory-clustering-methods [21]

Esta librería escrita en Python ofrece la novedad de clusterizar trayectorias, eliminando datos redundantes, donde si había demasiados puntos contiguos sin desviarse X grados determinados podrían simplificarse ubicando únicamente el punto original y final, como vemos en la figura 2.6.

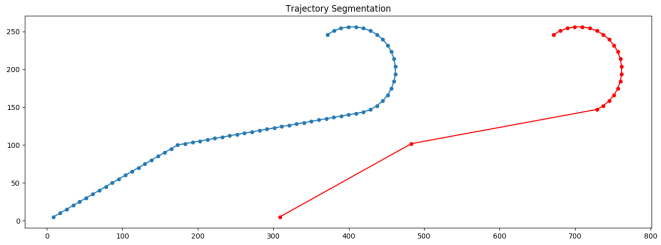


Figura 2.6: Representación de una trayectoria con puntos continuos para aplicar su segmentación y simplificarla [21].

Como se verá en la sección 3.3.1, esta simplificación no va a ser útil para nuestro problema, pero se quiso considerar en un primer momento por su potencial para reducir el coste computacional.

Librerías escogidas

De todas estas librerías, escogimos LCM, Pygmaps, Coherent Moving Cluster, py-st-dbscan y FP-Flock ya que se encontraban implementadas en Python y nos servirían de base para comenzar con una buena implementación, haciendo diversos cambios para adaptarlas a los requisitos del proyecto.

2.2. Obtención y procesamiento de trayectorias

Para poder trabajar con diferentes algoritmos, los cuales necesitaban los datos procesados de forma distinta, se presentan aquí los métodos que se han estudiado para transformar los datos. Además, se presenta un mecanismo habitual en recomendación para garantizar que los algoritmos tienen suficientes datos de los usuarios y los ítems que permite reducir el tamaño de los datos, lo cual es útil para algunos de los métodos con mayor carga computacional.

2.2.1. Partición de trayectorias por instantes de tiempo

Al estar analizando trayectorias espacio-temporales, no se podría hacer una partición de entrenamiento y test de manera aleatoria como si sería posible en otras situaciones, puesto que se dejarían trayectorias incompletas. De este modo, para realizar particiones en primer lugar examinaremos los usuarios que tengan más de X puntos de interés visitados, ya que no tiene sentido entrenar con usuarios que posteriormente no aparecerán en test. Por ello organizaremos los usuarios de forma cronológica por timestamp, y a partir de aquí calcularemos el porcentaje de entrenamiento y test aplicado a la longitud de la trayectoria de cada usuario.

Timestamp

Con el fin de comprender cómo trataremos posteriormente los timestamps para escoger los umbrales, se van a definir cómo están codificados. Estos vienen codificados en *Unix epoch time*, y se pueden convertir a formato legible con la página `https://www.epochconverter.com/`.

Por ejemplo *1374323535* se convierte a: *sábado, 20 de julio de 2013 14:32:15 GMT+02:00 DST*, y para utilizarlo como δ de tiempo se utilizará la unidad segundos.

En este trabajo, las marcas de tiempo (timestamp) se han utilizado con dos propósitos diferenciados:

1.– Agrupar las trayectorias por timestamp de 8 horas, de modo que se considere como una sola trayectoria todas las ubicaciones detectadas en una ventana de 8 horas.

2.– Convertir los timestamp de un usuario en una secuencia, es decir, ordenar todos los sitios visitados por un usuario según su timestamp en un orden temporal ascendente, y convertir estos datos en una serie temporal comenzando en 1 para cada usuario. De este modo tendremos una versión más legible y aplicable cuando algún algoritmo necesite información secuencial y no el instante de tiempo exacto en el que se produjo una acción.

2.2.2. Cluster de K-Cores

Para conseguir que los algoritmos tengan un mínimo de información tanto sobre los usuarios como de los ítems, aplicamos la llamada técnica de agrupación (o *clustering*) K-Cores. La técnica K-Cores surgió de la mecánica estadística para investigar las propiedades de los grafos. Este algoritmo, en una primera ronda, elimina todos los nodos y sus conexiones que contengan una o menos conexiones, es decir, vecinos. En una segunda ronda todos los nodos con dos o menos ramas se eliminan. Siguiendo esta mecánica, se realizan las N iteraciones donde todos los nodos con menos de K vecinos se eliminan del grafo.

Como se puede ver en la Figura 2.7, se recorrerán 3 iteraciones del algoritmo. En la primera los nodos azules se eliminan, en la segunda los nodos amarillos y finalmente en la última quedan los nodos rojos que son el resultado de este procesado de datos.

Como consecuencia, la aplicación de esta técnica nos permitirá reducir el tamaño del dataset, consiguiendo, por ejemplo, que algoritmos computacionalmente muy demandantes como Flock, puedan procesar un dataset muy grande.

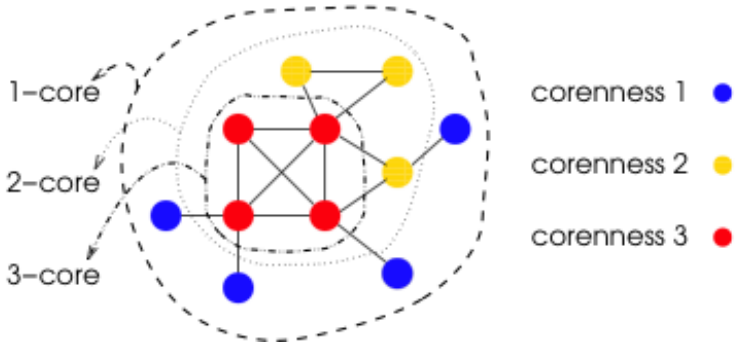


Figura 2.7: Procesado de K-Cores sobre un grafo [22].

2.3. Sistemas de recomendación

Los sistemas de recomendación son una herramienta presente en prácticamente cualquier servicio de internet debido al potencial de obtener los ítems más similares a los gustos de los usuarios. Esto engloba el dominio en prácticamente cualquier tipo de problemas, tanto artículos de compra como películas, etc. Un ejemplo de esto es cuando un usuario ve una película de una cierta temática, es muy posible que este encuentre recomendaciones de películas similares dentro de esa temática adaptadas para dicho usuario.

La recomendación es una rama en constante innovación y en pleno auge debido a los grandes beneficios devueltos, buscando cada vez nuevas técnicas y algoritmos. En este trabajo presentaremos la novedad de incluir el estudio de trayectorias de usuarios para conseguir recomendaciones más precisas.

2.3.1. Sistemas de recomendación de filtrado colaborativo

Los sistemas de recomendación de filtrado colaborativo realizan predicciones de los intereses de un usuario en base a los intereses obtenidos de muchos otros usuarios. Para estas recomendaciones es necesario un mecanismo para estimar un modelo estadístico a partir de los datos (métodos basados en modelos) o bien un método que determine qué elementos son similares entre sí, y a partir de esto realizar las predicciones (métodos basados en memoria).

Métodos basados en modelos

Frente a las estrategias basadas en vecinos, las basadas en modelo permiten reducir la complejidad del cálculo. Con el filtrado basado en modelos, se construyen modelos estadísticos de patrones de valoraciones de los usuarios e ítems, para realizar así las predicciones. Para encontrar estos patrones frecuentemente se utilizan algoritmos de aprendizaje automático. Entre los más comunes destacan: las redes neuronales, modelos probabilísticos o modelos de factores latentes. Este último destaca especialmente por la factorización de matrices.

Factorización de Matrices [23]

Para la factorización de matrices, las interacciones entre usuarios y productos se agrupan formando una matriz R, con tantas filas como usuarios y columnas como productos. Esta estará llena de 0, y tan

solo tendrá 1 en aquellos casos que haya habido interacción. Este algoritmo se encarga de completar la matriz de clasificación, *R* y proporcionar una predicción.

La factorización de matrices consiste en descomponer la matriz original, *R*, en dos sub-matrices más pequeñas, *U* y *V*, como se ve en la Figura 2.8. La recomendación se obtiene con la multiplicación de ambas. Como estas matrices son mucho más pequeñas que la original, su producto dará como resultado una matriz de clasificación sin ceros.

Es importante recalcar que gracias a esto se consigue reducir una matriz de decenas de millones, R, a dos sub-matrices menores, más fácilmente computables.

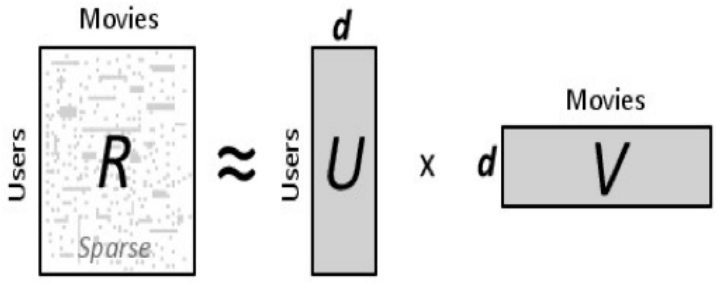


Figura 2.8: Representación de la factorización de matrices [24].

El rating predicho vendrá definido por el producto escalar de ambos vectores, siguiendo la fórmula:

$$\hat{r}(u,i) = x_u^T \cdot y_i \tag{2.1}$$

Siendo x_u el vector correspondiente al usuario u-ésimo aprendido por la matriz U que representa el interés del usuario u por los d factores, y el vector y_i con el mismo tamaño que el vector anterior, pero esta vez correspondiente al ítem i-ésimo de la matriz V representando cada coordenada el interés del ítem i por cada uno de los factores.

Métodos basados en memoria [25]

Métodos similares a vecinos próximos (KNN) en clasificación, en los cuales la similitud se basa en ratings de otros usuarios. Estos métodos son los más comunes, y utilizan los datos recogidos para calcular la similitud entre usuarios o ítems. De este modo se extrae un listado de los *K* elementos más similares. Estos, en detalle, representan la recomendación basada en:

- Usuarios, donde el algoritmo recomienda los ítems que le han gustado a usuarios parecidos.
- Ítems, donde el algoritmo recomienda los ítems que se parecen a otros ítems que le han gustado previamente al usuario.

Algunas ventajas de los métodos basados en memoria son:

- Facilidad para funcionar con ítems únicamente con los ratings asignados por usuarios, sin ninguna descripción.
- Buenos niveles de acierto.
- Más novedad basada en la experiencia de los usuarios.
- Recomendación más diversa que en el basado en contenido.

Algunos inconvenientes de estos son:

- Pueden haber usuarios no comparables entre sí, como en los siguientes casos: cuando no existe solapamiento, no hayan puntuado el ítem cuyo rating queramos predecir, o el vecindario es demasiado pequeño.
- Poco escalable en el caso de tener sistemas de millones de elementos al tener que buscar en la tabla de puntuaciones.
- Solo se tiene en cuenta las puntuaciones, no el contexto.

Predicción de rating y similitud entre usuarios

Las fórmulas utilizadas para el rating estimado son:

$$\hat{r}(u,i) = C \times \sum_{v \epsilon N_k(u), r(v,i) \neq \varnothing} sim(u,v) r(v,i) \tag{2.2}$$

Siendo C necesaria para obtener ratings en el rango normalizado.

$$C = \frac{1}{\sum_{v \epsilon N_k(u), r(v,i) \neq \varnothing} |sim(u,v)|} \tag{2.3}$$

Siendo $N_k(u)$ el top-k de vecinos más similares a u.

Predicción de rating y similitud entre ítems

Similar al basado en usuario, pero ahora teniendo en cuenta que la importancia que le puede dar el usuario al nuevo ítem se calculará mediante la importancia dada a ítems parecidos a este.

$$\hat{r}(u,i) = C \times \sum_{r(u,j) \neq \varnothing} sim(i,j) r(u,j) \tag{2.4}$$

Siendo C necesaria para obtener ratings en el rango normalizado.

$$C = \frac{1}{\sum_{r(u,j) \neq \varnothing} |sim(i,j)|} \tag{2.5}$$

Funciones de similitud

Las funciones utilizadas más frecuentes para el cálculo de la similitud, tanto las basadas en usuario como en ítem, son:

- **Coseno**: Es la más frecuente, donde la similitud entre dos elementos está basada en la distancia coseno entre los dos vectores a comparar. Estos pueden representar los ratings de cada uno de los usuarios con respecto a un determinado ítem.

$$sim(u,v) = \frac{\sum_{v \epsilon i: r(u,i) \neq \varnothing, r(v,i) \neq \varnothing} r(u,i) r(v,i)}{\sqrt{\sum_{i: r(u,i) \neq \varnothing} r(u,i)^2 \sum_{i: r(v,i) \neq \varnothing} r(v,i)^2}} \epsilon [0,1] \tag{2.6}$$

- **Correlación de Pearson**: En Pearson la puntuación promedio, $\bar{r}_v$, $\bar{r}_u$, se toma sobre todos los ratings del usua-

rio. Con Pearson se pueden obtener ratings negativos, aunque es improbable salvo en ejemplos pequeños.

$$sim(u,v) = \frac{\sum_{v \epsilon i:r(u,i)\neq\varnothing, r(v,i)\neq\varnothing}(r(u,i)-\bar{r}_u)(r(v,i)-\bar{r}_v)}{\sqrt{\sum_{i:r(u,i)\neq\varnothing, r(v,i)\neq\varnothing}(r(u,i)-\bar{r}_u)^2 \sum_{i:r(u,i)\neq\varnothing, r(v,i)\neq\varnothing}(r(v,i)-\bar{r}_v)^2}} \epsilon [0,1] \tag{2.7}$$

Para nuestra implementación se utilizará la similitud Pearson ya que nos permite normalizar los datos. Debido a que no todos los usuarios puntúan los ítems con el mismo criterio, ya que algunos pueden tender a poner notas más altas fácilmente, se normalizan la media de los ratings del usuario para obtener la correlación Pearson de cada uno de los usuarios y hacer una valoración más equitativa.

2.3.2. Métricas de evaluación

Para comprobar cómo de preciso es el sistema de recomendación se utilizan diferentes métricas. La metodología para realizar la evaluación, como en clasificación, consiste en separar ratings en entrenamiento y test (por ejemplo, 80 % entrenamiento y 20 % test), prediciendo los ratings de test usando el entrenamiento y comprobando el nivel de acierto.

Las métricas más comunes son [26]:

- **MAE**: Calcula la distancia entre el valor de rating real y predicho con valor absoluto:

$$MAE = \frac{1}{|test|} \sum_{(u,i)\epsilon test} |\hat{r}(u,i) - r(u,i)| \tag{2.8}$$

- **RMSE**: Calcula la distancia entre el valor de rating real y predicho al cuadrado:

$$RMSE = \sqrt{\frac{1}{|test|} \sum_{(u,i)\epsilon test} (\hat{r}(u,i) - r(u,i))^2} \tag{2.9}$$

- **Precisión**: Representan los elementos relevantes devueltos, verdaderos positivos (TP) y el número total de elementos devueltos, siendo estos la suma de los verdaderos positivos (TP) y falsos positivos (FP):

$$P = \frac{TP}{TP + FP} \tag{2.10}$$

- **Recall**: Representa los elementos relevantes devueltos, verdaderos positivos (TP) entre todos los elementos buscados por el usuario, compuesto por los devueltos, verdaderos positivos (TP) y los no devueltos, falsos negativos (FN):

$$R = \frac{TP}{TP + FN} \tag{2.11}$$

Nótese que el cálculo de métricas basadas en ranking (Precisión, Recall) es muy sensible a la lista de ítems que se llega a predecir, como se ha evaluado en estudios previos [27]. Por ello, la forma más justa de hacerlo es generando un ranking donde se estima el rating de todos los ítems del sistema (excepto aquellos ya puntuados por el usuario en entrenamiento), a diferencia de las métricas de error (MAE, RMSE) donde las únicas predicciones que se realizan se hacen sobre el test conocido.

2.4. Librerías externas

En el proyecto hemos utilizado diferentes librerías externas para buscar la eficiencia y centrarse en el desarrollo importante y novedoso de este trabajo. Como consecuencia de la búsqueda de librerías externas, las que finalmente hemos utilizado han sido clasificadas en los siguientes módulos:

2.4.1. General

Setuptools [28]

Librería utilizada para facilitar el uso de directorios en proyectos Python, permitiéndonos agrupar en módulos funcionalidades independientes pero que podrán ser llamadas entre sí. Se escogió esta librería ya que se buscaba en Python conseguir una estructura bien organizada de clases y módulos independientes entre sí, pero que pudiesen seguir comunicándose entre sí, y con directorios superiores. Esta librería contribuye a facilitar esta tarea.

```
1 from setuptools import setup, find_packages
2
3 setup(name='src', version='1.0', packages=find_packages(), install_requires=['surprise', '
    pandas', 'numpy', 'click', 'pyproj', 'matplotlib'])
```

2.4.2. Procesado de datos

Pandas [29]

Para el tratamiento de los elementos en los diferentes datasets hemos utilizado Pandas. Esta librería nos permite realizar de manera muy intuitiva todas las operaciones en un dataset, como si estuviésemos realizando consultas en una base de datos, así como volcar y leer de ficheros de forma muy simple.

2.4.3. Recomendación

Surprise

Entre las múltiples librerías de recomendación, escogimos Surprise para nuestro sistema ya que estaba implementada en Python y ofrecía una documentación bastante clara en `https://surprise.readthedocs.io/en/stable/`. Dicha librería la hemos utilizado como punto de partida puesto que gran parte de los métodos han tenido que ser modificados para tomar por vecinos un fichero de entrada. Esta nos aporta gran rapidez para extraer métricas como RMSE, Precisión o Recall.

LibRec [30]

Entre las múltiples librerías de recomendación en Java destacamos esta por ser una de las más relevantes en ese lenguaje. Pese a esto, escogimos *Surprise* por ofrecer la implementación en Python y pretendimos mantener el mismo lenguaje en todo el desarrollo para hacerlo más homogéneo.

2.4.4. Visualización de trayectorias

Pygmaps

Esta librería permite generar mapas interactivos en Python y exportarlos a HTML a partir de los resultados de nuestros experimentos, pudiendo mostrarlos en el navegador. Su metodología está basada en interaccionar con Google Maps mediante Google Cloud.

Flask

Este Framework escrito en Python permite implementar la parte backend de una aplicación web. Lo utilizamos para crear la aplicación en la que visualizaremos los resultados de los diferentes experimentos. Este nos permitirá integrar los diferentes algoritmos con una visualización fácil para el usuario.

Ruby on Rails

Se planteó la posibilidad de realizar la implementación de la interfaz en lenguajes más comunes para desplegar aplicaciones web como Ruby on Rails, pero de nuevo se decidió optar por Flask, pues en primer lugar fue la tecnología enseñada durante la carrera y al mismo tiempo nos permitiría mantener el lenguaje Python a lo largo de todo el proyecto.

Bootstrap

Esta es la librería más común en el desarrollo web por la gran cantidad y constantes actualizaciones para agregar una gran cantidad de elementos ya diseñados facilitando la implementación de la página, al mismo tiempo que mantiene un atractivo visual. Dicha librería se utilizó para mejorar el CSS de nuestra página.

Heroku, Kubernetes y Docker

Heroku es una de las plataformas más comunes para desplegar aplicaciones web y hacerlas públicas. A pesar de que la velocidad de descarga se veía reducida al utilizar un plan gratuito, para nuestros requisitos fue suficiente. Se planteó utilizar Kubernetes y Docker, pero tras barajar diferentes opciones, se optó por utilizar únicamente Heroku puesto que la complejidad del proyecto reside en los patrones y debíamos perfeccionarlos con mayor prioridad.

En definitiva, de todas las librerías mencionadas para visualización de las trayectorias, finalmente escogimos Heroku, Bootstrap y Flask ya que permitían implementar la interfaz web con rapidez; y Pygmaps puesto que ofrecía la representación de mapas de una forma muy sencilla.

3 Diseño e Implementación

En este capítulo se abordará el diseño del proyecto, explicando la estructura general de este, el ciclo de vida utilizado, y sus requisitos. Posteriormente se detallará la implementación mostrando el flujo de la aplicación, y la implementación de todos los módulos utilizados.

3.1. Diseño

En esta sección se detallarán los módulos en los que se ha organizado el proyecto, las librerías externas y el proceso realizado para el procesado de datos, análisis de patrones, su posterior aplicación a la recomendación, evaluación y visualización de estos.

3.1.1. Estructura general

En la Figura 3.1 se puede ver el esquema del proyecto.

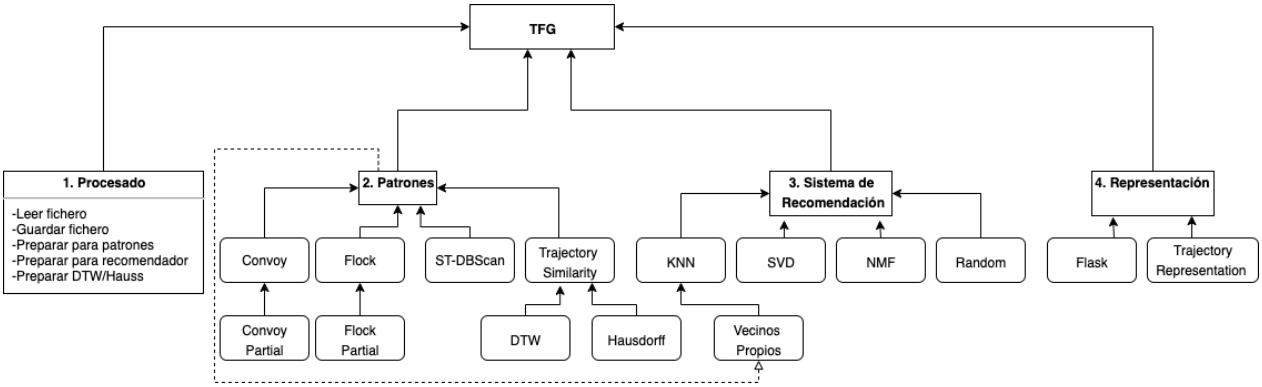


Figura 3.1: Diagrama de los diferentes módulos en la aplicación, y su relación entre sí y con el proyecto principal.

El sistema se compone de 4 módulos escritos en Python claramente diferenciados con el fin de en un primer lugar poder aplicar mejoras y añadir elementos de manera independiente, y en segundo lugar marcar etapas en la ejecución claramente diferenciadas para retomar la ejecución en cualquier punto. Los módulos en los que se ha dividido el proyecto son:

1. Extracción y procesado de datos: Este módulo es el primer paso en el proyecto y permite convertir los datos a partir de los datasets en el formato requerido para la entrada de los módulos 2

y 3, es decir, procesado de patrones y recomendación respectivamente. Este permite dividir los datos en entrenamiento y test viendo los métodos del estado del arte descritos en la sección 2.2.

2. Patrones de movimiento: En este módulo se obtienen los patrones de movimiento según los algoritmos explicados en la sección 2.1.1. Esta esperará como entrada los datasets procesados por el módulo 1, y preparará un fichero de similitudes asociando vecinos entre usuarios para el módulo de recomendación. Gracias a este módulo, los diferentes algoritmos permiten un mismo fichero de entrada, y darán un fichero de salida común, con el fin de poder cohesionar correctamente todos los módulos entre sí.

3. Recomendación: Este módulo conectará con la salida del módulo anterior, patrones de movimiento, mediante el fichero de similitudes. Este nos proporcionará un fichero asociando para cada par de usuarios una similitud, y a partir de esta se ejecutará la recomendación mediante las adaptaciones hechas a partir de la librería Surprise. Para realizar comparaciones y ver la validez de las predicciones, este módulo es capaz de generar recomendaciones sin similitudes y generar las métricas de los sistemas de recomendación.

4. Representación y Evaluación: Módulo utilizado para la visualización de mapas mediante los datos de los sistemas de procesado de datos y patrones de movimiento.

Diagrama de secuencia

Los módulos anteriormente mencionados ejecutan las llamadas a esta aplicación como vemos en la Figura 3.2. Podemos llamarlos de manera secuencial con llamadas individuales a cada uno de los módulos, o mediante un script que lo ejecutará de forma automática.

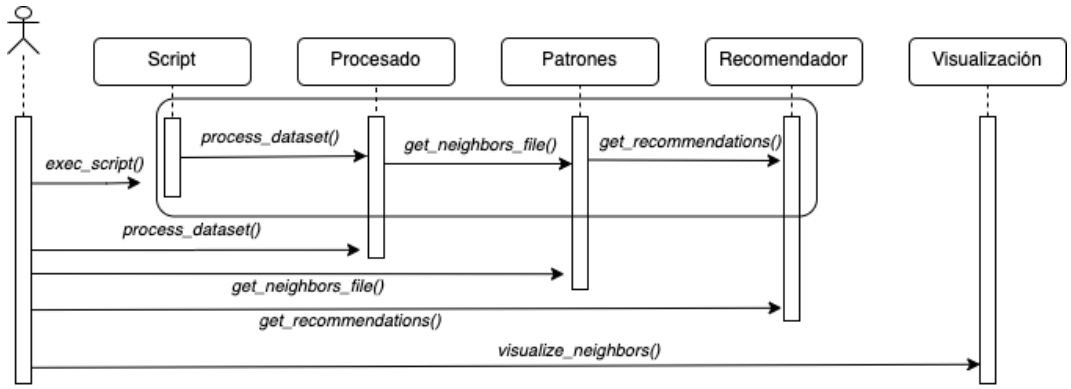


Figura 3.2: Diagrama de secuencia de la aplicación donde se muestra cada uno de los módulos y su interacción con el usuario. También se aprecia cómo sería la ejecución del script para hacerlo automático.

Caracterización a aplicaciones de Turismo

El principal foco de este proyecto es el estudio de trayectorias de usuarios para aplicarlos al dominio de turismo. De este modo, se adaptarán los métodos de co-movimiento, donde se utilizan coordenadas geográficas de GPS para utilizar los check-in de usuarios extraídos de aplicaciones como FourSquare, y se aplicarán todas las técnicas hasta ahora descritas a este nuevo escenario, tomando determinadas decisiones en consecuencia.

3.1.2. Ciclo de Vida

Para el desarrollo de la aplicación hemos seguido el Ciclo de Vida Iterativo, siguiendo las fases mostradas en la Figura 3.3. Se ha escogido este ya que al ser el trabajo extenso y contar con gran cantidad de módulos, cada uno de ellos con diferentes implementaciones, se escogió un diseño en iteraciones para poder solventar cualquier complicación y tener un producto entregable en cada una de ellas. Gracias a este, en cada una de las iteraciones, tuvimos un proyecto entregable con todos los módulos, donde pudimos comprobar que los patrones de movimiento se generaban bien, y al mismo tiempo las pruebas procesaban correctamente estos.

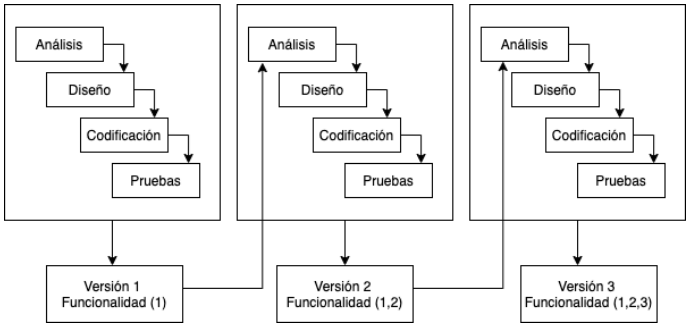


Figura 3.3: Diagrama del Ciclo de Vida iterativo mostrando las 3 fases en las que se ha llevado a cabo la implementación. En cada una de ellas se ve la funcionalidad añadiendo los resultados de las etapas anteriores.

3.2. Requisitos

En esta sección se muestran los diferentes requisitos funcionales y no funcionales de este proyecto:

3.2.1. Requisitos Funcionales

General:

RF-1.- El lenguaje debe ser Python.

RF-2.- Todo el almacenamiento debe ser mediante ficheros, no mediante RAM para permitir retomar la ejecución desde cualquier módulo del sistema.

RF-3.- El sistema debe notificar de cualquier error y exponer la causa, así como informar del estado en el que se encuentra el algoritmo.

Procesado

RF-4.- El sistema debe permitir diferentes datasets de entrada y unificarlos en caso de ser necesario.

Patrones de movimiento

RF-5.- El sistema debe permitir elegir qué patrón utilizar.

RF-6.- El sistema debe permitir modificar todos los parámetros que acepta el patrón de movimiento.

RF-7.- Deben haber múltiples algoritmos en el sistema.

Recomendación

RF-8.-El sistema debe permitir realizar recomendaciones dado un ID de usuario.

RF.9.- El sistema debe devolver métricas para conocer la exactitud de las precisiones.

RF.10.- El sistema debe permitir utilizar diferentes sistemas de recomendación para sus métricas.

Visualización

RF-11.- El sistema debe contener una representación gráfica de las trayectorias en el mapa.

3.2.2. Requisitos no Funcionales

RNF-1.- La representación del mapa debe estar visible desde cualquier dispositivo sin necesidad de arrancar el proyecto de forma local.

RNF.2.- La aplicación debe ser fácilmente accesible desde Git.

RNF.3.- Se debe adjuntar un manual con los diferentes comandos para realizar las ejecuciones.

RNF.4.- La visualización debe mostrar una interfaz fácil de utilizar y estable, pudiendo alternar diferentes comparaciones.

RNF.5.- El tiempo de ejecución de un patrón no debe durar más de 12h.

RNF.6.- La memoria total utilizada debe ser inferior al 50 % disponible del dispositivo.

3.3. Implementación

Flujo de la aplicación

En la Figura 3.4 se puede ver el flujo de ejecución del proyecto. Esta se compone principalmente por los módulos *Procesado y Patrones* que se encargarán de generar los ficheros, que utilizarán tanto el recomendador como la representación de las trayectorias.

En el módulo de Patrones, el fichero *pattern_name.py* se refiere a escribir el nombre del patrón deseado para ejecutarlo: *ConvoyTrajectory.py, fpFlockOnline, main_stdbscan.py, trajectory_similarity.py*, o *greedy_approach.py*.

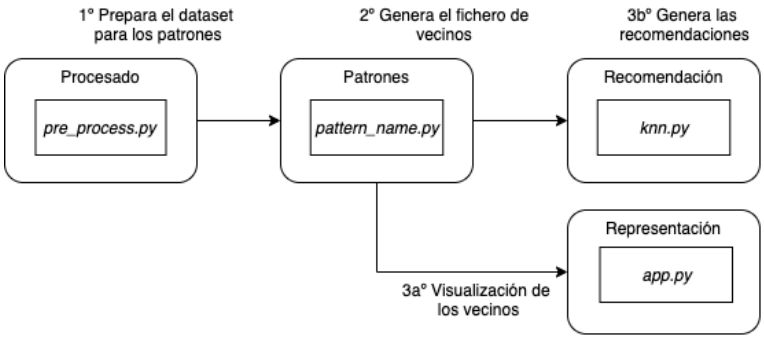


Figura 3.4: Flujo de la ejecución de los diferentes módulos en la aplicación separado en las distintas fases para lograr la ejecución y representación.

Al estar los módulos totalmente independientes, se puede recuperar el flujo en cualquier parte del

programa, pudiendo incluso en un futuro adaptarlo y mezclar distintos lenguajes.

Para ejecutarlo, un flujo a modo orientativo (se pueden elegir distintos algoritmos) sería:

```
1. python3 src/Processing/pre_process.py --input_file entradas/Rome10K/romeTempTrain.txt --
    coords_file entradas/POIS_rome__Coords.txt --output_file romePOISCompleto.txt
2. python3 src/Patterns/Convoy/ConvoyTrajectory.py --filename romePOISCompleto.txt --output
    similarity_output_convoy.txt --minpoints 3 --lifetime 2 --distance_max 0.1 --partials
    False
3. python3 src/Recommender/knn.py --train_file entradas/Rome10K/romeTempTrain.txt --
    test_file entradas/Rome10K/romeTest.txt --k 1 --neighbors_classified
    similarity_output_convoy.txt --output_file salida_knn_custom_rome.txt
4. python3 app.py
```

O para ejecutar todos los métodos obteniendo las distintas métricas, se ha preparado el script:

```
./experiments_tfg.sh
```

A continuación se explicarán las diferentes decisiones tomadas para implementar el problema.

3.3.1. Preparación del Dataset

En primer lugar se parte de dos datasets de cada una de las ciudades: Tokyo, New York y Rome, proporcionados por FourSquare y formados por checkpoints de diferentes lugares de interés. El primer dataset contiene los puntos de interés que ha visitado cada usuario acompañado del timestamp, y un segundo dataset contiene las coordenadas latitud y longitud de cada punto de interés. A partir de aquí realizamos un primer filtrado agrupando las trayectorias de cada usuario en intervalos de 8 horas para hacer el dataset más liviano, y a continuación mezclamos ambos datasets. Este subsistema se ha basado principalmente en la librería *pandas* por todas las facilidades que ofrece.

Para preparar los datos para entrenamiento y test se hicieron particiones en el tiempo, es decir, al dividir los datos que estén en test, estos deberán ser posteriores a los datos de entrenamiento para poder predecir comportamientos lógicos, no dejando trayectorias incompletas. Los datos de entrenamiento permitirán realizar un agrupamiento de los usuarios que se desplazan juntos, para poder utilizar esta información en la recomendación, y posteriormente evaluar los datos de test, acompañado de unas comparativas de los resultados obtenidos con los diferentes patrones.

Las trayectorias pueden simplificarse eliminando puntos redundantes, como vemos en la Figura 2.6. Para ello comprobamos si algunos puntos se desvían más de cierto umbral, *Acimut*, a partir del cual podemos sacarlo como un punto distinto, mientras que si sigue la trayectoria podemos simplificar todos los puntos entre el origen y el destino. El término **Acimut** describe el ángulo creado por dos líneas, una que une su posición actual y el Polo Norte, y la que une su posición actual y la ubicación distante. Esta alternativa ofrece tiempos de ejecución mucho mejores que Flock.

En el notebook aportado en el proyecto presentado en el apartado 2.1.2, se puede ver cómo agrupar las trayectorias, pero tras haber realizado diversas pruebas en los datasets anteriormente presentados se comprobó que esto no aportaba suficiente mejoría. Al fragmentar cada 8h las trayectorias, ya quedan los datos suficientemente dispersos, no ayudando a la hora de agrupar trayectorias similares, como vemos en la Figura 3.5 donde se puede apreciar que la reducción de trayectorias es mínima.

En una primera etapa se seleccionaron los atributos importantes de los dataset proporcionados, unificándolos en uno solo para poder tener la información de los usuarios y las coordenadas de los

puntos de interés unidos. Como resultado se obtienen las siguientes columnas:

```
   user_id  item_id  latitude   longitude   timestamp
15   14627  40.757564 -73.989238 1354881399
```

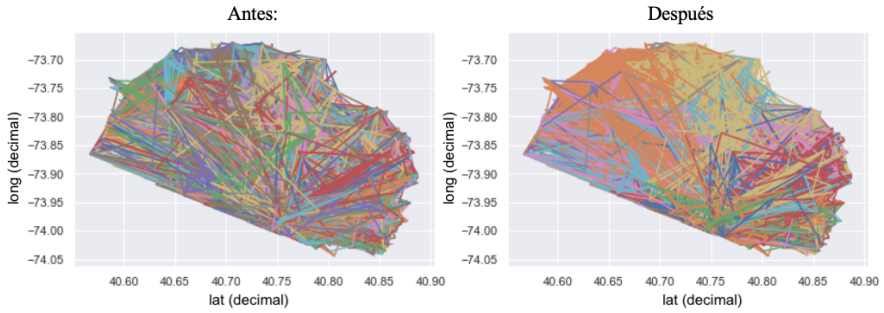


Figura 3.5: Segmentación de las trayectorias en nuestro dataset para simplificarlas sobre el dataset de New York.

Debido a que los datasets eran muy grandes, se utilizaron diferentes optimizaciones para tratar de reducir al máximo el tamaño, convirtiendo los dígitos a 32 bits y volcándolos a ficheros tras cada ejecución para evitar ocupar al máximo la memoria RAM.

Aquí se pueden diferenciar 2 cauces distintos debido al formato de los datos que esperan los algoritmos:

- Si se va a utilizar **Flock, Convoy, ST-DBSCAN** se guardan los datos en un fichero .dat tal cual los hemos agrupado.
- Si se va a utilizar **DTW y Hausdorff** debemos aplicar una conversión de formato para pasar de check-ins de usuarios a trayectorias, es decir, llegar a formato diccionario, agrupando los puntos de interés por intervalos de 8 horas. De este modo, un usuario tendrá un listado de todos los puntos que ha visto agrupados en arrays de 8 horas, con el formato:

```
{"user_id": [{"0": [[lat elements], [long_elements]], "1": ...}]}

{"15": [{"0": [[40.77383804321289, 40.757564544677734, 40.75837326049805,
40.76288986206055], [-73.87122344970703, -73.9892349243164, -73.98849487304688,
-73.97402954101562]], "1": [[40.753875732421875], [-73.98442840576172]],
"2": [[40.76355743408203], [-73.97286987304688]]}]}
```

Para ejecutar el pre-cargado del dataset se debe ejecutar el comando:

```
python3 src/Processing/pre_process.py --method dataset_similarity --input_file entradas/
    DatosNewYork/US_NewYorkTempTrain.txt --coords_file entradas/DatosNewYork/
    POIS_Coords_Foursquare.txt --output_file dataset_processed.txt
```

Para preparar el fichero en caso de estar ejecutando DTW/Hausdorff, puesto que estos necesitan una estructura de datos diferente para aplicar las sucesivas operaciones matemáticas, se debe ejecutar previamente al comando anterior:

```
python3 src/Processing/pre_process.py --input_file entradas/romeTempTrain.txt --coords_file
    entradas/POIS_rome__Coords.txt --output_file dataset_processed.txt
```

3.3.2. Ejecución de patrones de movimiento

Se escogió utilizar Python para nuestra implementación a fin de hacerla más homogénea. Por ello, estableciendo este criterio se escogieron las implementaciones de Flock y Convoy que estaban disponibles en Python.

Este módulo enlaza directamente con el contenido generado de la anterior. Aquí disponemos de los diferentes algoritmos para detectar patrones definidos en la Figura 3.1. Una de las prioridades al dividir el proyecto en diferentes módulos y tener gran cantidad de algoritmos distintos fue homogeneizar las salidas, de modo que independientemente de qué algoritmo se utilice, se podrán llamar a los mismos métodos. Así, todos estos algoritmos guardarán como respuesta un fichero de similitud con el siguiente formato:

```
user1_id user2_id occurrences
2        21        1.0
21       39        2.0
```

Donde *occurrences* es el número de veces que ambos han visitado un lugar en común. A continuación se explicará cómo ejecutar cada uno de ellos:

Flock

Flock se desarrolla en 2 fases:

1. Determinar aquellos objetos móviles que se encuentran relativamente cerca según el parámetro ε.
2. Combinación y agrupación de patrones que ocurran durante el parámetro δ.

Se comienza descubriendo el conjunto de discos máximos en el primer instante de tiempo, y posteriormente en el siguiente instante. Cuando se han analizado dos instantes consecutivos se determina los objetos que han permanecido juntos. Una vez establecidas las combinaciones, se crea un conjunto de discos candidatos que contabilizan cuántos instantes de tiempo los objetos cubiertos por los discos han permanecido juntos. A partir de aquí, el proceso se repite con el conjunto de discos candidatos y el conjunto de discos máximos que se encontrará el siguiente instante de tiempo disponible. Después de cada combinación, se analiza el conjunto de discos candidatos y se devuelven aquellos que han superado el mínimo número de intervalos de tiempo establecido por el parámetro δ. Para ejecutar Flock hacemos uso de una librería LCM que nos asegurará encontrar la trayectoria común más larga.

La comunicación entre programas se realiza mediante ficheros, utilizando ficheros *.dat* para dar la entrada a LCM, y ficheros *.mfi* para procesar la salida de la librería LCM.

Los discos que se detectan acorde a lo explicado, siguen el siguiente formato:

```
KeyFlock: 30 Begin: 238 End 239 [0, 1025, 2437, 3720]
KeyFlock: 32 Begin: 231 End 239 [0, 1025, 2437]
KeyFlock: 34 Begin: 238 End 240 [0, 1025, 2437, 3720]
```

Durante la implementación se tuvo que modificar el dataset para adaptarlo al formato de la librería, ya que los patrones Flock los detecta mediante timestamp, pero estos son ordenados desde 0 hasta N, agrupados por id, es decir, para un mismo id (1024) el timestamp debería ir en orden ascendente. El dataset lo preparamos en la función *preprocessDataset* (que pertenece al módulo previamente explicado en la Sección 3.3.1).

```
id        item_id     latitude      longitude      timestamp
796       140514      40.758328     -73.985457     0
1024      788744      40.730084     -73.989256     0
1024      788734      40.730085     -73.989257     1
1024      788784      40.730086     -73.989258     2
```

Como se comentó anteriormente en la sección 2.1.1, Flock necesita una serie de argumentos para determinar los elementos que viajan juntos. Se puede ejecutar mediante el comando:

```
python3 src/Patterns/Flock/fpFlockOnline.py --filename entradas/Datasets/
    US_NewYork_POIS_Coords.txt --output flock_output.txt --epsilon 0.2 --mu 2 --delta 0.2
```

Convoy

Convoy necesita una serie de argumentos, como se explicó en la sección 2.1.1, para determinar los elementos que viajan juntos. Este nos dará más flexibilidad a la hora de buscar los elementos que visitan los mismos lugares. El comando para ejecutarlo es:

```
python3 src/Patterns/Convoy/ConvoyTrajectory.py --filename entradas/Datasets/
    US_NewYork_POIS_Coords_short.txt --output similarity_output_convoy.txt --minpoints 3 --
    lifetime 2 --distance_max 0.1 --partials False
```

En este comando se puede ver la opción *partials* la cual nos mostrará los elementos que cumplen los criterios de Convoy, y todos sus sub-conjuntos. Si esta está activada, devolverá un listado muy completo pero requiere mucho tiempo para ejecutarse. Si se quiere utilizar partials, recomendamos utilizar la versión Convoy Partials. En caso contrario mantener desactivado para experimentar un rendimiento óptimo.

Consideraciones Adicionales

Realizando pruebas extensas con datasets muy grandes, se apreció la principal flaqueza de Flock, y de Convoy con la opción *partials* activada. Al buscar trayectorias comunes entre todos los usuarios, mantiene durante toda la ejecución la información completa en RAM, al mismo tiempo que su coste computacional crece de manera exponencial. Tras diversos cálculos se comprobó que podría llegar a tardar hasta 1 año una ejecución de apenas 1 millón de entradas en el dataset. Para solucionar esto, se llegó a una idea muy intuitiva: hasta ahora estábamos cargando todo el dataset para ejecutar el Flock, pero en lugar de esto se podría hacer un pre-procesado adicional, para únicamente cargar los vecinos de cada usuario, es decir, hacer múltiples ejecuciones de Flock (tantas como usuarios distintos haya) en paralelo, de modo que en lugar de ejecutar Flock sobre 1 millón de entradas, se ejecute Flock 500 veces sobre datasets de apenas 2.000 entradas. Esto permitió ejecutar Flock y Convoy sobre un dataset muy grande. Para ello hicimos las 2 siguientes implementaciones adicionales:

Flock y Convoy Partials

Para ejecutar estos algoritmos de manera más eficiente se prepararon los siguientes pasos:

1.– Calcular la similitud de todos los usuarios empleando un método rápido como *DTW* o *ST-DBSCAN*.

2.– Sobre los resultados obtenidos, coger los k usuarios más similares a nuestro usuario, y ejecutar Flock/Convoy iterativamente con esta lista reducida.

Los comandos para ejecutarlos son:

Flock:

```
python3 src/Patterns/Flock/flock_partial.py --dataset entradas/Datasets/
    US_NewYork_POIS_Coords_short_10k.txt --similarity_file
    US_NewYorkTempTrain_1k_greedy_similarity.txt --output prueba_output.txt --k 100 --
    epsilon 0.02 --mu 2 --delta 0.002
```

Convoy:

```
python3 src/Patterns/Flock/convoy_partial.py --dataset entradas/Datasets/
    US_NewYork_POIS_Coords_short_10k.txt --similarity_file
    US_NewYorkTempTrain_1k_greedy_similarity.txt --output prueba_output.txt --k 100 --
    minpoints 2 --lifetime 2 --distance_max 0.2 --partials False
```

DTW y Hausdorff

Debido a la visible falta de rendimiento principalmente de Flock, evaluar los patrones mediante el desvío en la trayectoria es una opción muy viable computacionalmente. En esta metodología, gracias a haber pre-procesado correctamente los datos y tenerlo en formato de diccionario, tan solo se necesitará completar una búsqueda de cada elemento contra todos, pudiendo paralelizarlo muy fácilmente.

Para esta aproximación, es necesario dividir el dataset de los check-in de los usuarios en trayectorias, agrupándolos de tal modo que si dos interacciones son muy distantes en el tiempo (superan un umbral determinado, por ejemplo 8 horas) se consideran dos trayectorias distintas, como se ha hecho en trabajos previos [31].

Definiendo un usuario u como un conjunto de trayectorias $(x_1^u, ..., x_n^u)$, calcularemos la similitud con otro usuario mediante la media de las similitudes en las trayectorias de ambos usuarios:

$$sim(u, v) = \frac{1}{n \cdot m} \sum_{j=1}^{n} \sum_{k=1}^{m} tsim(x_j^u, x_k^v) \tag{3.1}$$

Donde n y m corresponden al número de trayectorias de los usuarios u y v respectivamente, y $tsim(.,.)$ es cualquier función de similitud de trayectorias (DTW, Hausdorff en nuestras pruebas).

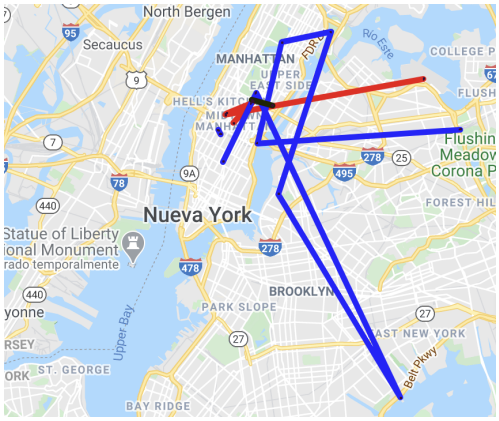


Figura 3.6: Registros recopilados de dos usuarios (uno en azul y el otro en rojo) en la ciudad de New York. Tenga en cuenta que hay muchos más lugares en común en el centro de Manhattan.

Como se muestra en la Figura 3.6 con este tipo de similitud, algunas partes de la trayectoria de cada

usuario son bastante cercanas entre sí y, por lo tanto, similares, mientras que otras partes están muy distantes entre sí. Concluimos así que esta similitud propuesta es más flexible tanto en las dimensiones temporales como espaciales, ya que todos los puntos visitados dentro de un marco de tiempo (por ejemplo 8 horas) se considerarán dentro de la trayectoria.

Previo a aplicar el algoritmo se hicieron diversas pruebas para ver si se podía segmentar la trayectoria, pero resultó imposible ya que al ser checkpoints esparcidos en el tiempo, eliminar más puntos por usuario supondría perder mucha información. Se hicieron pruebas en un dataset de 800.000 elementos, reduciendo apenas 5.000. El pseudocódigo del algoritmo utilizado es:

```
    D = [elementos_traj, elementos_traj]
    for user in trajectorias_usuarios
        calcular similitud DTW/Hausdorff
    Almacenar los k mas similares
```

De este modo, se consiguió tener por una parte en cuenta la similitud entre las trayectorias de los usuarios, y al mismo tiempo ponderar los usuarios que hayan compartido más trayectorias similares en el tiempo, apareciendo en los k mejores.

Para ejecutarlo utilizamos el comando:

```
python3 src/Patterns/TrajectorySimilarity/trajectory_similarity.py --dataset dataset_file.
  txt --output_file similarity_output.txt --function "hausdorff o dtw" --k 100 --threads 1
```

Ad-hoc

Esta solución se presenta como punto base de las futuras pruebas, ya que en ella nos basamos en la premisa de que los usuarios que visiten la misma localización alrededor de la misma fecha serán más similares. Se formula como:

$$sim_\delta(u, v) = \|i \in I : |t(u, i) - t(v, i)| < \delta)\| \tag{3.2}$$

Donde I es el conjunto de todos los ítems (lugares) en el sistema, $t(u, i)$ devuelve el timestamp donde el usuario u se registró en la ubicación i (si no se registró nunca el valor será infinito), y δ es el parámetro para controlar la ventana temporal que se permite para considerar si dos usuarios tienen un lugar en común.

3.3.3. Recomendación

Para la recomendación se ha utilizado la librería *Surprise*. La entrada a este módulo es el fichero de recomendación que sale de los diferentes algoritmos de patrones de movimiento y utilizaremos una recomendación basada en usuario, con los datos $user1\_id, user2\_id, rating$. De este modo, para los usuarios más similares calcularemos una puntuación en base a los ratings que proporcionó este usuario y la similitud entre ellos, devolviendo un fichero de recomendación en el cual se mostrará para cada usuario e ítem de test la puntuación de este:

```
user_id     item_id     rating
   8           7        1.0964
   8           64       1.2289
   110         7        1.2235
```

Se escogió *Surprise* por la facilidad que ofrece para realizar la recomendación, pudiendo variar con tan solo este parámetro qué tipo de similitud se quiere:

```
algo = KNNCustom(k=k, sim_options={'name': 'pearson_baseline', 'user_based': True})
```

Recomendador KNN utilizando la salida de vecinos similares

En nuestro propio KNN, se introdujo la similitud para el cálculo de vecinos a partir del fichero extraído de los módulos de búsqueda de patrones de movimiento. De este modo, cuando KNN inicia la búsqueda de qué vecinos recomienda, buscará aquellos que se han introducido en el fichero para crear la matriz de recomendación, y así calculará el valor estimado de la recomendación para estos. Este cálculo se puede visualizar en el pseudocódigo:

```
    if neighbors:
        for (neighbor_id, similarity) in neighbors:
            for (item, rating) in train[neighbor_id]:
                if item == user_id:
                    sum_ratings += rating*similarity
```

Recomendadores SVD, Random, NMF y KNN, todos sin fichero de similitud

Para las posteriores métricas del apartado de pruebas, se implementaron diversos recomendadores adicionales sin tener en cuenta el fichero de similitud, para comprobar su rendimiento y la utilidad de este. Al contar con Surprise, pudieron implementarse de forma mucho más fácil mediante el siguiente pseudocódigo:

```
        algo_svd = SVD()
        algo_random = NormalPredictor()
        algo_knn = KNNWithMeans()
        algo_nmf = NMF()

        algo_svd.fit(train)
        predictions_svd = algo_svd.test(test, verbose=False)
```

De este modo, al no necesitar estos una implementación propia pudieron instanciarse, entrenar y ejecutar las diversas métricas con apenas unos comandos.

3.3.4. Visualización

Para hacer más sencilla la visualización de los resultados, se ha implementado una aplicación web mediante *Flask* y desplegada en *Heroku* que nos permitirá subir en tiempo real los ficheros generados por los módulos previamente descritos y poder interactuar de manera dinámica con ellos. Decidimos implementarlo en Python debido a que sería el lenguaje más rápido de conectar con nuestro sistema, ayudado de *jQuery* y *CSS* para realizar las animaciones. La página se visualizará como se puede ver en la Figura 3.7.

En ella, para poder manejar el mapa interactivo, se debe adjuntar:

- El fichero del dataset procesado con las columnas (user_id, item_id, lat, long, timestamp).
- El fichero de similitudes con las columnas (user1_id, user2_id, similitud).
- Opcional: El fichero de trayectorias con el formato ('user_id': ['traj_counter': [...]]).

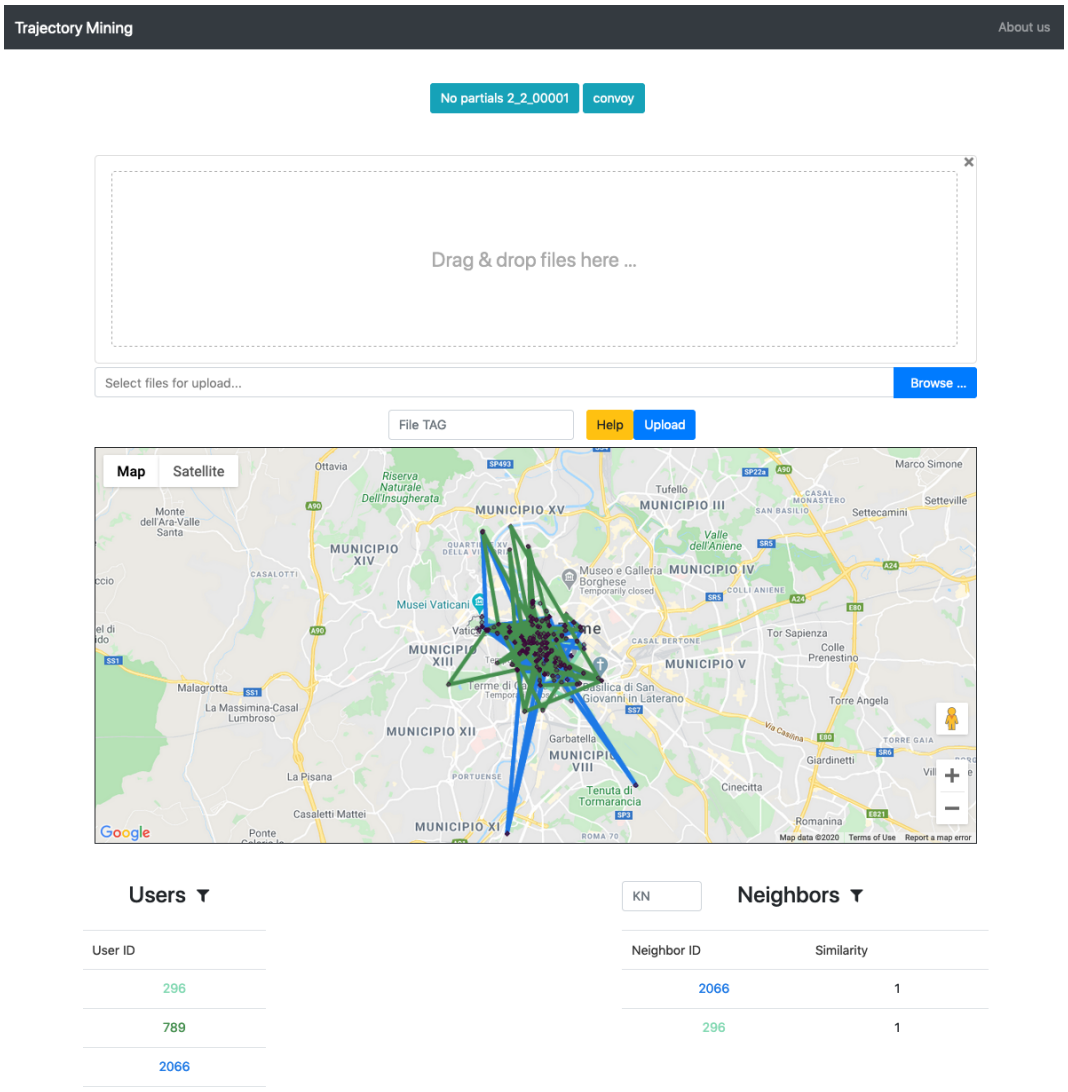


Figura 3.7: Interfaz web para visualizar los resultados en el servidor Heroku.

La aplicación se puede ejecutar desde local o desde la desplegada en *Heroku*: `https://tourismrecommender.herokuapp.com/`

Por otro lado, el comando para ejecutarla en local es:

```
python3 app.py
```

En la aplicación, como se muestra en la Figura 3.7 se puede filtrar los id de los usuarios, y según el seleccionado se mostrará un listado de vecinos. Se pueden escoger los K vecinos más próximos (*K Nearest Neighbors*) para cada usuario, y estos serán los que se representen en el mapa. Sus similitudes se calcularán según el fichero subido, y ordenándose de forma ascendente. El color de los *user_id* tanto en la tabla de usuarios como en la de vecinos se corresponde con el color de la línea en el mapa de la trayectoria. En la parte superior se muestra una sección donde se puede ir guardando de forma temporal todos los ficheros que se vayan añadiendo, y serán fácilmente identificables mediante el TAG asignado a la hora de subirlo. Gracias a estos tags se podrá retomar la visualización de cualquiera de ellos cuando se estén realizando pruebas en la aplicación, y así poder comparar resultados.

4

Pruebas y resultados

En este capítulo se detallarán los resultados obtenidos en las diferentes pruebas para analizar el desempeño de aplicar trayectorias a sistemas de recomendación. En primer lugar, se mostrarán los tiempos para cada uno de los métodos así como diferentes variaciones en el tamaño del dataset, ya que hay métodos excesivamente costosos. A continuación, se comparará el número medio de vecinos para cada método y las métricas de los sistemas de recomendación, para discutir posteriormente su utilidad.

4.1. Entorno de pruebas

Las pruebas se han llevado a cabo en un equipo con las siguientes características:

Recursos	Características
Versión Python	2.7
S.O.	macOS Mojave 10.14.6 (18G103)
CPU	2,7 GHz Intel Core i7 (I7-8559U)
GPU	Intel Iris Plus Graphics 655 1536 MB graphics
RAM	16 GB 2133 MHz LPDDR3

Tabla 4.1: Tabla de las características del entorno de pruebas.

4.2. Comparativa del rendimiento de los distintos métodos para calcular vecinos

A continuación se mostrarán las pruebas realizadas con los diferentes métodos, donde por una parte mostraremos el tiempo de cada uno de ellos, así como cuánto afecta el tamaño del dataset a cada uno de los métodos. Escogimos trabajar con el conjunto de datos [1] usado en el trabajo de [32]. De entre las ciudades disponibles en estos datos, se escogió Roma por ser la que más datos aportaba y a la vez era la más equilibrada para ejecutar las pruebas de rendimiento de los diferentes métodos.

Como se puede apreciar en la tabla 4.2, los diferentes métodos implementados muestran grandes variaciones entre sí al modificar el tamaño del dataset. Algunos como Flock, finalizan el proceso al utilizar datasets a partir de un dataset de 50K elementos. Para entender el por qué, debemos ver la

[1] Se encuentra disponible en el siguiente repositorio: https://github.com/igobrilhante/TripBuilder (accedido en Enero 2020).

Algorithm	1K	10K	20K	60K
Ad-Hoc	0m0.477s	0m1.952s	0m5.253s	0m35.758s
Sim. tray. Hausdorff	0m10.568s	22m32.164s	88m17.315s	1157m31.251s
Sim. tray. DTW	0m3.386s	6m16.519s	26m9.038s	232m25.592s
ST-DBSCAN	0m0.817s	0m3.173s	0m9.487s	1m29.065s
Convoy	0m5.786s	0m6.086s	0m6.266s	0m6.098s
Convoy partials	0m2.786s	0m3.086s	0m3.266s	0m4.098s
Flock	0m5.199s	24m23.087s	49m34.065s	†
Flock partials	0m5.500s	0m30.545s	0m14.880s	3m11.811s

Tabla 4.2: Comparativa de tiempos para los diferentes métodos y versiones del dataset, donde 'Sim. tray.' hace referencia a los métodos de similitud de trayectorias explicados en la Sección 2.1.1.

distribución del número de usuarios y los ítems (que en este caso son POIs, o puntos de interés) de cada uno de los datasets en la tabla 4.3. Recordemos que los puntos de interés son puntos de ubicación específicos que puede resultar útil o interesante para alguna persona; en nuestro trabajo representará estos lugares registrados en los datos de FourSquare.

Dataset	1K	10K	20K	60K
Número POIs	163	319	346	394
Número usuarios	129	1208	2618	7954

Tabla 4.3: Comparativa de la distribución de usuarios y POIs por dataset.

Debido a que métodos como Flock tienen unos requisitos muy estrictos para definir a dos usuarios como vecinos, que deben visitar los diferentes POIs en el mismo orden y a una distancia fijada, su coste computacional crece muy rápido conforme aumenta la distribución del número de usuarios diferentes en el dataset. Por otra parte, métodos como análisis de series temporales (DTW, Hausdorff), al tener que comprobar todos con todos, no se beneficia de que algunos usuarios no cumplan sus condiciones, y por ello presentan un coste mayor cuanto más crece el tamaño del dataset. En contraste, métodos como Convoy o ST-DBSCAN, son fácilmente escalables gracias a sus bajas condiciones para detectar vecinos.

Debido a que los diferentes métodos cuentan con una gran cantidad de parámetros, para poder ejecutar múltiples combinaciones de todos ellos y mostrar el número de vecinos de cada uno, se escogió trabajar con el dataset de 10K puesto que es el más equilibrado según nuestras pruebas en cuanto a tiempo requerido y exactitud de los resultados.

4.3. Número de vecinos promedio

En la tabla 4.4 se muestra el número promedio de vecinos encontrado por cada uno de los métodos, para cada combinación analizada. En esta se puede ver cómo afectan los diferentes argumentos en el comportamiento de cada uno de los algoritmos:

Convoy: Se observa que, dependiendo del número mínimo de puntos y el tiempo que deben viajar juntos escogidos, existe una variación en el número de vecinos encontrado. Esto no afecta en gran medida por ser relativamente flexible en el orden de visita de los puntos de interés, y estar estudiando únicamente el ámbito de una ciudad, por lo que permite cierta holgura. No ocurre lo mismo con el

Algoritmo	Parámetros	Número vecinos promedio
Ad-Hoc	δ=3600	1.3
	δ=10800	1.8
Sim. tray.	Hausdorff	1206
	DTW	1206
ST-DBSCAN	ε=5000, *th*=6000, *minEps*=1	1.7
	ε=5000, *th*=6000, *minEps*=5	3.1
	ε=4000, *th*=2000, *minEps*=1	2.2
	ε=1000, *th*=2000, *minEps*=1	1.5
Convoy	*min points*=20, *lifetime*=2, *dist max*=10^{-2}	22.8
	min points=2, *lifetime*=2, *dist max*=10^{-3}	2.1
	min points=2, *lifetime*=5, *dist max*=10^{-2}	1.23
	min points=5, *lifetime*=2, *dist max*=10^{-2}	5.92
	min points=5, *lifetime*=2, *dist max*=10^{-15}	6
	min points=3, *lifetime*=5, *dist max*=10^{2}	0
Convoy partials	*min points*=2, *lifetime*=2, *dist max*=10^{-4}	1.4
	min points=2, *lifetime*=2, *dist max*=10^{-3}	1.7
Flock	ε=10, μ=2, δ=2	4
	ε=0.1, μ=2, δ=10	2
	ε=10, μ=2, δ=10^2	0
	ε=0.1, μ=2, δ=2	4
Flock partials	ε=10, μ=2, δ=2	5.2
	ε=0.1, μ=2, δ=10	5.09
	ε=0.1, μ=2, δ=2	8.04

Tabla 4.4: Comparativa de vecinos obtenidos para los diferentes métodos, donde st y th denotan spatial threshold y temporal threshold en el método ST-DBSCAN, y el resto de notación como en la Tabla 4.2.

argumento *dist max*. Como se puede ver, en Convoy esto determina el radio máximo del disco donde se deben encontrar los vecinos. Así, si se introduce *dist max*=100, lo que sucede es que todos los elementos se encuentran dentro del mismo disco, y por ello no encuentra ningún vecino (0). En cambio, si jugamos con el radio de este, podemos encontrar diferentes números medios de vecinos. En cuanto a los partials se aprecia una disminución en el número medio de vecinos, ya que al estar acotando el tamaño de búsqueda para cada uno de los usuarios, el número medio de vecinos se reduce, aunque no tanto la precisión de este.

Ad-Hoc: Se puede ver cómo afecta el δ de tiempo en este método, variando ligeramente el número medio de vecinos encontrado. Se aprecia una oscilación entre 1.3 y 1.8, siendo valores razonables, ya que a mayor ventana temporal, mayor número de vecinos encontraremos.

Análisis de series temporales (DTW, Hausdorff): Este método devuelve siempre 1206 vecinos, ya que se basa en el cálculo de la métrica de similitud para todos y cada uno de los usuarios, por lo que incluso siendo totalmente diferentes, se obtendrá un número de similitud, aunque este sea bajo.

Flock: En Flock vemos que debido a sus grandes restricciones, el número de vecinos promedio encontrados por usuario es relativamente bajo. Podemos apreciar cómo afecta la constante δ, ya que a mayor ventana de tiempo, más estricto es al necesitar que todos los puntos se mantengan juntos durante ese intervalo de tiempo. Se puede apreciar cómo afecta en partials el pre-procesado de similitud

realizado a la hora de particionar qué vecinos compararemos con cada uno de los usuarios. Esto se representa en la alta cantidad de vecinos medios por usuario, ya que gran parte de los vecinos contra los que se comparan ya han mostrado cierta similitud, siendo mucho más fácil para Flock encontrar vecinos.

ST-DBSCAN: En este método podemos ver cómo afecta el parámetro *minEps*, pues a un número menor se permite cierta soltura a la hora de escoger los vecinos, buscando el número más similar a este argumento. Por ello cuando *minEps* es 5, trata de buscar conjuntos que tengan un número de puntos lo más similar a este número. Vemos también que a un menor umbral espacial (ε) y temporal, el número de vecinos disminuye, ya que es más difícil encontrar usuarios que cumplan los criterios.

4.4. Validación del número promedio de vecinos

En la Figura 4.1, se muestran diferentes capturas de las trayectorias para la configuración más significativa de los métodos seleccionados. Es de recalcar que mostramos la trayectoria completa del usuario, aunque para el método de similitud solo haya coincidido en N puntos. Esto nos dará una visión más holística de la similitud de trayectorias entre los usuarios. Dentro de cada una de las imágenes de esta figura, a la izquierda se muestra un listado de usuarios (contienen un número mayor que el mostrado) y a la derecha un listado con los vecinos para dicho usuario (también con un número mayor del mostrado en la captura). Estos se relacionan entre sí y en la representación del mapa mediante el color del *id* de usuario.

En la Figura 4.1(a) se puede corroborar que coincide aproximadamente con el número promedio de 4 usuarios encontrado, al mismo tiempo que la Figura 4.1(b) con 22.8. Habiendo cogido la configuración mayor en la que encontramos vecinos para cada uno de los métodos, se aprecia cómo **Flock**, al ser más estricto que **Convoy**, nos muestra una cantidad menor de vecinos, pero mantienen una trayectoria similar dentro del radio seleccionado. Es de remarcar que en **Flock**, a diferencia de **Convoy**, es necesario que siga una trayectoria espacio-temporal similar.

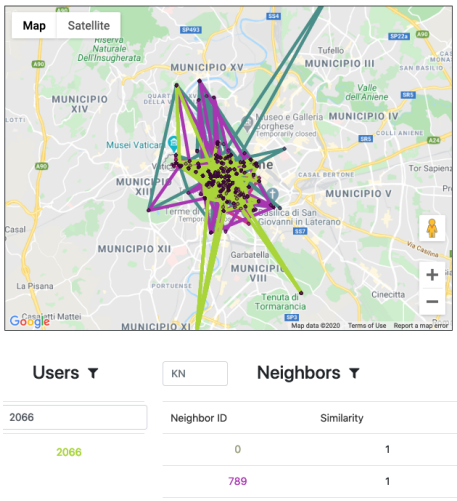
En la Figura 4.1(c) vemos cómo se comporta **Ad-Hoc** con un δ de 3600s. Se observa que la cantidad de vecinos es muy reducida, ya que 3600s representa vecinos en un intervalo de 1h, siendo una ventana temporal relativamente reducida.

El método **DTW** representado en la Figura 4.1(d) muestra todos los vecinos, 1206, ya que este no criba vecinos sino que calcula la similitud, ordenándolos cuanto mayor es esta en el listado de vecinos, pudiendo cribarlos en el recomendador.

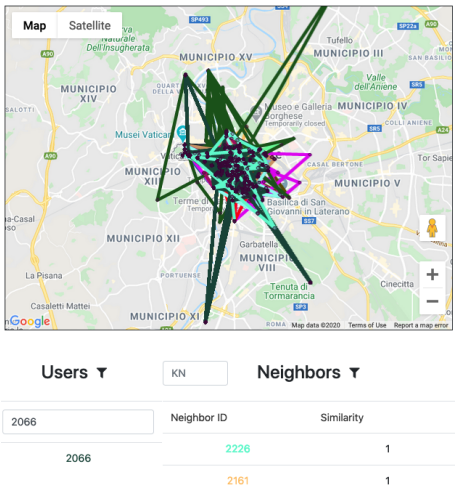
ST-DBSCAN en la Figura 4.1(e), se puede ver que es capaz de encontrar una media de 3.1 vecinos, aunque para el usuario seleccionado *2066* encuentra 20 vecinos con la configuración seleccionada. Esto es debido a que hemos permitido una ventana espacio-temporal suficientemente amplia.

4.5. Análisis del número promedio de vecinos

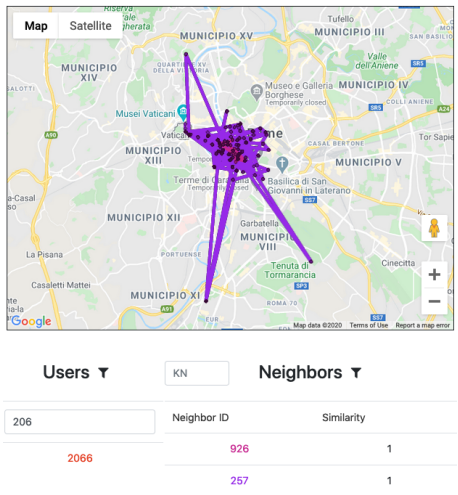
En esta sección se mostrará cómo afectan los parámetros a Flock, Convoy y ST-DBSCAN, pues han sido los que más variación han sufrido en función de sus parámetros, como se puede observar en la tabla 4.4.



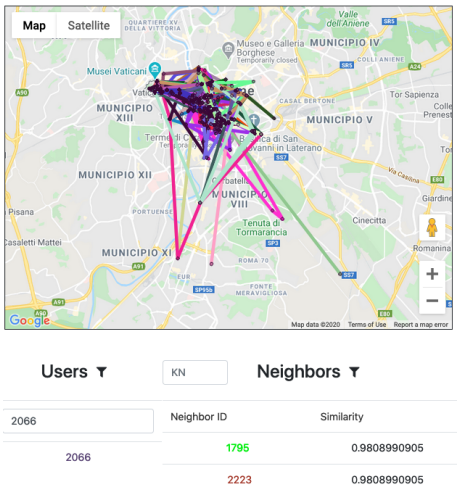
(a) Trayectoria Flock ε=10, μ=2, δ=2.



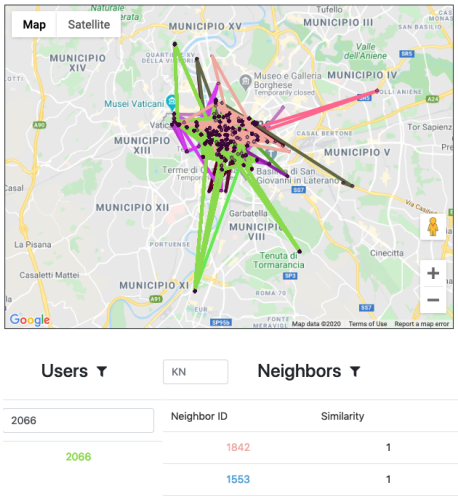
(b) Trayectoria Convoy *min points*=20, *lifetime*=2, *dist max*=0.01.



(c) Trayectoria Ad-Hoc δ=3600.



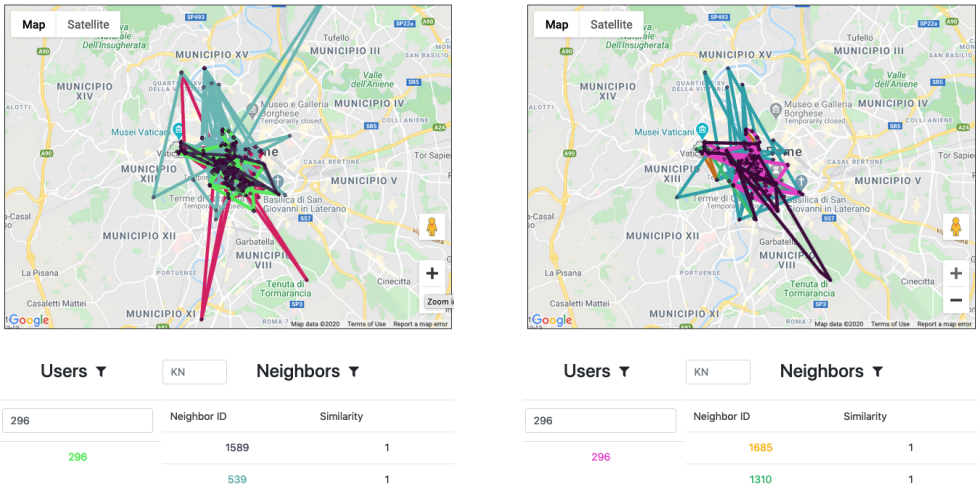
(d) Trayectoria DTW.



(e) Trayectoria ST-DBSCAN ε=5000, *th*=6000, *minEps*=5.

Figura 4.1: Visualización de las trayectorias en los diferentes métodos de similitud aplicadas a un mismo usuario, 2066.

4.5.1. Convoy



(a) Trayectoria Convoy *min points*=2, *lifetime*=2, *dist max*=0.01.

(b) Trayectoria Convoy *min points*=5, *lifetime*=2, *dist max*=0.01.

Figura 4.2: Comparación entre dos resultados de Convoy, donde se recalca la importancia de elegir los parámetros.

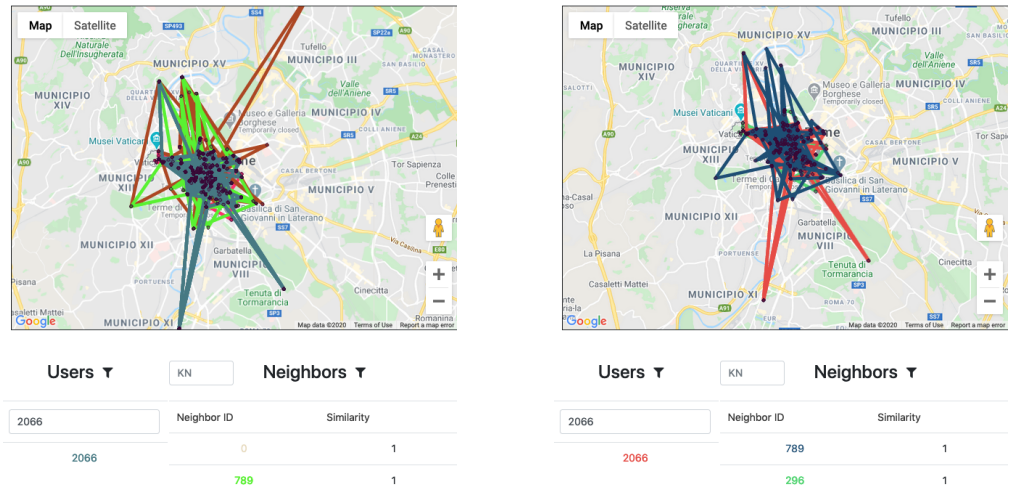
Un aspecto a destacar de Convoy es cómo trata el argumento de número mínimo de puntos. Dependiendo de este punto se afinará más o menos la búsqueda. En la Figura 4.2(a) se puede ver cómo si el número mínimo de puntos está a 2, se intenta amoldar lo máximo a este, devolviendo trayectorias de aproximadamente 2 usuarios, mientras que si se pone el numero mínimo a 5, como en la Figura 4.2(b), intenta aproximarse lo máximo. Este parámetro ofrece resultados hasta el valor en el que por más puntos similares que se le requieran, no sea capaz de encontrarlos.

4.5.2. Flock

En Flock, según se explicó en la sección 2.1.1, δ representa el intervalo de tiempo definido, tomando instantes (1, 2, 3, ...) como unidad. En la Figura 4.3, se observa que a mayor restricción en los instantes, es decir, el intervalo de tiempo, muestra menos vecinos promedio. Esto se debe a que el algoritmo Flock restringe que deben haber al menos δ instantes de tiempo de diferencia entre un punto y otro. Por ello cuanto mayor es este valor, menor número de puntos se encontrarán, como se ve en la tabla 4.2 en el caso con $\delta = 10^2$, donde no se encuentra ningún vecino.

Como podemos ver, esto se contrasta en la Figura 4.3(a) donde el número de vecinos es el doble que en la Figura 4.3(b).

Si se comparan estos mismos parámetros ejecutando Flock partials, en la Figura 4.4 se observa que se obtiene una cantidad mucho mayor de vecinos, pasando de 2 y 4, a 5.09 y 8.04 respectivamente. Esto es debido a que Flock partials se basa en ejecutar Flock para cada usuario de forma individual. Dado que Flock restringe que quiere obtener la secuencia de usuarios que hayan permanecido juntos durante más tiempo, si lo fraccionamos en pequeñas ejecuciones de Flock, obtendremos un número mayor de vecinos. De este modo, pese a obtener más vecinos, conseguimos obtener vecinos centrados a un usuario, pero no tanto a un grupo de N elementos consecutivos. Para nuestra finalidad, que es

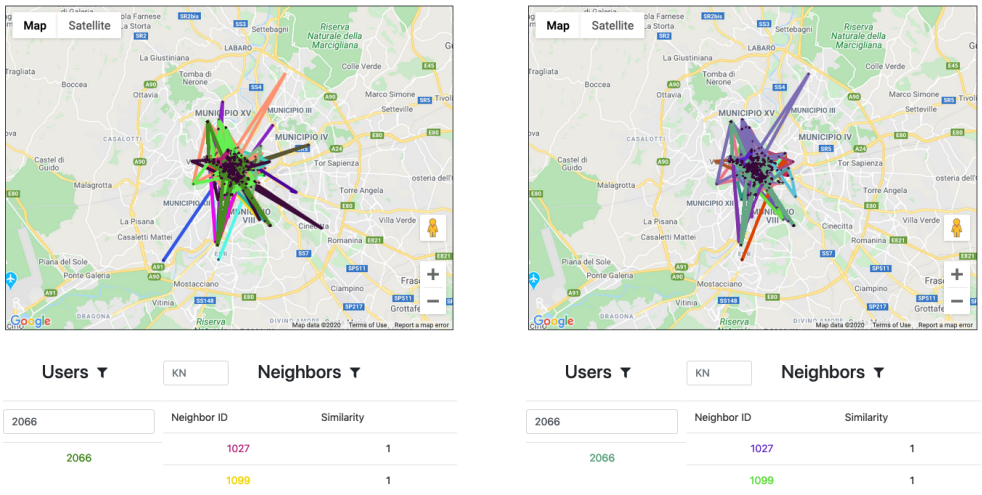


(a) Trayectoria Flock ε=0.1, μ=2, δ=2.

(b) Trayectoria Flock ε=0.1, μ=2, δ=10.

Figura 4.3: Comparación entre dos resultados de Flock, donde se recalca la importancia de elegir los parámetros.

buscar los usuarios comunes a otro, nos es más que válida esta implementación. Podemos ver cómo el número promedio de vecinos se duplica de la Figura 4.4(a) a la Figura 4.4(b).



(a) Trayectoria Flock partials ε=0.1, μ=2, δ=2.

(b) Trayectoria Flock partials ε=0.1, μ=2, δ=10.

Figura 4.4: Comparación entre dos combinaciones para Flock partials, donde se ve la importancia de elegir los parámetros.

4.6. Validación del recomendador

Se ejecutó el recomendador KNN utilizando los usuarios encontrados con los métodos de similitud para los vecinos de cada uno de ellos en KNN. Entre las métricas obtenidas, destacamos la precisión y MAE.

En **Convoy** se puede apreciar en la Figura 4.2 que la mejor puntuación la ofrece convoy partials con una configuración de *min points=2, lifetime=2, dist max=0.1*. Esto es debido a que permite que únicamente habiendo viajado juntos en 2 puntos lo considere como vecino. En *Convoy normal* se observa una correlación entre cuanto más estrictos son los parámetros, es decir, menor es la distancia acotada y mayor es el tiempo necesario de POIs en los que coinciden, ligeramente menor es la precisión.

Ad-Hoc permite una precisión bastante competitiva dada la simplicidad del algoritmo, obteniendo la mejor precisión con un δ de 3600s.

Los métodos basados en análisis de tiempo, **Hausdorff y DTW** no han destacado por sus métricas, aunque en el caso de DTW si que ha ofrecido una gran rapidez para el cálculo de los resultados.

Entre los diferentes resultados de **Flock** no se aprecian grandes diferencias, y esto se justifica a que debido a la gran cantidad de condiciones establecidas para contar a los usuarios como vecinos asegura que una vez encontrados ofrecerá métricas competitivas.

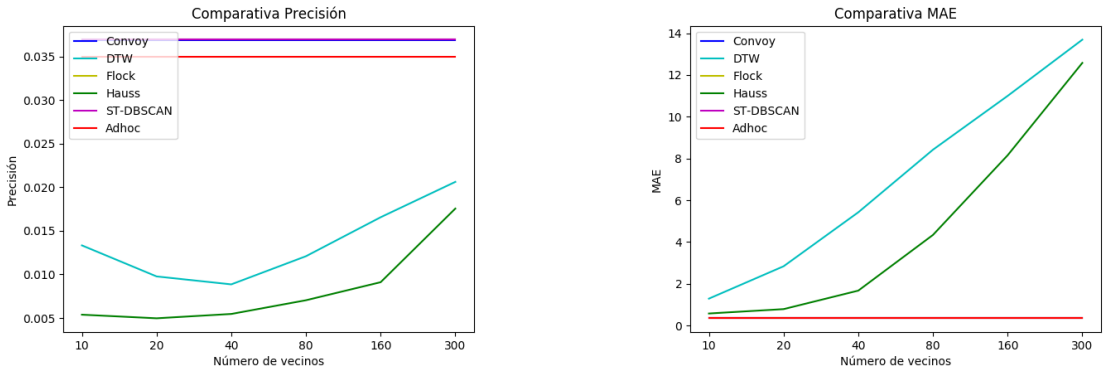
ST-DBSCAN destaca por una precisión bastante alta en comparación con el bajo tiempo de ejecución. La mejor combinación ha sido ε=5000, δ=6000, *minEps*=10 de la que se puede extraer que cuanto mayor es la ventana temporal y espacial, manteniendo un número razonable de puntos mínimos, ofrecerá unos resultados competitivos en comparación con el resto de métodos.

Algoritmo	Parámetros	Precisión	MAE
Ad-Hoc	δ=3600	0.0361754	**0.3487827**
	δ=10800	0.0349337	0.3637682
Sim. tray.	Hausdorff	0.0175497	0.5784518
	DTW	0.0206126	1.2903128
ST-DBSCAN	ε=5000, *th*=6000, *minEps*=1	**0.0369205**	0.4251326
	ε=5000, *th*=6000, *minEps*=10	0.0316225	0.3530357
	ε=5000, *th*=6000, *minEps*=10	**0.0369205**	0.4000756
	ε=1000, *th*=2000, *minEps*=1	0.0336921	0.3530357
	ε=4000, *th*=2000, *minEps*=1	0.0277318	0.3525041
Convoy	*min points*=2, *lifetime*=10, *dist max*=10^{-2}	0.0362583	0.3566276
	min points=20, *lifetime*=2, *dist max*=10^{-2}	0.0332781	1.0634231
	min points=2, *lifetime*=2, *dist max*=10^{-3}	0.0243377	0.5973882
	min points=2, *lifetime*=5, *dist max*=10^{-2}	0.0352649	0.4018430
Convoy partials	*min points*=2, *lifetime*=2, *dist max*=10^{-3}	0.0360927	0.3925327
	min points=2, *lifetime*=2, *dist max*=10^{-1}	0.0368377	0.3525041
	min points=3, *lifetime*=5, *dist max*=10^{-1}	0.0276490	0.7646296
Flock	ε=10, μ 2, δ=2	**0.0369205**	0.3530357
	ε=0.1, μ 2, δ=10	**0.0369205**	0.3704636
	ε=10, μ 2, δ=2	**0.0369205**	0.3530357
Flock partials	ε=10, μ 2, δ=2	0.0368377	0.3525041
	ε=0.1, μ 2, δ=10	0.0355132	0.3530357
	ε=0.1, μ 2, δ=2	0.0368377	0.3670011

Tabla 4.5: Comparativa de precisión y MAE para el recomendador KNN utilizando los vecinos calculados por similitud.

En la Figura 4.5 se representa la evolución de la precisión y MAE de la mejor combinación para cada uno de los métodos en función del número de vecinos. En la Figura 4.5(a), podemos ver la evolución

de la precisión donde los métodos Convoy, Flock, ST-DBSCAN y Ad-Hoc destacan por mantenerse sin apenas variación en una línea prácticamente recta. Esto es debido a que según lo visto en la tabla 4.4, el número promedio de vecinos oscila entre 2 y 20, por lo que el incremento de los K vecinos en el recomendador no mejorará al no haber calculado ningún vecino similar para esta. En cambio DTW y Hausdorff, ya que calculan la similitud entre todos los usuarios, sí que muestra un aumento conforme mayor es el número de vecinos. Esta misma justificación aplica para la gráfica 4.5(b), ya que los vecinos más similares los ordenamos de forma decreciente al calcular la similitud. Por ello, cuanto menor es el número de vecinos, más preciso es el cálculo acercándose la similitud a 1, y al ir alejándose la K, disminuye hasta acercarse al 0. Como resultado de esto, se puede ver cómo aumenta rápidamente.



(a) Evolución precisión recomendador.

(b) Evolución MAE recomendador.

Figura 4.5: Representación de la evolución Precisión y MAE para el recomendador KNN utilizando los vecinos calculados por similitud.

4.7. Comparativa entre recomendadores con similitud de trayectorias vs estándar

En la tabla 4.6 se muestran los errores mínimos para cada uno de los métodos KNN implementados utilizando los patrones de movimiento para detectar vecinos, y lo comparamos contra recomendadores basados en recomendación de matrices como SVD y NMF, un recomendador Random y KNN sin tener en cuenta los patrones como similitud.

Recomendador	KNN Ad-Hoc	KNN Convoy	KNN DTW	KNN Flock	KNN Hausdorff	KNN ST-DBSCAN	SVD	Random	KNN Baseline	NMF
MAE	0.363768	0.352504	1.290313	0.352504	0.578452	0.353036	0.333831	0.475569	0.343762	**0.272056**

Tabla 4.6: Comparativa recomendador KNN con similitud y otros recomendadores.

Se puede apreciar que la mejor puntuación la obtiene NMF, seguido de Flock y Convoy. Esto es debido a que dichos algoritmos obtienen vecinos con unos criterios selectivos, de modo que nos garantiza que encontrará los vecinos más exactos para el usuario. Queda destacada la proximidad de los vecinos encontrados por el método ST-DBSCAN debido a la simpleza y rapidez de este, demostrando poder conseguir unos resultados relativamente óptimos sin gran carga computacional.

5 Conclusiones y trabajo futuro

En este capítulo se recogen las conclusiones extraídas del proyecto, explicando los resultados y el potencial para futuros trabajos.

5.1. Conclusiones

Con la aparición de las nuevas tecnologías, cada vez se utiliza menos el mapa tradicional y las recomendaciones verbales de conocidos, dando lugar a la utilización de aplicaciones para buscar nuevos lugares de interés. Esto ha iniciado el estudio de nuevos sistemas de recomendación para tratar de mejorar la experiencia de los usuarios a través de sugerencias más precisas. Para el estudio de estas técnicas, se destinan grandes presupuestos y está formado por miles de profesionales en todo el mundo buscando obtener el sistema de recomendación más preciso para cada aplicación.

Tras un gran trabajo de investigación y búsqueda de algoritmos en la literatura de patrones de movimiento, nos dimos cuenta de las principales ventajas e inconvenientes de cada uno de ellos, destacando principalmente la paralelización. Tras diversos ajustes descubrimos su potencial, y lo explotamos con múltiples pruebas.

En este trabajo hemos comprobado cómo puede ser de utilidad el estudio de las trayectorias de los usuarios a fin de tratar de encontrar aquellos más similares y utilizarlos posteriormente para la recomendación. Nos basamos en el hecho de que si hemos viajado junto a varios usuarios durante N instantes de tiempo, y pese a no conocerlos, es muy posible que estos compartan gustos con nosotros. Tras haber hecho diversas pruebas, nos dimos cuenta de que esta primera aproximación es muy válida y tiene mucho potencial para el estudio en este área. Esto dio lugar a la publicación de dos artículos cortos en las conferencias *ACM UMAP 2020* [33] y *CIRCLE 2020* [34], donde expusimos estos avances para los sistemas de recomendación, así como un subconjunto de los resultados mostrados en esta memoria.

El aspecto más determinante a la hora de encontrar resultados precisos han sido las restricciones de cada uno de los algoritmos, puesto que algunos como Flock exigían que todos los instantes de tiempo entre los usuarios fuesen consecutivos, lo cual daba pie a perder muchos potenciales vecinos. No obstante, hay que encontrar un equilibrio entre la exactitud al encontrar estos usuarios similares y la complejidad o costes computacionales requeridos. En este trabajo hemos presentado una solución aproximada, a partir de otras funciones de similitud entre trayectorias, que ha dado resultados con bastante potencial.

También hemos comprobado la utilidad del KNN propuesto utilizando las similitudes entre usuarios

como vecinos frente a otros algoritmos ya implementados, como aquellos basados en matrices, a pesar de la simplicidad de dicho método con respecto a otros sistemas de recomendación.

Por último, todo nuestro código y los scripts de pruebas están incluidos en el repositorio:

`https://github.com/storrijos/FourSquare-Dataset`

5.2. Trabajo Futuro

En el futuro nos gustaría seguir mejorando el rendimiento de los actuales algoritmos, tratando de llegar a ejecuciones paralelas para poder examinar datasets mayores. De este modo podría simularse este sistema en un entorno más realista. También nos gustaría seguir perfeccionando la búsqueda de usuarios similares mediante el uso de otras restricciones a la hora de limpiar el dataset, por ejemplo proporcionando una mayor ventana temporal u otros parámetros más específicos para los algoritmos.

Otra de las herramientas que proporcionan estos datasets son la validación de la veracidad de las similitudes encontradas con las cuentas de redes sociales asociadas a cada usuario, como Twitter. Esto nos permitiría comprobar qué usuarios sigue en Twitter cada uno de los usuarios del dataset, y así poder verificar si, de forma general, la solución que usa los patrones de movimiento es capaz de encontrar potenciales amigos.

Desde un punto de vista más general, nos gustaría experimentar con los patrones de movimiento que se han implementado en este proyecto más allá de los recomendadores basados en vecinos cercanos. Creemos que los usuarios encontrados según estos patrones pueden ser muy útiles para solventar el problema de la dispersión (*sparsity*) en los datos, de forma que se podrían generar subconjuntos de los datos de entrenamiento para obtener ejecuciones más eficientes, al igual que se ha hecho en el trabajo para acelerar los algoritmos de Flock y Convoy, pero aplicado a cualquier tipo de algoritmo de recomendación, como los basados en factorización de matrices.

Un estudio que se ha quedado fuera de esta memoria pero que se ha comenzado a raíz de este trabajo (y cuyos resultados preliminares están incluidos en una de las dos publicaciones mencionadas anteriormente), es el de analizar el efecto de estos algoritmos de patrones en el tipo de usuario del sistema de recomendación turístico, es decir, si es un turista o un local (un usuario que conoce la ciudad). Estos resultados iniciales han mostrado una tendencia positiva hacia los usuarios turistas, pero en el futuro queremos entender mejor por qué ocurre y proponer otros algoritmos que exploten estos datos.

Bibliografía

[1] Y. Koren and R. M. Bell, "Advances in collaborative filtering. in recommender systems handbook," *Springer*, p. 77–118, 2015.

[2] C. D. Xia Ning and G. Karypis, "A comprehensive survey of neighborhood-based recommendation methods. in recommender systems handbook," *Springer*, p. 37–76, 2015.

[3] A. S. Yong Liu, Wei Wei and C. Miao, "Exploiting geographical neighborhood characteristics for location recommendation. in proceedings of the 23rd acm international conference on conference on information and knowledge management," *CIKM ACM*, p. 739–748, 2014.

[4] G. C. Yiding Liu, Tuan-Anh Pham and Q. Yuan, "An experimental evaluation of point-of-interest recommendation in location-based social networks," *PVLDB*, vol. 10, no. 10, p. 1010–1021, 2017.

[5] Y. Zheng and X. Zhou, eds., *Computing with Spatial Trajectories*. Springer, 2011.

[6] H. W. Qi Fan, Dongxiang Zhang and K.-L. Tan, "A general and parallel platform for mining co-movement patterns over large-scale trajectories," *PVLDB*, vol. 10, no. 4, pp. 313–324, 2016. (Descargar).

[7] O. E. C. Rosero and A. O. C. Romero, "Fpflock: An algorithm to find frequent flock patterns in spatio-temporal databases," *Int. J. Online Eng*, 2018.

[8] M. J. Wen Chen and J. Wang, "T-dbscan: A spatiotemporal density clustering for gps trajectory segmentation," *Int. J. Online Eng*, vol. 10, no. 6, p. 19–24, 2014. (Descargar).

[9] Wikipedia contributors, "Hausdorff distance," 2020. (Descargar).

[10] G. P. A. M. K. Ovidiu Csillik, Mariana Belgiu, "Object-based time-constrained dynamic time warping classification of crops using sentinel-2," *Remote Sens*, vol. 11, p. 1257, 10 2019. (Descargar).

[11] P. Senin, "Dynamic time warping algorithm review," 01 2009. (Descargar).

[12] T. Uno, M. Kiyomi, and H. Arimura, "Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets," 01 2004. (Descargar).

[13] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, "Discovery of convoys in trajectory databases," *ArXiv*, vol. abs/1002.0963, 2008. (Descargar).

[14] "St-dbscan: An algorithm for clustering spatial–temporal data," *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208 – 221, 2007. (Descargar).

[15] Wikipedia contributors, "Hausdorff distance," 2020. (Descargar).

[16] T. Kieu, "Trajectory mining." https://github.com/tungk/TrajectoryMining, 2015.

[17] O. E. C. Rosero, "Fpflock." https://github.com/poldrosky/FPFlock/tree/master/Src, 2014.

[18] abensaid, "Coherent-moving-cluster." https://github.com/abensaid/Coherent-Moving-Cluster, 2018.

[19] BIGSEA, "An implementation of st-dbscan algorithm using python language." https://github.com/eubr-bigsea/py-st-dbscan, 2017.

[20] D. Spyropoulos, "Trajectory analysis and classification in python pandas and scikit learn." https://github.com/jim-spyropoulos/Trajectory-Analysis-and-Classification-in-Python-Pandas-and-Scikit-Learn-, 2018.

[21] Shathra, "Comparing different clustering methods and similarity metrics on trajectory datasets." https://github.com/Shathra/comparing-trajectory-clustering-methods, 2017.

[22] A. B. Ignacio Alvarez-Hamelina, Luca Dall'Astaa and A. Vespignanib, "k-core decomposition: a tool for the visualization of large scale networks," *Advances in Neural Information Processing Systems 18*, vol. 41, 2006. (Descargar).

[23] F. Ricci, L. Rokach, and B. Shapira, *Recommender Systems Handbook*. Springer Publishing Company, Incorporated, 2nd ed., 2015.

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

[25] F. Ricci, L. Rokach, and B. Shapira, *Recommender Systems Handbook*. Springer Publishing Company, Incorporated, 2nd ed., 2015.

[26] F. Ricci, L. Rokach, and B. Shapira, *Recommender Systems Handbook*. Springer Publishing Company, Incorporated, 2nd ed., 2015.

[27] A. Bellogín, P. Castells, and I. Cantador, "Precision-oriented evaluation of recommender systems: an algorithmic comparison," in *Proceedings of the 2011 ACM Conference on Recommender Systems, RecSys 2011, Chicago, IL, USA, October 23-27, 2011* (B. Mobasher, R. D. Burke, D. Jannach, and G. Adomavicius, eds.), pp. 333–336, ACM, 2011.

[28] BIGSEA, "Official project repository for the setuptools build system." `https://github.com/pypa/setuptools`, 2006.

[29] Pandas, "Pandas, fast, powerful, flexible and easy to use open source data analysis and manipulation tool." `https://github.com/pandas-dev/pandas`, 2006.

[30] G. Guo, J. yi Zhang, Z. Sun, and N. Yorke-Smith, "Librec: A java library for recommender systems," in *UMAP Workshops*, 2015.

[31] E. Palumbo, G. Rizzo, R. Troncy, and E. Baralis, "Predicting your next stop-over from location-based social network data with recurrent neural networks," in *RECSYS 2017, 2nd ACM International Workshop on Recommenders in Tourism (RecTour'17), CEUR Proceedings Vol. 1906, August 27-31, 2017, Como, Italy*, (Como, ITALY), 08 2017. (Descargar).

[32] I. R. Brilhante, J. A. F. de Macêdo, F. M. Nardini, R. Perego, and C. Renso, "Where shall we go today?: planning touristic tours with tripbuilder," in *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013* (Q. He, A. Iyengar, W. Nejdl, J. Pei, and R. Rastogi, eds.), pp. 757–762, ACM, 2013.

[33] S. Torrijos, A. Bellogín, and P. Sánchez, "Discovering related users in location-based social networks," in *User Modeling, Adaptation, and Personalization - 28th International Conference, UMAP 2020, Genoa, Italy, July 12-18, 2020. Proceedings*, Lecture Notes in Computer Science, Springer, 2020.

[34] S. Torrijos and A. Bellogín, "Analysis of trajectory pattern mining methods for recommendation: Extended abstract," in *Proceedings of the Joint Conference of the Information Retrieval Communities in Europe, CIRCLE 2020, Samatan, France, July 6-9, 2020*, 2020.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID