



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería  
Informática**

**Urban Street Mapping  
Transfer**



Presentado por Mario Hurtado Ubierna  
en Universidad de Burgos — 28 de agosto  
de 2023

Tutor: Virginia Ahedo García y Jesús Manuel  
Maudes Raedo







UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



Dña. Virginia Ahedo, profesora del departamento de Ingeniería de Organización, área de Organización de Empresas y D. Jesús Manuel Maudes, profesor del departamento de Ingeniería Informática, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Mario Hurtado Ubierna, con DNI 71365182T, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado « > ».

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 28 de agosto de 2023

Vº. Bº. del Tutor:

Vº. Bº. del co-tutor:

Dña. Virginia Ahedo

D. Jesús Manuel Maudes





## Resumen

Uno de los principales factores que considerar cuando se busca abrir un nuevo negocio es la localización. Esto plantea un problema conocido como el *Location Problem*, el cual busca encontrar la ubicación más adecuada que pueda reportar más beneficios. En el caso del proyecto se aplicará al *retailing*, es decir, a tiendas minoristas.

Existen distintas aproximaciones a este problema, en este proyecto se aplicará un enfoque desde la ciencia de redes, teniendo en consideración las distintas categorías comerciales y demás servicios en determinadas ciudades con el objetivo de crear un sistema de recomendación con capacidad de ofrecer propuestas de comercios. Para ello se extraerán las relaciones de atracción y repulsión entre las distintas categorías en base a la distribución geográfica de estas.

Para lograr lo anterior se utilizarán datos extraídos de una API de geolocalización de un proyecto conocido como *OpenStreetMap* de uso libre, así como, *Neo4j*, que forma parte de un nuevo paradigma conocido como la orientación a grafos en el contexto de las bases de datos.

Además de contemplar recomendaciones a nivel local, también se empleará la transferencia de información entre ciudades, pudiendo usar las relaciones comerciales de una ciudad sobre otra en lugar de las propias. Con este objetivo se creará también un modelo de inteligencia artificial que integre los datos de todas las ciudades.

Para facilitar el uso de la herramienta de este proyecto, se desarrollará una aplicación web sencilla que permita a los usuarios obtener recomendaciones basadas la ubicación.

## Descriptores

Ciencia de redes, sistema de recomendación, comercio minorista, *machine learning*, web, aprendizaje transferido, API de geolocalización, base de datos de grafos.

## Abstract

One of the main factors to consider when looking to open a new business is location. This poses a problem known as the Location Problem, which seeks to find the most suitable location that can bring the most benefits. In the case of this project, it will be applied to retailing, i.e. retail stores.

There are different approaches to this problem, in this project we will apply a network science approach, taking into consideration the different commercial categories and other services in certain cities in order to create a recommendation system with the ability to offer proposals of stores. For this purpose, the attraction and repulsion relationships between the different categories will be extracted based on their geographic distribution.

To achieve this, we will use data extracted from a geolocation API of a project known as OpenStreetMap of free use, as well as Neo4j, which is part of a new paradigm known as graph orientation in the context of databases.

In addition to contemplating recommendations at the local level, the transfer of information between cities will also be used, being able to use the commercial relations of one city over another instead of its own. To this end, an artificial intelligence model will also be created to integrate data from all cities.

To facilitate the use of this project's tool, a simple web application will be developed to allow users to obtain location-based recommendations.

## Keywords

Network science, recommendation system, retailing, machine learning, web, transfer learning, geolocation API, graph database



---

# Índice general

---

Índice general	iii
Índice de figuras	v
Índice de tablas	vi
Introducción	1
Objetivos del proyecto	3
Conceptos teóricos	5
3.1. <i>Retail Location Problem</i> . . . . .	5
3.2. Teoría de grafos . . . . .	5
3.3. Ciencia de Redes . . . . .	6
3.4. Técnicas . . . . .	7
3.5. Índices de calidad . . . . .	10
3.6. <i>Transfer Learning</i> . . . . .	12
Técnicas y herramientas	13
4.1. Herramientas de gestión de proyectos . . . . .	13
4.2. Lenguajes de programación . . . . .	14
4.3. Base de datos . . . . .	14
4.4. Librerías . . . . .	15
4.5. APIs . . . . .	17
Aspectos relevantes del desarrollo del proyecto	19
5.1. Carga y almacenamiento de ubicaciones . . . . .	19

<b>Trabajos relacionados</b>	<b>35</b>
6.1. <i>Knowledge Transfer in Commercial Feature Extraction for the Retail Store Location Problem</i> . . . . .	35
6.2. <i>Retail Store Location Selection Problem with Multiple Analytical Hierarchy Process of Decision Making an Application in Turkey</i> . . . . .	37
<b>Conclusiones y Líneas de trabajo futuras</b>	<b>39</b>
7.1. Conclusiones . . . . .	39
7.2. Líneas de trabajo futuras . . . . .	42
<b>Bibliografía</b>	<b>45</b>

---

## Índice de figuras

---

3.1. Ejemplo del método <i>Permutation</i> . . . . .	9
3.2. Ejemplo del método <i>Rewiring</i> . . . . .	9
5.1. Ejemplo de creación de enlaces por proximidad . . . . .	21
5.2. Ejemplo de interacción entre categorías . . . . .	23
5.3. Ejemplo de red de categorías . . . . .	24
5.4. Representación de las simulaciones del método <i>Permutation</i> . . . . .	25
5.5. Representación de almacenamiento de coeficientes de Jensen en la base de datos . . . . .	26
6.1. <i>Mean Reciprocal Rank</i> por método y ciudad . . . . .	36
6.2. <i>Mean Reciprocal Rank</i> al utilizar transferencia . . . . .	36

---

# Índice de tablas

---

5.1. Nodos por ciudad en la primera carga de datos . . . . .	20
5.2. Nodos iniciales por ciudad en la segunda carga de datos . . . . .	22
5.3. Nodos por ciudad en la tercera carga de datos . . . . .	28

---

# Introducción

---

El conocido como *Retail Location Problem* consiste en un problema al que se enfrentan los negocios a la hora de elegir una ubicación donde abrir un nuevo establecimiento. Un correcto emplazamiento puede incrementar enormemente el rendimiento del negocio, proporcionando ventajas que la competencia difícilmente puede imitar. Esto puede suponer la diferencia entre el éxito o el fracaso

*Urban Street Mapping Transfer* se trata de un proyecto que busca ofrecer soluciones para este problema. La técnica que se utilizará para ello consiste en el análisis de la estructura comercial de cada ciudad mediante el uso de conceptos propios de la ciencia de redes así como de distintos métodos matemáticos y estadísticos que nos permitan estimar qué ubicaciones pueden ser más provechosas para un negocio. También se emplearán modelos de inteligencia artificial para integrar los resultados del análisis en un sistema de recomendación.

Además de lo anterior se utilizará *transfer learning*, una técnica propia del *machine learning* que permitirá utilizar datos de unas ciudades para poder realizar recomendaciones sobre otras.



---

## Objetivos del proyecto

---

Este proyecto tiene como objetivo principal la creación de recomendaciones de categorías comerciales dadas ubicaciones, de forma que se pueda estimar el mejor lugar para como localización de un determinado tipo de establecimiento comercial. Para ello se utilizarán distintos índices de calidad propuestos para este problema así como su uso combinado mediante un modelo de inteligencia artificial destinado a problemas de clasificación. Para facilitar el uso de la herramienta se diseñará una aplicación web sencilla que permita hacer las recomendaciones a un usuario.

Los objetivos para llevar a cabo el proyecto son los siguientes:

1. Obtención de establecimientos comerciales con sus respectivas categorías de distintas ciudades mediante el uso de una API de geolocalización.
2. Uso de una base de datos orientada a grafos donde almacenar las ubicaciones comerciales de forma que podamos operar sobre ella como se haría en una red.
3. Análisis de la estructura comercial de cada ciudad así como de la interacción entre las distintas categorías.
4. Uso y cálculo de distintos índices de calidad con objetivo de realizar recomendaciones.
5. Uso de transferencia de información entre la estructura comercial de distintas ciudades.
6. Creación de un sistema de recomendación basado en un modelo de inteligencia artificial.

7. Creación de una aplicación web con objetivo de facilitar el uso de la herramienta.



---

## Conceptos teóricos

---

La cuestión que se busca abordar con este proyecto es el *Retail Location Problem*. Este consiste en la elección de la ubicación en la que colocar una nueva tienda de forma que esta le aporte el máximo beneficio posible. Para abordar esto se emplearán técnicas propias de la ciencia de redes y del aprendizaje automático.

### 3.1. *Retail Location Problem*

El conocido como «Retail Location Problem» trata de un problema enfrentado por las empresas pertenecientes al comercio minorista al elegir una ubicación donde empezar un nuevo negocio. La ubicación, lejos de ser un factor carente de importancia puede marcar la diferencia entre el éxito o el fracaso del comercio. Un negocio adecuadamente posicionado cuenta con ventajas sobre la competencia que pueden hacer aumentar significativamente su rendimiento sobre el resto [1, 2].

La elección de la ubicación puede hacerse ateniendo a distintos factores, logísticos, poblacionales, fiscales, entre otros. En este caso se hará basándose en cómo complementan los distintos comercios cercanos a una ubicación a un hipotético nuevo negocio, considerando su efecto sobre este.

### 3.2. Teoría de grafos

Los grafos consisten en unas estructuras matemáticas compuestas por dos elementos [?]:

- $V$ : Vértices o nodos. Suponen la entidad más básica del grafo, pudiendo representar distintos conceptos.
- $E$ : Enlaces o aristas. Representan las relaciones entre los distintos nodos de un grafo.

Dada su naturaleza, los grafos han sido usados habitualmente para modelar relaciones entre objetos.

Los nodos, además de constituir el elemento más básico, pueden contener ciertas características asociadas, como pertenencias a un grupo.

Existen distintos tipos en función de como se modelen las relaciones entre objetos. Si las relaciones no cuentan con dirección alguna se les conoce como no dirigidos, si existe alguna restricción en las direcciones con las que se recorren los enlaces entonces se trata de un grafo dirigido; y en caso de permitir múltiples enlaces entre cada par de nodos se trataría de un multigrafo. También pueden presentar una ponderación en los enlaces entre nodos, siendo de esta forma un grafo pesado [?].

## Modelos nulos

En el ámbito de las matemáticas y la teoría de grafos un modelo nulo se trata de un modelo con una generación aleatoria en alguno de sus aspectos. El objetivo detrás de estos consiste en establecerlos como punto de comparación para analizar si algunas propiedades presentes en otro grafo se alejan de ser casuales, como lo serían en estos modelos nulos. De esta forma se puede extraer ciertas características distintivas de los grafos sujetos a análisis.

## 3.3. Ciencia de Redes

La ciencia de redes se trata de una disciplina que utiliza los conocimientos que la teoría de grafos provee para representar elementos y relaciones de la realidad y de esta forma ser capaces de estudiar y analizar sistemas complejos. Los grafos utilizados por esta rama de conocimiento se suelen conocer como redes.

Además de basarse en los conceptos de la teoría de grafos, también aplica técnicas propias de la informática y el «Machine Learning» para facilitar las labores de análisis de redes. Algunos ejemplos de aplicación de estas técnicas podrían ser la predicción de posibles enlaces entre nodos, detección de comunidades o la clasificación de nodos en distintas categorías.

En los últimos años ha obtenido gran relevancia al aplicarse a ámbitos como las redes sociales, de transporte, de transmisión de enfermedades, entre otros. En el caso de este trabajo, se aplicará para modelar la estructura comercial de las distintas ciudades con el objetivo de hacer recomendaciones en base a la ubicación para determinados negocios.

### 3.4. Técnicas

Con el objetivo de extraer características de la estructura comercial de cada ciudad se proponen una serie de métodos. Estos nos proveen coeficientes entre las distintas categorías que nos permitirán extraer las relaciones de afinidad y repulsión; además de posteriormente permitirnos los cálculos de los índices de calidad para cada ubicación, pudiendo realizar con ellos las recomendaciones que buscamos en este trabajo.

Se trata de los siguientes:

- Jensen
- *Permutation*
- *Rewiring*

Antes de poder aplicar estos métodos será necesario contar con una representación en forma de red de la estructura de la ciudad. Los nodos de este grafo representarán cada una de las ubicaciones. En cuanto a las relaciones, estas se crearan en base a la proximidad espacial entre nodos. Para esto se tomará una distancia de 100 metros como máximo para la creación de un enlace.

A continuación, se hará una descripción de cada uno de estos métodos. Conviene hacer una breve explicación de los términos utilizados.

- $T$ : Conjunto de todas las distintas ubicaciones.
- $A, B$ : Conjuntos de ubicaciones de determinadas categorías.
- `null_model`: Obtenido a partir de las simulaciones de Monte Carlo de *Permutation* y *Rewiring*.
- $N_s(p, r)$ : Tamaño del conjunto de ubicaciones de categoría  $s$  en una proximidad de  $r$  metros.

## Jensen

Jensen propone unos coeficientes para una categoría sobre otra. Estos pueden ser obtenidos simplemente mediante los datos que provee la estructura comercial de la ciudad bajo análisis, sin requerir la aplicación de métodos matemáticos o estadísticos complejos. Este método puede obtener tanto las relaciones de atracción como de repulsión entre categorías comerciales. Las primeras serán aquellas con un valor por encima de 1, y las segundas aquellas en las que esté por debajo de 1.

El cálculo de estos coeficientes es distinto cuando se trata de una categoría sobre sí misma con respecto a una categoría sobre otra distinta.

$$M_{AA} = \frac{|T| - 1}{|A|(|A| - 1)} \sum_{a \in A} \frac{N_A(a, r)}{N_T(a, r)}$$

$$M_{AB} = \frac{|T| - |A|}{|A||B|} \sum_{a \in A} \frac{N_B(a, r)}{N_T(a, r) - N_A(a, r)}$$

El cálculo de estos coeficientes llevará a la obtención de una matriz de interacciones que podrá ser utilizada para posteriores cálculos.

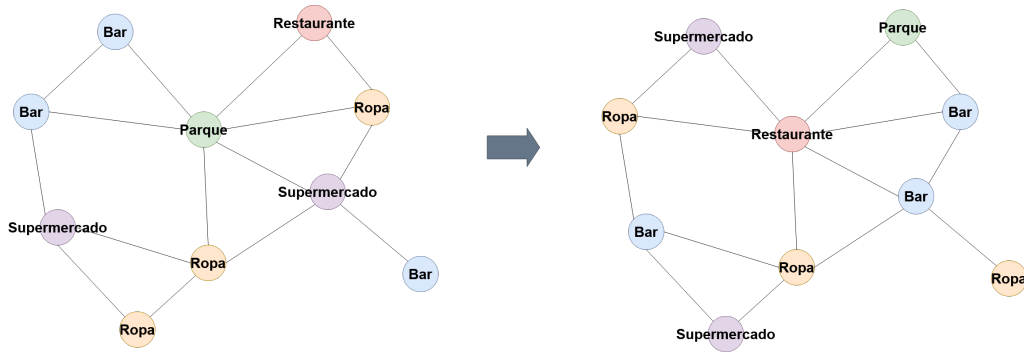
## Permutation

El método *Permutation* al contrario del anterior, requiere de la aplicación de simulaciones de Monte Carlo con modelos nulos sobre la red. En este caso se crearán redes aleatorias, pero con la restricción de que se mantendrá la estructura real de la red, permutándose aleatoriamente las categorías asociadas a cada nodo.

De cada simulación se obtendrá el valor de interacción entre categorías, para posteriormente obtener la media y la desviación típica de los resultados de las simulaciones.

Con ello se podrá obtener el Z-Score para cada par de categorías, con un funcionamiento análogo a los coeficiente de Jensen. Se obtendrá una matriz simétrica con ellos.

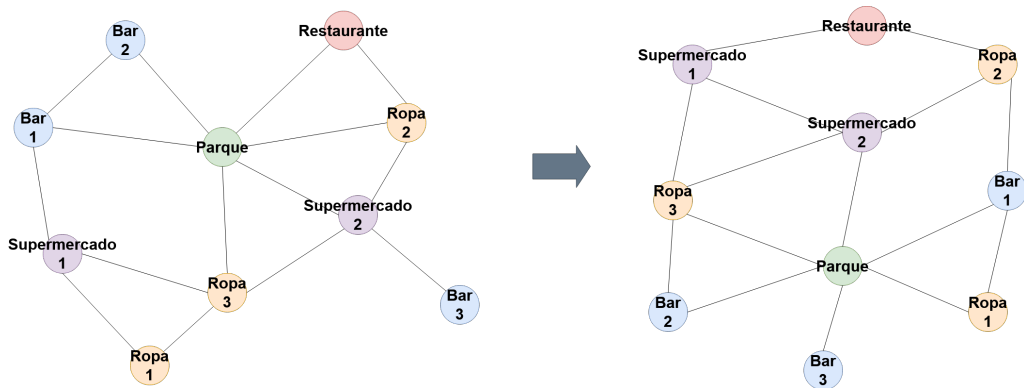
$$Z_{AB} = \frac{x_{AB} - \overline{x_{AB}^{null\_model}}}{s_{AB}^{null\_model}}$$

Figura 3.1: Ejemplo del método *Permutation*

## Rewiring

*Rewiring* se trata de un método similar al anterior, también está basado en la generación de modelos nulos con simulaciones de Monte Carlo. La diferencia entre este y el anterior está en las restricciones. En este la categoría comercial es mantenida en cada uno de los nodos, mientras que los enlaces entre nodos son aleatorizados, con la única restricción de que se debe mantener el grado que los nodos poseían en la red original.

Al igual que en el anterior se obtendrá una matriz con los Z-Score por pares de categorías.

Figura 3.2: Ejemplo del método *Rewiring*

### 3.5. Índices de calidad

Con los coeficientes de interacción entre categorías de la anterior sección se puede obtener los llamados *quality indices* o índices de calidad. Con estos se puede determinar la adecuación de un negocio de cierta categoría en una ubicación determinada. En el caso de Jensen se aplica el valor del logaritmo del coeficiente, es decir,  $a_{ij} = \log(M_{ij})$ , mientras que en el caso de *Permutation* y *Rewiring* se utilizará el propio Z-Score para esas categorías,  $a_{ij} = Z_{ij}$ .

$$Q_i(x, y) \equiv \sum_{j=1}^N a_{ij} (nei_{ij}(x, y) - \overline{nei_{ij}})$$

- $a_{ij}$ : Interacción de la matriz de coeficientes entre categorías  $i$  y  $j$ .
- $N$ : Número total de categorías.
- $nei_{ij}(x, y)$ : Número de negocios vecinos con categoría  $j$  de la ubicación  $(x, y)$ .
- $\overline{nei_{ij}}$ : Promedio de negocios vecinos con categoría  $j$  sobre negocios con categoría  $i$ .

Con esto se obtendría la adecuación de una ubicación  $(x, y)$  con la categoría  $i$ . Si se parte de un conjunto predefinido de categorías comerciales se puede encontrar el negocio más adecuado para una ubicación  $(x, y)$  calculando los índices de calidad para todas las categorías, siendo el más apropiado aquel con mayor valor de  $Q_i$ .

#### Índices *Raw*

Detrás de los índices de calidad de la sección anterior se encuentra la idea de que una ubicación que se asemeje al promedio de negocios reales de cierta categoría puede ser una buena ubicación para un nuevo negocio con dicha categoría propuesto por Jensen. Lo anterior se manifiesta al sustraer el valor promedio para cada categoría  $\overline{nei_{ij}}$  en el cálculo de los índices de calidad. Por ello se proponen unos nuevos que no tienen en consideración los valores medios. Su cálculo es el siguiente:

$$Q_i(x, y) \equiv \sum_{j=1}^N a_{ij} (nei_{ij}(x, y))$$

### ***Mean Reciprocal Rank***

Los índices de calidad nos proporcionan una serie de categorías comerciales ordenadas en función de la adecuación de cada una. Para poder evaluar la eficiencia de cada índice se emplea la medida conocida como *Mean Reciprocal Rank*. Esta se suele emplear en sistemas de recomendación para evaluar su rendimiento usando la posición que ocupa la categoría real sobre el *ranking* provisto, en este caso, por los índices de calidad. Sigue la siguiente formula:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

- $Q$ : Conjunto de ubicaciones con categoría conocida.
- $rank_i$ : Posición que ocupa la categoría real de la ubicación en el *ranking* ofrecido por un índice de calidad.

### **Uso combinado de índices de calidad**

Si bien el uso simple de los índices de calidad ya es suficiente para obtener recomendaciones, puede darse la situación en que estas difieran dependiendo del método utilizado para obtener los índices. Si bien podemos ver qué método es más eficaz utilizando el MRR y decantarnos por uno o por otro, puede que el motivo por el que difieran es debido a que cada método esté captando un aspecto distinto del problema. Para solventar eso se puede recurrir al empleo de un modelo de inteligencia artificial de clasificación para obtener recomendaciones usando todos los índices de calidad disponibles.

Para poder entrenar el modelo se debe contar primero con un conjunto de datos. Este lo podremos construir con una serie de ubicaciones de las que conozcamos su categoría real, siendo esta la etiqueta a predecir. El resto de atributos se corresponderán con los valores que los métodos proporcionan a cada categoría para las ubicaciones con categoría conocida.

El modelo entrenado podrá realizar las funciones de sistema de recomendación, pudiendo a su vez ser evaluado usando el MRR.

En este trabajo se empleará un *Random Forest*, clasificador de la familia de los *ensembles*. El uso de un modelo de esta familia es debido a las ventajas que estos poseen frente a los modelos simples, ofreciendo mejor rendimiento y versatilidad.

### 3.6. *Transfer Learning*

El conocido como *Transfer Learning* se trata de una técnica propia del aprendizaje automático que consiste en utilizar conocimiento obtenido previamente para resolver nuevos problemas similares. Esto permite mejorar el rendimiento ya que se cuentan con datos previos que pueden ser reutilizados.

En el caso de este trabajo el *Transfer Learning* se puede aplicar de dos formas distintas:

1. Utilizar datos provenientes de una ciudad para realizar recomendaciones sobre otra distinta. Para ello se requiere un modelo de inteligencia artificial, utilizando los índices de calidad previamente calculados tanto para la ciudad origen como destino.
2. Calcular los índices de calidad de una ciudad utilizando la matriz de interacciones de los distintos métodos de otra.

La aplicación de esta técnica puede ser de gran importancia ya que permitiría realizar predicciones sobre nuevas ciudades utilizando datos con los que ya se contaba anteriormente. Además permitiría analizar si las distintas ciudades comparten una estructura comercial similar, pudiendo utilizarse con bastante confianza para predecir sobre nuevas; o si por el contrario cada ciudad posee una estructura distinta y más específica con respecto a otras.

Al igual que con las predicciones que se realizaban a nivel local, estas podrán ser evaluadas utilizando el MRR.



---

# Técnicas y herramientas

---

En este apartado se tratarán las distintas técnicas y herramientas utilizadas durante el desarrollo del proyecto.

## 4.1. Herramientas de gestión de proyectos

Para la planificación del proyecto se ha buscado una herramienta que soporte los elementos característicos de un desarrollo ágil como SCRUM permitiendo la creación de distintas tareas y su gestión en el tiempo de desarrollo de la aplicación.

Inicialmente se optó por Zenhub, aunque mientras se estaba desarrollando el proyecto esta pasó a requerir una licencia de pago para su uso, por lo que se substituyó esta por Zube.

### Zenhub

Zenhub se trata de una herramienta de gestión de proyectos basada en las metodologías ágiles, por lo que cuenta con funcionalidades que facilitan la creación de tareas así como su gestión. Además de crear tareas nos permite asignar a estas un *Sprint*, puntos de poker (utilizados en SCRUM) o poder definir aquellas tareas que lo requieran como *Epics* para diferenciarlas de las otras.

Un punto importante para su elección inicial fue que esta cuenta con integración en *GitHub*, por lo que facilita la gestión de tareas al hacerla directamente desde el propio repositorio.

## Zube

Una vez Zenhub pasó a ser una herramienta de pago se optó por el uso de Zube. Esta cuenta con muchos de los elementos que Zenhub también tiene. Permite definir tareas y gestionarlal mediante un tablero *Kanban*, colocando las tareas en un area u otra en función de su estado (Backlog, Ready, In Progress,...)

## 4.2. Lenguajes de programación

Una elección bastante significativa para el proyecto será el lenguaje de programación empleado para su desarrollo.

### Python

Python se trata de un lenguaje interpretado de alto nivel bastante utilizado en ámbitos como el *Machine Learning*. Además de lo anterior cuenta con multitud de librerías de diversos propósitos desarrolladas por los propios usuarios, pudiendo facilitar las labores a realizar en el proyecto. En su elección también influye que es uno de los lenguajes que más hemos utilizado durante la carrera.

### JavaScript

Al igual que Python, JavaScript se trata de un lenguaje interpretado, pero este es utilizado por los navegadores para poder crear páginas web dinámicas. También cuenta con diversas librerías con funcionalidades que permiten manejar el DOM, hacer distintas visualizaciones, entre otras. Este lenguaje será de gran importancia en el apartado web del proyecto.

## 4.3. Base de datos

Dada la naturaleza del proyecto y los datos que se manejarán, una base de datos no relacional puede dar ciertas facilidades en el manejo de estos. También cabe destacar que la información tomará forma de grafo y se tratará usando técnicas propias de la ciencia de redes, por lo que estas necesidades tomarán gran importancia en la decisión de la base de datos.

## Neo4j

Neo4j se trata de una base de datos no relacional de la familia de los grafos escrita en Java. Esta es utilizada por grandes empresas como Intel, Adobe, AstraZeneca o incluso la NASA. También cabe destacar su uso en el terreno de la ciencia de datos gracias a *plugins* con utilidades propias de la ciencia de redes.

Cabe mencionar que cuenta con un lenguaje de consultas propio, *Cypher*, con una sintaxis que facilita la obtención de información en un contexto de grafos.

Expuestos los hechos anteriores, Neo4j se trata de la elección lógica para un proyecto de nuestras características.

### APOC (Awesome Procedures On Cypher)

Como se ha mencionado antes, Neo4j cuenta con *plugins* que facilitan algunas labores en la base de datos. Entre ellos se encuentra APOC, que tiene tanto utilidades un tanto más generales, así como algunas más específicas, como por ejemplo cargar datos directamente de peticiones a una API, que nos será bastante útil en la carga de ubicaciones durante el desarrollo del proyecto.

## 4.4. Librerías

Como se ha mencionado anteriormente, se hará uso de diversas librerías para las labores del proyecto, tanto de Python como de JavaScript.

### Driver de Neo4j

Para el manejo de la base de datos de Neo4j a través de Python existen 3 distintos drivers:

**Driver Oficial** Se trata del driver oficial de Neo4j para Python. Cuenta con cursos para su uso y una extensa documentación.

**Py2Neo** Es un driver creado por la comunidad de Neo4j con una interfaz sencilla. Lamentablemente ya no recibe soporte para las últimas versiones de Neo4j.

**Neomodel** Un driver alternativo al oficial que ofrece un OGM (Object Graph Mapper), similar a los ORMs de otras bases de datos; además de integración con el framework web *Django*.

Dado que Neo4j es una herramienta nueva, que requiere de un proceso de aprendizaje, se ha optado por el driver oficial por su simple uso y la gran cantidad de documentación existente. Se ha desestimado Py2Neo puesto que ya no recibe soporte; y en el caso de Neomodel, las funcionalidades que trae no aportan una ventaja respecto al driver oficial, puesto que en primer lugar no utilizaremos Django, sino Flask para la parte web del proyecto; y en cuanto al OGM que provee este no nos es necesario puesto que no requeriremos de las ventajas que este nos podría aportar.

## OSMPythonTools

OSMPythonTools se trata de una librería que facilita el acceso a la información de los servicios de *Open Street Map* mediante una interfaz simple.

## Flask

Flask se trata de un framework de desarrollo web para Python. Está caracterizado por ser ligero y flexible, además de contar con librerías que expanden sus funcionalidades. Permite el uso de distintas bases de datos además de contar con un motor de plantillas llamado *Jinja2* para el renderizado de páginas web.

Se ha escogido este framework frente a otros debido a su simplicidad y fácil uso, además cubre todas las necesidades del proyecto. Otro punto significativo en su elección es que ya se había trabajado con él durante la carrera en la asignatura de *Diseño y mantenimiento del Software*.

## Flask JWT Extended

Es una librería que extiende las funcionalidades de Flask para poder utilizar JWT (JSON Web Token) para las labores de autenticación de usuarios y en la seguridad de la aplicación.

## Flask WTForms

Extensión de Flask para la creación de formularios web mediante una interfaz simple, permitiendo la validación de estos desde el propio Python.

## Leaflet.js

Se trata de una librería de código abierto de JavaScript que permite la visualización de mapas interactivos en páginas web. Para sus mapas puede utilizar información extraída de *OpenStreetMap* para hacer las visualizaciones. Nos servirá para mostrar las distintas ubicaciones que almacenemos en la base de datos en la web.

## Vis.js

Es una librería de JavaScript cuyo objetivo es realizar visualizaciones interactivas de grafos en la web. Permite personalizar las visualizaciones con distintas configuraciones. Con esta podremos mostrar la información de la base de datos en forma de grafo en la forma que se haría desde el punto de vista de la ciencia de redes.

## Bootstrap 5

Se trata de un framework que se suele utilizar para la creación de interfaces web *responsive*. Cuenta con una extensa documentación y ejemplos para su uso. Facilita el diseño de páginas web ofreciendo componentes reutilizables pudiendo utilizarse directamente en las plantillas HTML.

## 4.5. APIs

Una parte importante de este proyecto será la información de las ubicaciones de las ciudades escogidas para el proyecto. Para ello se obtendrán mediante APIs de información geográfica, en este caso se utilizará OpenStreetMap para este propósito.

OpenStreetMap se trata de un proyecto abierto y colaborativo que cuenta con información geográfica recopilada por los propios usuarios para la visualización de mapas, rutas de navegación y demás.

Para este proyecto se utilizarán las siguientes APIs de este proyecto.

## Overpass

Se trata de una de las APIs que componen el proyecto de OpenStreetMap, siendo esta solo de lectura de datos. Está optimizada para las operaciones de lectura, contando con un lenguaje propio de consultas llamado *Overpass QL*,

además de introducir algunas estructuras con respecto a OpenStreetMap, como es el caso de las áreas, facilitando el acceso a la información.

## **Nominatim**

Nominatim es un motor de búsqueda que forma parte de OpenStreetMap. Sus funcionalidades están centradas en la geolocalización en base a direcciones, nombres de ubicaciones y coordenadas. Cuenta con una API propia que será de utilidad para encontrar las áreas de las distintas ciudades con las que se trabajarán en el proyecto.

---

# Aspectos relevantes del desarrollo del proyecto

---

## 5.1. Carga y almacenamiento de ubicaciones

El primer paso del desarrollo del proyecto consiste en el almacenamiento de las ubicaciones obtenidas desde un servicio de geolocalización como *Open Street Map*. La elección de este sobre otras viene motivada debido principalmente a que se trata de un proyecto completamente abierto sin ninguna restricción monetaria sobre la extracción de datos, además es mantenido por la comunidad proporcionando documentación sobre el uso y la estructura de datos que se manejan.

Para la obtención de las ubicaciones se realizarían peticiones a la API de Overpass, siendo esta perteneciente al proyecto de Open Street Map. Esta es la que se suele utilizar para labores de lectura de datos.

### Carga inicial de ubicaciones

Inicialmente se consideró el cargar las ubicaciones de distintas capitales de Europa, incluyéndose entre estas Madrid, París, Roma, Londres, Berlín y Amsterdam. Dado que Open Street Map cuenta con un sistema de etiquetado con formato *clave-valor* sobre los elementos del su modelo de datos se determinó el incluir únicamente aquellos nodos con clave «amenity». Esta se suele utilizar para cubrir diversos establecimientos públicos, servicios y negocios, por lo que se consideró adecuada para los objetivos buscados en el proyecto.

Para introducir las ubicaciones en la base de datos se optó inicialmente por usar la librería *OSMPythonTools*, que provee una interfaz sencilla con la que hacer peticiones a la API. Debido a que el proceso conllevaba cierto tiempo, se empleó una alternativa con el plugin *APOC* que permite cargar datos de la respuesta de una petición HTTP o un fichero JSON.

Una vez el proceso de carga se terminó, contábamos con un total de unos 280.000 nodos y 443 distintos valores de «amenity».

Ciudad	Número de nodos
Madrid	27.117
París	54.048
Berlín	82.900
Londres	84.558
Roma	16.675
Amsterdam	13.214

Tabla 5.1: Nodos por ciudad en la primera carga de datos

Al analizar los datos nos percatamos de que muchas «amenities» tenían fallos ortográficos o poseían un significado similar a otros. Este problema es debido a que las distintas claves de *Open Street Map* no cuentan con un diccionario predefinido de posibles valores, sino que son los usuarios que crean las ubicaciones quienes escogen el valor de estas, estén registradas anteriormente o no.

Para resolver este problema se intentaron aplicar algoritmos de distancia de cadenas con el objetivo de unificar valores de «amenity». Esta solución concluyó con que solo a partir de un 95 % de similitud se podrían juntar etiquetas con relativa precisión, aunque esta operación tendría que hacerse con supervisión para evitar posibles errores.

Otra solución que se buscó es utilizar la información de una página auxiliar de *Open Street Map* que cuenta con estadísticas de los valores de las distintas etiquetas. Se intentó unificar las etiquetas a solo aquellas consideradas como «oficiales», siendo estas las que cuentan con una página dentro de la wiki de *Open Street Map*. Esta técnica se acabó desestimando debido a que muchas de las categorías con más aparición entre las ubicaciones cargadas no contaban con página en la wiki, siendo irrelevante este factor en su relevancia.



Adicionalmente se descubrió que gran cantidad de los nodos que estaban cargados en la base de datos pertenecían a mobiliario urbano (bancos, aparcamientos, papeleras, fuentes...), por lo que se consideró su eliminación debido a que no se tenían como tan importantes en comparación con el resto de categorías. Finalmente se desestimó para evitar el caso de un posible sesgo de los datos con su eliminación. Se terminó borrando aquellos nodos que contaban con un valor de *amenity* auxiliar del modelo de datos de *Open Street Map* (Yes, No, Fixme...) debido a que no contaban con ningún significado comercial o similar.

### Enlazado de nodos

Mientras se encontraba una solución a los problemas enumerados anteriormente se decidió continuar con el enlazado de los nodos en la base de datos. Esto consistía en crear enlaces entre aquellos nodos que estuvieran en un radio de 100 metros. Para hacer esto se dotaron a los nodos con atributos tipo «Point», creados a partir de la latitud y longitud de cada una de las ubicaciones, ya que estos eran valores obligatorios en los nodos de *Open Street Map*.

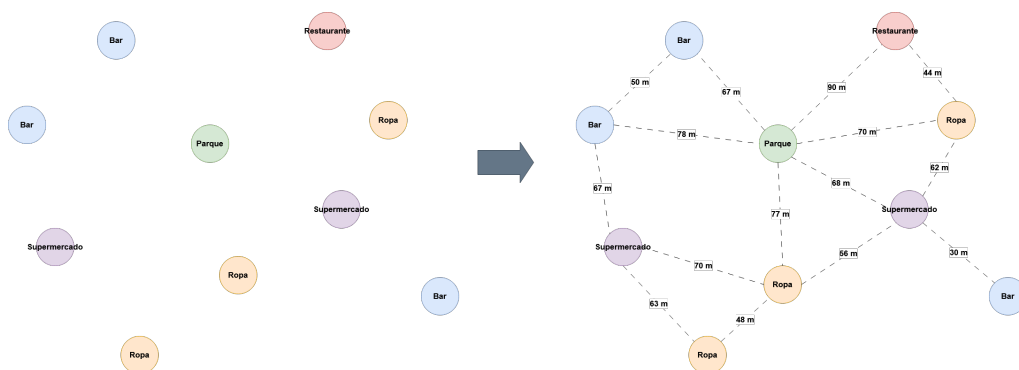


Figura 5.1: Ejemplo de creación de enlaces por proximidad

Una vez dotados a los nodos de estos atributos se crearon los enlaces entre los nodos con tal proximidad usando utilidades que el lenguaje de consultas Cypher provee para manejar distancias y coordenadas. Uno de los problemas de Neo4j es que posee es que solo permite crear enlaces unidireccionales. Si bien esto al principio resultaba problemático puesto que se creaban enlaces dobles entre cada par de nodos duplicando el número necesario de relaciones, al final se optó por crear un único enlace por par de nodos teniendo en consideración que habría que obviar la dirección en las consultas que se

hagan a la base de datos sobre las ubicaciones, reduciendo así el número de aristas del grafo.

Tras finalizar este proceso se completó para todas las ciudades, se saldó el número total de enlaces de proximidad en 5.712.246. Dados los resultados anteriores se consideró que el número de conexiones era bastante superior al esperado y se desestimó el continuar con las ciudades actuales debido al esfuerzo computacional que conllevaría su procesamiento.

## Segunda carga de nodos

Una vez vistos los resultados aportados por la elección de las ciudades anteriores se decidió utilizar otras nuevas con un tamaño inferior. Se eligieron Barcelona, Bilbao, Logroño, Oviedo, Santander, Zaragoza, Valladolid, Sevilla, Valencia y Madrid como primera aproximación, con intención de descartar alguna en caso de que se contara con un número muy elevado de ubicaciones.

Ciudad	Número de nodos
Barcelona	24.860
Madrid	27.186
Sevilla	6.323
Zaragoza	6.384
Oviedo	1.983
Valladolid	3.034
Bilbao	4.352
Valencia	8.154
Logroño	1.363
Santander	3.038

Tabla 5.2: Nodos iniciales por ciudad en la segunda carga de datos

Tras el análisis de las nuevas ciudades reducimos estas a solo 3; Sevilla, Zaragoza y Valencia, para evitar incurrir en el mismo problema que hemos visto anteriormente, centrándonos en alcanzar los objetivos propuestos para el proyecto con estas ciudades antes de incluir otras nuevas.

Una vez determinadas las ciudades con las que trabajar se procedió a realizar el enlazado de nodos como se hizo anteriormente, dándonos un número manejable de relaciones de proximidad, por lo que se continuó con la creación de las redes de interacción entre categorías.

### Interacción entre categorías

Tras haber creado la red de ubicaciones para cada ciudad el siguiente paso sería el de ver la interacción de las categorías por ciudad. Esta tomaría forma similar a una matriz de adyacencia siendo cada componente el número de veces que los nodos de una categoría poseen enlaces de proximidad con nodos de la misma u otra categoría.

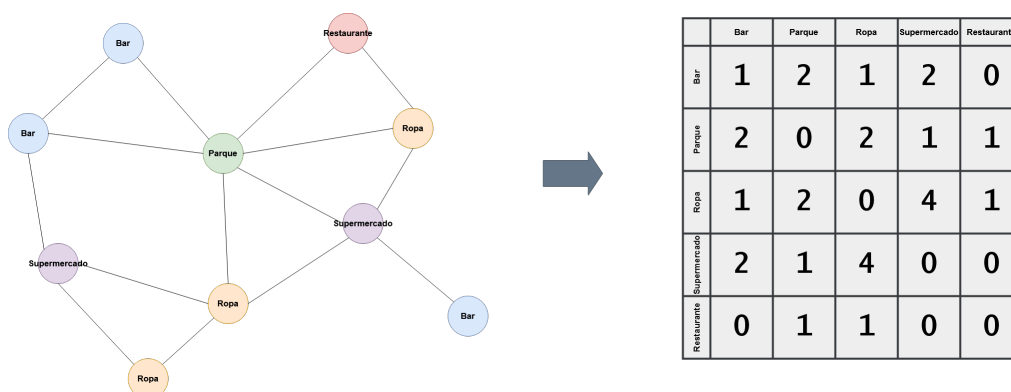


Figura 5.2: Ejemplo de interacción entre categorías

Neo4j nos permite obtener mediante consultas la información que contendrían estas matrices de categorías, solo que en una estructura tabular en lugar de matricial, por lo que habría que transformar los datos a una matriz. Un problema que se nos presenta es que en la base de datos no podemos almacenar matrices, ya que Neo4j no soporta este tipo de estructuras, además de sería complicado hacer consultas para obtener datos de esta matriz.

Se intentó buscar alguna forma en la que almacenar los datos de las categorías además de herramientas para operar con ellos en los *plugins* que Neo4j tiene, especialmente GDS (Graph Data Science) por obvias razones. Tras realizar búsquedas en la documentación de dichos *plugins*, así como hacer distintos cursos oficiales que se ofrecen de forma gratuita no se encontró ninguna utilidad que nos sirviera.

Para solventar el problema de la representación de la matriz de categorías se pensó esta como un grafo pesado con enlaces no dirigidos, forma que Neo4j sí que es capaz de manejar. Para ello se crearán nodos «Categoría» que contarán como atributos el nombre de la categoría, la ciudad a la que pertenecen y el número de nodos con los que cuenta. En cuanto a los enlaces estos serían no dirigidos (dentro de las limitaciones de Neo4j) con un atributo representativo del número de interacciones entre categorías. Este

grafo también contaría con autoenlaces para almacenar la interacción de una categoría consigo misma.

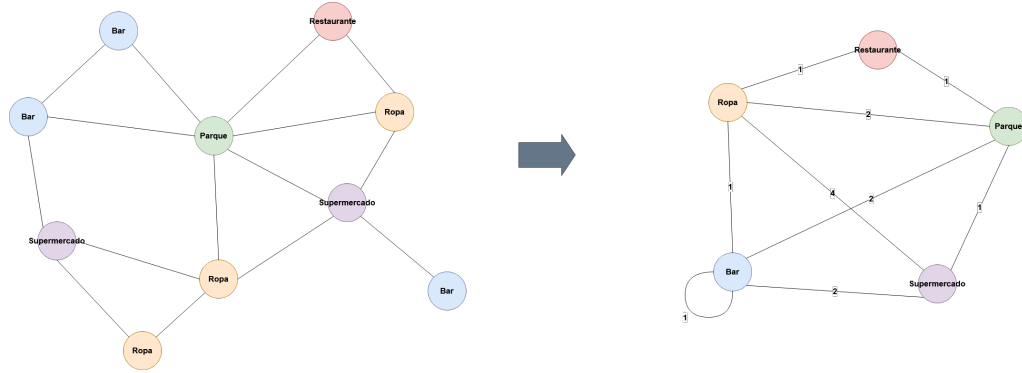


Figura 5.3: Ejemplo de red de categorías

Un problema de esta representación es que normalmente conlleva a crear grafos totalmente conectados y por tanto con un número elevado de enlaces. Pese a las desventajas que esto presenta, se tomó ventaja creando enlaces entre par de categorías aunque estas no interactúen, pero con un atributo de interacción igual a 0. De esta forma se podrán almacenar los coeficientes que nos proporcionen los distintos métodos a aplicar en un atributo del enlace pese a que no exista interacción.

### Aplicación del método *Permutation*

Una vez obtenidas las redes de categorías para cada ciudad tocaría aplicar unos de los métodos con los que obtendríamos métricas con las que hacer recomendaciones. En este caso el método *Permutation*.

Este método consiste en, sobre la red de ubicaciones de cada ciudad, mantener su estructura (nodos y enlaces) pero permutando las categorías de cada nodo. Una vez permutados se volvería a crear la red de categorías y guardarla. Esto se haría multitud de iteraciones haciendo así una simulación de Monte Carlo. Inicialmente el número de simulaciones que se realizarían sería de 10.000, pero una vez observado el tiempo que conlleva el realizar cada una se redujo a 1000. Todo lo anterior se hizo modificando la red desde Neo4j.

Otra vez más se nos planteó el problema de cómo almacenar la red de categorías de cada iteración de este método, ya que crear una nueva red con nuevos nodos y enlaces supondría una gran carga en cuestión de almacenamiento en la base de datos. Como solución se aprovechó el hecho

de que la red de categorías constituye un grafo completamente conectado, almacenando dentro de las propiedades de los enlaces una lista que contenga la interacción entre cada par de categorías en cada iteración del método.

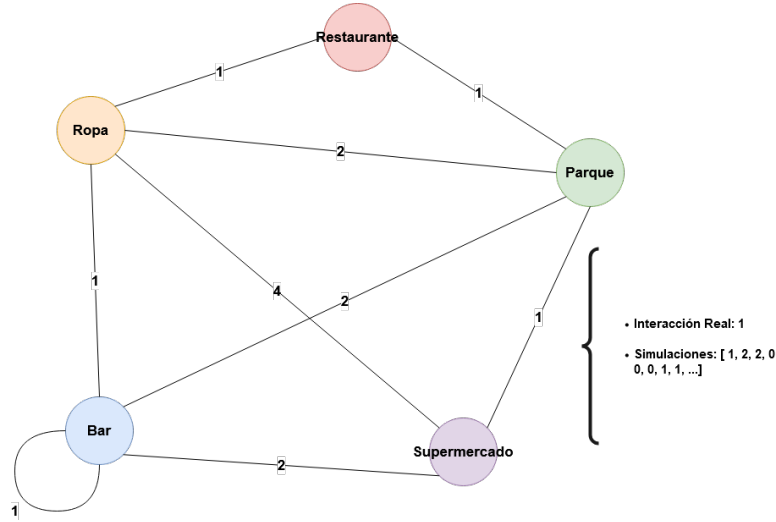


Figura 5.4: Representación de las simulaciones del método *Permutation*

### Aplicación de cálculo de coeficientes de Jensen

Dado que para obtener los coeficientes de Jensen no es necesario recurrir a simulaciones de Monte Carlo ni generación de modelos nulos, su aplicación fue más inmediata. El problema que presenta está en su almacenamiento. Dado que los coeficientes no son simétricos i.e.  $M_{AB} \neq M_{BA}$  no podemos utilizar los enlaces de anteriores métodos o de la red de interacción para almacenar estos coeficientes, ya que estos se usaban como si fuesen no dirigidos. Para solventar esto se recurrió a crear nuevos enlaces dirigidos con una etiqueta distinta para almacenar los coeficientes como atributos del propio enlaces. Para cada par de categorías contaríamos con dos enlaces con distinta dirección entre cada nodo categoría.

Si bien lo anterior funcionaba, lo cierto es que aumentaba significativamente el número de enlaces presentes en la base de datos.

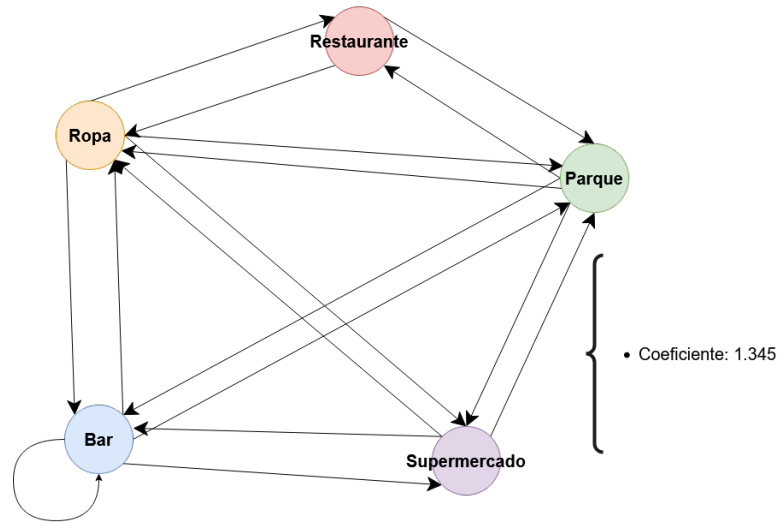


Figura 5.5: Representación de almacenamiento de coeficientes de Jensen en la base de datos

### Aplicación del método *Rewiring*

En paralelo a la realización del método de *Permutation* se pensó en una forma de realizar este método desde Neo4j a pesar de su complejidad.

El método del *Rewiring* trata de, dada la red de ubicaciones de una ciudad, conservar los nodos y su categoría, pero crear enlaces aleatorios entre nodos con la condición de que los nodos deben mantener el grado que tenían en la red original. Esto constituye una variante del *Configuration Model*, método que a partir de unos nodos y su grado crea redes aleatorias, pero con una diferencia, *Rewiring* no permite autoenlaces. Esto lejos de ser algo banal nos plantea un problema, puesto que la creación de enlaces aleatorios cumpliendo la restricción del grado de los nodos puede llevar a casos donde se requiera de autoenlaces para cumplirse. Además, al igual que en *Permutation* se tendrían que realizar 1.000 simulaciones.

Expuesto lo anterior se desestimó la posibilidad de hacer esto desde Neo4j, requiriendo de un entorno que nos aporte más flexibilidad en las operaciones como el que nos pueda proporcionar un lenguaje de programación. Se hicieron unos prototipos de *Rewiring* desde Python creando redes con distinto número de nodos y grados, así como creando pruebas para comprobar su correcto funcionamiento. Esto se haría al mismo tiempo que el otro método, priorizando más este último debido a que es más sencillo de realizar y menos costoso computacionalmente.

### Análisis de los resultados de los métodos

Tras la finalización del método *Permutation* se procedió a obtener ciertas estadísticas de los resultados de este método. En primer lugar se calcularía la media, desviación típica y Z-Scores; además de los percentiles 97.5 y 2.5 para cada interacción entre categorías. Con estas medidas podríamos obtener las relaciones significativas entre categorías, siendo estas aquellas cuyo valor real se encuentre por encima del percentil 97.5 o por debajo del 2.5 de los valores obtenidos mediante simulación.

Una vez obtenidas las relaciones más significativas estas mostraban como interacciones más importantes aquellas que incluían elementos del mobiliario urbano (Bancos, papeleras, fuentes, etc.). Lo anterior no era información que consideráramos relevante dado que este trabajo está centrado en tiendas y servicios. Esto es debido a que estos elementos están enormemente sobre-representados, llegando a ser más de un tercio de la totalidad de ubicaciones que disponíamos. Inicialmente se estimó el no eliminar estas puesto que podía conllevar un sesgo en los datos, pero dados estos resultados quedó claro que suponían un problema y por tanto debían eliminarse.

Otra conclusión que sacamos tras el análisis de los datos fue que la etiqueta «amenity» que utilizábamos para obtener las ubicaciones comerciales no contemplaba a todo el conjunto de tiendas, solo a una parte, existiendo una etiqueta propia de *Open Street Maps* para este propósito, «shop». Nos decidimos a incluir los nodos con esta etiqueta en la base de datos y volver a iniciar las simulaciones eliminando también las ubicaciones de mobiliario urbano.

Antes de comenzar el proceso anterior también se contempló la posibilidad de utilizar datos de otra API de geolocalización gratuita y dejar de usar *Open Street Map*. Entre las que se consideraron tenemos a MapBox y HereMaps con sus planes gratuitos que ofertaban, ya que no se tratan de proyectos abiertos. Después de investigar sobre su modelo de datos se abandonó la posibilidad de su uso puesto que utilizaban múltiples valores en sus etiquetas, cosa que complicaba enormemente su uso a la hora de estructurar por categorías comerciales, además estaban las limitaciones en sus planes, que podrían suponer un problema en el desarrollo del proyecto.

Tras la inclusión de los nuevos nodos y la eliminación del mobiliario urbano de la base de datos se volvió a comenzar las simulaciones del método *permutation*. Una vez estas comenzaron nos dimos cuenta que estas tardaban un tiempo significativamente mayor que con los datos anteriores, tanto que resultaba inviable proseguir con ellas. Lo anterior es debido a que

se incrementó el tiempo que conllevaba obtener la red de categorías para cada simulación i.e. la interacción entre categorías. La causa más probable es que la inclusión de las ubicaciones con etiqueta «shop» prácticamente duplicaba el total de distintas categorías con respecto a los datos anteriores, complicando el recuento de interacciones.

Una vez más se tomó como inviable el continuar con los datos que teníamos en el momento.

### Tercera carga de nodos

Dados los hechos anteriores se buscó como solución cargar nodos de ciudades más pequeñas, por lo que se utilizarían datos obtenidos de las ciudades de Castilla y León.

Una vez cargadas la ubicaciones se hizo un análisis y eliminación de los nodos pertenecientes a mobiliario urbano. El análisis mostraba que algunas ciudades tenían un número significativamente bajo del total de ubicaciones, menor que 300, por lo que se eliminaron de la base de datos al considerarse insuficientes. Al finalizar la carga y depurado de nodos contamos con las ciudades de Valladolid, Burgos, Salamanca, León y Palencia.

Ciudad	Número de nodos
Palencia	947
León	741
Salamanca	3.224
Valladolid	3.237
Burgos	3.999

Tabla 5.3: Nodos por ciudad en la tercera carga de datos

### Obtención de coeficientes

Tras la carga de datos se crearon las redes de interacción para obtener los coeficientes del método *Permutation* además de los de *Jensen*, que no se había probado en los intentos anteriores.

En cuanto a *Permutation* se obró como se hizo anteriormente, con un tiempo por simulación asumible, por lo que se obtuvieron los resultados de las simulaciones sin mayor problema. Una vez se analizaron los datos no se



detectó ninguna incoherencia con las relaciones significativas señaladas por el método, dándose los resultados como buenos.

*Jensen* suponía un problema que el método anterior no tenía. Este no requiere de simulaciones de Monte Carlo, si no que obtiene los coeficientes a partir de la red de ubicaciones original. El problema viene en que, al contrario que el método de *Permutation*, los coeficientes son diferentes para cada par de categorías según cuál se toma origen o destino. i.e.  $M_{AB} \neq M_{BA}$ .

## 5.2. Desarrollo Web

Uno de los objetivos del proyecto consiste en la creación de una aplicación web que permita visualizar los resultados de los análisis realizados sobre las ciudades que hemos escogido, así como, desde un punto de vista de usuario, poder obtener recomendaciones de categorías comerciales dadas una ubicaciones, que es el propósito último de este trabajo.

Dado que la obtención de los datos empezaba a convertirse en un lastre para el desarrollo del proyecto, se decidió hacer en paralelo la creación de la web tras la finalización de la segunda carga de ubicaciones.

Inicialmente se consideraron distintos *frameworks* web con los que crear la aplicación. Dado que el lenguaje de programación elegido para la realización del trabajo es Python tenemos a nuestra disposición *frameworks* como *Flask*, *FastAPI* y *Django*.

Finalmente se acabó por decantarse por *Flask*, esto es debido a que nos aporta una interfaz más sencilla con una curva de aprendizaje menos pronunciada que *Django*, por lo que se adecua a las necesidades del proyecto, además otro factor a considerar es que ya se tiene experiencia en él puesto que se ha utilizado en la asignatura «Diseño y Mantenimiento del Software». Pese a que no se terminó eligiendo, *Django* tiene como factor diferenciador posee un driver de *Neo4j* que incluye un OGM (*Object Graph Mapper*) que facilita el manejo de las entidades en la base de datos, aunque este no nos sería de gran ayuda en este proyecto en concreto.

Otro factor a tener en consideración es la falta de experiencia que se tiene de desarrollo web en el momento de la creación de esta aplicación. Esto es debido a que durante la carrera no se nos han impartido contenidos relacionados a esta disciplina, si bien en momento puntuales y sin hacer mucho hincapié en ellos. Esto afecta ya que el aprendizaje de las técnicas y herramientas relacionadas con el desarrollo web tendrán un peso importante durante la creación proyecto.

Para solventar la carencia de habilidades en este respecto se recurrió tanto a tutoriales como a documentación de sitios como MDN para el aprendizaje de HTML, JavaScript y CSS. Otro recurso que se utilizó fue un curso gratuito impartido por *Neo4j* en el que se enseña como construir aplicaciones web usando el driver de Neo4j para *Python* y *Flask*. Este último, si bien no enseñaba elementos transversales en el desarrollo web, mostraba cómo usar el driver (aunque en estos momentos ya se conocía su uso) y cómo estructurar un proyecto web en Flask.

### 5.3. Sistema de Recomendación

Una vez obtenidos los coeficientes de los distintos métodos la siguiente tarea consistiría en calcular los *Quality Indices*. Esto supondría que cada ubicación cargada en la base de datos tenga asociada un valor para categoría y para cada índice de calidad. Una forma de ver esto matricialmente sería una matriz de tamaño (nº de categorías) x (nº de ubicaciones) para cada ciudad. Dado que la base de datos no permite almacenar matrices, una forma equivalente de representarlo sería asociar a cada nodo un «diccionario» siendo las claves las categorías y los valores el índice de calidad, todo esto para cada índice de calidad que utilicemos.

Neo4j nos supone un problema en este aspecto ya que los nodos que utiliza funcionan como un JSON de un único nivel de profundidad, pudiendo ser las claves únicamente *strings* y los valores listas, tipos primitivos o tipos especiales (fechas, puntos de coordenadas...). Esto nos perjudica ya que no sería posible almacenar de una forma simple los índices de calidad.

Dado lo anterior, se decidió cambiar la aproximación que iba a tomar el proyecto. Si inicialmente se buscaba encontrar la categoría más adecuada para unas ubicaciones, ahora sería, dadas unas ubicaciones encontrar cuál es la más adecuada para una categoría determinada. Esto haría que en lugar de almacenar los índices de calidad estos se calcularían «al vuelo», bajo demanda. Lo anterior viene también motivado por los tiempos de cálculo que, haciéndose desde Neo4j conllevaban un tiempo de unos 3 minutos por nodo. Teniendo en cuenta que contamos con un número de aproximadamente 12.000 nodos resultaría inviable.

Esta forma implicaría que no sería posible el uso combinado de índices de calidad mediante un modelo de inteligencia artificial para realizar recomendaciones. No tendríamos un conjunto de datos con el que poder entrenar y evaluar la precisión del modelo.

Cambiada la aproximación, se prosiguió con los correspondientes cambios en la web para adaptar la funcionalidad.

Tras un tiempo después centrándose en desarrollo web se encontró una posible solución. Para resolver el problema que suponía el almacenamiento, los índices de calidad se almacenarían como un JSON formateado como *string*. Esto pese a parecer una solución pobre nos solventa los problemas que teníamos anteriormente, pudiendo convertir este *string* de vuelta a JSON desde Python y así obtener los valores que buscábamos. Además dentro del plugin APOC de Neo4j, se incluyen procedimientos que permiten para hacer estas conversiones, ya que, aunque Neo4j no permita almacenar «maps» o «diccionarios» en sus nodos, permite utilizarlos en su lenguaje de consultas, Cypher.

En cuanto al problema que suponía el tiempo de computo de los índices de calidad se probó a hacer los cálculos en Python en lugar de Neo4j. Para ello se construirían las matrices de coeficientes de calidad y de promedio de vecinos por categorías mediante consultas a la base de datos y almacenándolas en diccionarios de Python, consiguiendo así un tiempo de acceso constante. Uno de los problemas que presenta esta forma es que los cálculos se deben de hacer nodo a nodo en lugar de hacer actualizaciones masivas cómo se podría hacer desde Neo4j. Para cada nodo además de la información contenida en las matrices de promedio de vecinos por categorías y de coeficientes de calidad se tendría que contar con el número de interacciones con las categorías, cosa que se haría también mediante consultas a la base de datos. Una vez hechos los cálculos de los índices de calidad estos se agruparían en un JSON formateado como *string* y se añadirían a la información del nodo en la base de datos.

Una vez completada la alternativa, esta mostraba buenos resultados temporales, llevando alrededor de 0,1 segundos por nodo. A la vista de los resultados se prosiguió con esta forma. Con esto contábamos con los índices de calidad calculados, permitiéndonos contar con un conjunto de datos sobre el que aplicar un modelo de clasificación.

## Random Forest

Obtenidos los índices de calidad para cada una de la ubicaciones contamos con un conjunto de datos con los que poder entrenar un modelo y hacer recomendaciones en base a ubicación.

La elección de *Random Forest* sobre otros modelos de inteligencia artificial viene motivada por el buen rendimiento que presentan los modelos de tipo

*ensemble* sobre otros simples. Estos están compuesto por un conjunto de clasificadores en lugar de solo uno, permitiendo reducir el sobre ajuste asegurando un buen rendimiento del clasificador.

### Recomendaciones Locales

Una parte de las recomendaciones que se harían serían utilizando únicamente datos locales, es decir, de la propia ciudad. Esto implica crear un modelo de *Random Forest* para cada ciudad. Las posibles categorías que se pueden recomendar serían exclusivamente aquellas que la propia ciudad posee.

Para poder evaluar el rendimiento real de los modelos se aplicará una técnica común en el ámbito del *machine learning* conocida como validación cruzada. En nuestro caso realizaremos 100 *folds* para los modelos de cada ciudad.

Tras realizar la validación cruzada se nos arrojan los siguientes resultados:

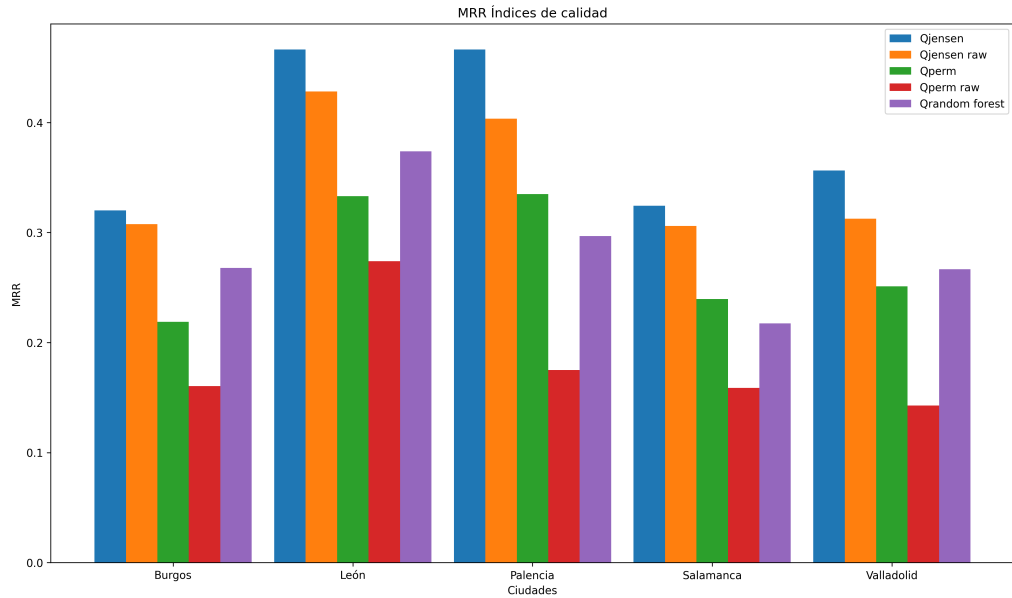


Figura 5.6: *Mean Reciprocal Rank* en las predicciones a nivel local

Como se puede observar, en comparación al resto de índices de calidad los modelos de *Random Forest* no logran superar a todos, viéndose superados en la mayoría de casos por *QJensen* y *QJensen\_raw* que parecen ser los que mejor rendimiento ofrecen. Pese a lo anterior, sí que consigue obtener resultados considerables.

Otra conclusión que se puede extraer de los resultados es el impacto del tamaño de la ciudad a la hora de obtener buenas predicciones. León y Palencia son las ciudades que consiguen obtener un MRR más alto que el resto siendo estas dos las regiones más pequeñas, en cuanto a nivel de nodos, de las que contamos.

### Transferencia

A la hora de realizar transferencia de información mediante la utilización de un *Random Forest* se nos presenta un problema con respecto al conjunto de datos a utilizar. Este problema tiene origen en el hecho de que las ciudades tienen conjuntos distintos de categorías, teniendo un total de unas 250 en el conjunto de datos. Esto supone que las ubicaciones de una ciudad no van a tener índices de calidad asociados a categorías que no están presentes en dicha ciudad. Además la diferencia de categorías entre ciudades en algunos casos roza las 100, por lo que de hacerse de esta forma habría muchas ubicaciones con gran parte de índices de calidad desconocidos.

Otro problema derivado que supondría el utilizar toda la información de todas las ubicaciones que disponemos sería el tamaño del conjunto de datos con el que trabajaríamos. De hacerlo así cada ubicación tendría asociados valores para cada índice de calidad y para cada ubicación, teniendo así unos 1.000 atributos con 12.000 ubicaciones. Esto supone un conjunto de datos bastante grande, especialmente si utilizamos técnicas como la validación cruzada para validar el modelo.

Finalmente se decidió por crear distintos modelos de *Random Forest* para en las que apliquemos transferencia. Para ello se utilizaría una ciudad para entrenar (*source*) el modelo y otra sobre la que realizar las predicciones (*target*). Esto supondría tener 20 modelos distintos, puesto que contamos con 5 ciudades y descontamos los modelos que se utilizan para recomendaciones locales. A la hora de obtener el conjunto de datos de cada modelo se utilizarían únicamente las categorías pertenecientes a la intersección entre las categorías de ese par de ciudades, teniendo a su vez que eliminar filas con categoría fuera de esta, así como filtrar columnas con datos pertenecientes a índices de calidad asociados a categorías que no pertenecen a la intersección.

Tras realizar los entrenamientos y predicciones de los distintos modelos, estos nos arrojan los siguientes resultados:

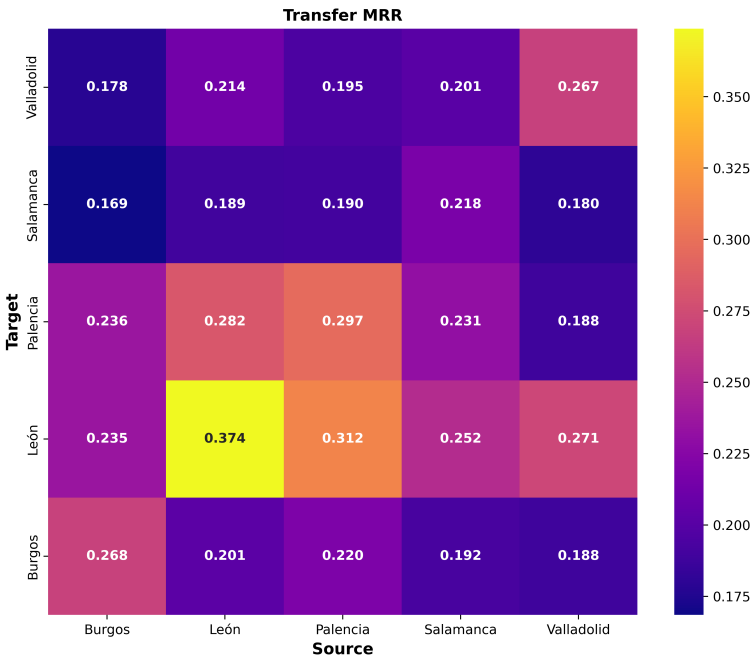


Figura 5.7: Mean Reciprocal Rank al aplicar transferencia con *Random Forest*

---

## Trabajos relacionados

---

### 6.1. *Knowledge Transfer in Commercial Feature Extraction for the Retail Store Location Problem*

Este artículo es la mayor inspiración detrás de este trabajo, puesto que tanto las técnicas que se han aplicado como la aproximación provienen de este.

Al igual que en este artículo, se ha trabajado con ciudades de Castilla y León, aunque no con todas, pese a que no era la intención inicial esta elección de ciudades.

Las principales diferencias con respecto lo realizado en este trabajo consisten en el origen de los datos y las categorías utilizadas. En el artículo los establecimientos comerciales son extraídos desde las Páginas Amarillas de 2017 y posteriormente geolocalizadas mediante APIs. En cuanto a categorías se utilizó 68 posibles valores definidos por *North American Industry Classification for Small business* (NAICS).

Aplicaron los 3 métodos expuestos anteriormente, incluido el *Rewiring* que no se pudo realizar en este trabajo, además de combinarlos mediante *Random Forest*. Obtuvieron los siguientes resultados a nivel local:

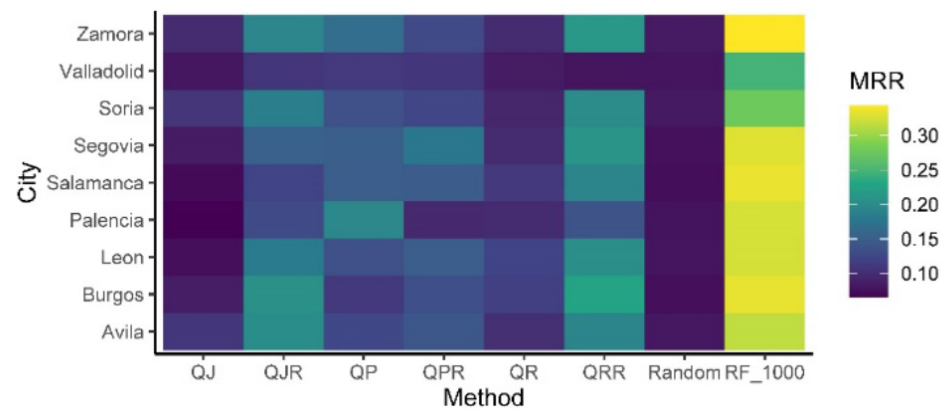


Figura 6.1: *Mean Reciprocal Rank* por método y ciudad

En cuanto a la transferencia puesto que contaban con más ciudades, podían utilizar más combinaciones para probar su rendimiento.

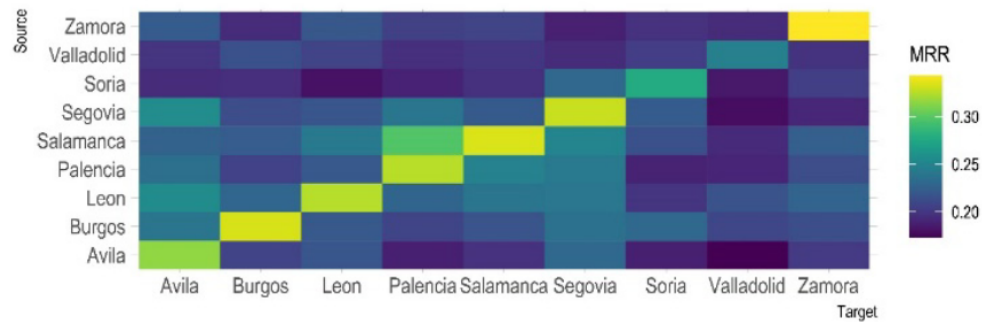


Figura 6.2: *Mean Reciprocal Rank* al utilizar transferencia

En contraste con lo realizado con este trabajo, parece que la inclusión de nuevas ubicaciones pertenecientes a categorías no contempladas por el NAICS ayuda a obtener mejores resultados.



## 6.2. *Retail Store Location Selection*

### *Problem with Multiple Analytical Hierarchy Process of Decision Making an Application in Turkey*

Este artículo es un ejemplo de otra aproximación al problema de selección de ubicación distinta a la ciencia de redes. En este emplean lo llamado *Analytical Hierarchy Process* (AHP). Este consiste en definir distintos componentes tanto objetivos como subjetivos y hacer una decisión multicriterio en base a estos. Define 15 criterios agrupados en 5 categorías:

- (M) Costes
  - **M1**: Coste del alquiler.
  - **M2**: Coste del mobiliario.
  - **M3**: Tiempos y condiciones de contratación.
- (R) Competencia
  - **R1**: Poder de la competencia.
  - **R2**: Número de competidores.
  - **R3**: Distancia a la competencia.
- (T) Densidad de tráfico
  - **T1**: Tráfico de vehículos.
  - **T2**: Tráfico de viajeros.
- (F) Característica físicas
  - **F1**: Tamaño de la tienda.
  - **F2**: Aparcamientos.
  - **F3**: Visibilidad.
- (Y) Localización
  - **Y1**: Sobre calle principal.
  - **Y2**: En centro comercial.
  - **Y3**: Cercano a centros de negocio.

- **Y4:** Cercano a áreas residenciales y sociales.

Si bien este artículo buscaba encontrar las mejores ubicaciones para un determinado negocio en 3 asentamientos, y por tanto, tomando una aproximación distinta a lo realizado en este proyecto, se puede apreciar la gran cantidad de información necesaria para poder estimar la mejor ubicación. Este problema no se presenta en nuestro proyecto, puesto que contando únicamente con las ubicaciones comerciales es capaz de inferir la adecuación de una localización a varias categorías comerciales, no solamente a una.

---

# Conclusiones y Líneas de trabajo futuras

---

## 7.1. Conclusiones

En este proyecto se ha conseguido realizar un sistema de recomendación de categorías que puede ser de gran utilidad para todas aquellas personas que estén pensando en abrir un negocio, o simplemente para analizar la estructura comercial de las distintas ciudades.

Durante el desarrollo han surgido bastantes dificultades que no se preveían inicialmente que han obligado tomar distintas decisiones para llevar el proyecto a cabo. Muchas de estas están relacionadas con las herramientas utilizadas. Si bien el uso de estas nos ha llevado a problemas, cabe destacar que son las mejores opciones disponibles. Desarrollaré en los apartados sucesivos al respecto.

### Sobre la base de datos

El uso de Neo4j vino principalmente motivado porque fue una de las herramientas propuestas para el proyecto. Tratándose de una base de datos orientada a grafos parecía ser apropiada para el desarrollo del trabajo, puesto que se han utilizado conceptos y estrategias propias de la ciencia de redes.

Desde el comienzo del proyecto se instó a usar lo máximo posible Neo4j para la mayoría de las etapas del proyecto con objetivo de aprovechar las funcionalidades que nos provee así como para justificar su uso de cara al trabajo.

Si bien es cierto que Neo4j hace bastante cómoda la obtención de datos de un grafo, no es tan flexible en otros aspectos. Esto se ha visto en los momentos en los que se buscaba aplicar los métodos de *Rewiring* y *Permutation*, así como el almacenamiento de los coeficientes que estos nos proporcionaban.

Cuando se requiere hacer una consulta más allá de lo más básico que se pueda hacer con *Cypher*, el lenguaje de consultas de Neo4j, inevitablemente se tendrá que recurrir a procedimientos del plugin APOC. Esta característica hace que las consultas se vuelvan bastante tediosas y complejas, con varias llamadas a procedimientos de este plugin, dificultando su uso. Las capacidades que nos provee *Cypher* nativo están bastante limitadas sin el uso de este plugin.

Otro problema que plantea la base de datos es el modelo de datos que emplea. Neo4j no cuenta con la flexibilidad que otras bases de datos no relacionales tienen en este aspecto. Esto puede verse cuando se busca almacenar datos anidados en alguna de las entidades (nodos o relaciones), que no pueden almacenarse como una simple propiedad de estas sino que tienen que serializarse de alguna forma (cómo se ha hecho en este trabajo) o recurrir a crear una nueva entidad relacionada que almacene dichos datos pese a que estos no tengan una relevancia suficiente como para considerarse como nodos o relaciones, además de complicar las consultas que pudieran hacerse al respecto.

En cuanto a la realización de cálculos complejos también se ha visto que no es recomendable su uso más allá de las operaciones que estamos acostumbrados en otras bases de datos, puesto que tras intentar hacerlos se ha comprobado que son mucho más lentas de lo esperado. Aunque lo anterior puede que sea consecuencia de la complejidad de las consultas como se comentó anteriormente. Además en general estas operaciones son realmente difíciles de hacer para un usuario que no cuente con bastante experiencia anterior con esta base de datos.

Sobre conceptos propios de la ciencia de redes tampoco aporta mucha ayuda en ver las entidades de la base de datos cómo se haría desde esta disciplina, puesto que no aporta algunas representaciones utilizadas habitualmente como matrices o el no soporte a relaciones no dirigidas. Además resulta complicado trabajar con datos derivados de grafos para realizar operaciones o cálculos sobre el grafo, especialmente si estos constituyen una entidad propia como se ha hecho en este trabajo con las matrices de interacción y relacionados.

Habiendo expuesto lo anterior, parece que Neo4j no termina de comprometerse totalmente con ninguno de los conceptos en los que se basa,

las bases de datos no relacionales y la ciencia de redes. No cuenta con la flexibilidad de otras bases de datos ni aporta algunos elementos propios de la ciencia de redes. Si alguien fuese a valorarse su uso, como opinión personal diría que debería verse como más próximo a las bases de datos no relacionales que a la ciencia de redes. Tras haber trabajado con ella parece más apropiada para otros contextos, como redes sociales y similares, en lugar de la temática de este trabajo. Además la amplia mayoría de funcionalidades que ofrece el *plugin Graph Data Science* están relacionadas con aspectos más convencionales de la ciencia de redes, como detección de comunidades, *Random Walk*, medidas de centralidad y demás.

Pese a lo que se ha dicho, probablemente no exista una herramienta más adecuada para las labores realizadas en este proyecto que esta. También cabe mencionar que quizás se tenían unas expectativas no realistas con respecto a Neo4j. Si bien nos permite almacenar un grafo de forma persistente y operar con el directamente creo que no es competencia de una base de datos la realización de unas operaciones tan complejas como las que se han realizado, teniendo en su lugar que realizarse desde un lenguaje de programación que nos aporte una versatilidad que un lenguaje de consultas no es capaz de dar. Lo anterior sería lo ideal en caso de no contar con limitaciones como la capacidad de la memoria para alojar el grafo o similares, aunque tiene el impedimento de tener que construir el grafo mediante consultas, proceso que lleva tiempo, además de no poder operar directamente sobre el grafo en casos que se requiera hacer modificaciones o borrados ni obtener información tan fácilmente como se hace desde Neo4j mediante consultas de *Cypher*.

## Sobre *OpenStreetMap*

*OpenStreetMap* también se trataba de una de las herramientas propuestas. Si bien la obtención de datos no ha supuesto un problema de por sí ya que al tratarse de un proyecto abierto no ha habido barreras monetarias al respecto, y contar con un lenguaje de consultas propio para la obtención de datos; sí que lo ha habido sobre el contenido de las ubicaciones y su cantidad.

Uno de los problemas que nos ha presentado los datos son los valores de sus etiquetas de categorías. Cómo se ha mencionado en uno de los apartados del trabajo, no existe un diccionario de valores, sino que los usuarios son quienes los asignan. Esto conlleva a que existan gran cantidad de categorías, algunas con múltiples valores, faltas ortográficas y demás que han dificultado ciertas fases del proyecto.

Otro punto a destacar es la diferencia de número de ubicaciones en distintas ciudades, esto lejos de adecuarse a la realidad creo que viene determinado por la falta de cobertura de algunas zonas por parte del proyecto, además de la falta de puesta al día de la información de ciertas ubicaciones.

*Overpass QL*, el lenguaje de consultas de esta API, si bien facilita la obtención de datos presenta problemas al obtener datos de ciertas áreas si existen más de una con ese mismo nombre. Tampoco pueden hacerse búsquedas de áreas dentro de áreas, cosa que es problemática al querer operar con determinadas zonas. En este caso planteó problemas al obtener datos de las ciudades finales del proyecto puesto que había más ciudades con ese nombre, por lo que se tuvo que recurrir al número de municipio para obtener sus nodos.

Expuesto lo anterior concluir que *OpenStreetMap* supone una buena herramienta abierta para la obtención de datos, no sin sus inconvenientes, que probablemente si se hubiese utilizado una API de pago no se presentarían además de probablemente contar con una mejor calidad de datos.

## 7.2. Líneas de trabajo futuras

Debido a la gran cantidad de imprevistos que nos hemos encontrado durante proyecto hay apartados que han acabado con un desarrollo menor a lo esperado inicialmente. Además con los descubrimientos realizados podemos hacer algunas recomendaciones para proyectos similares a este o que busquen continuarlo.

Sobre las categorías de ubicaciones que se han utilizado, tras su empleo se ha llegado a la conclusión de que contar con un número elevado de categorías aumenta exponencialmente el esfuerzo computacional que conlleva la obtención de índices de calidad, así como las recomendaciones, además de contar con algunas que pueden ser redundantes o no relevantes. Lo más adecuado quizás sería emplear alguna forma de normalización de estas que nos permita transformarlas para tener un número más reducido de categorías aunque contemple todos los posibles valores, facilitando la obtención de métricas de calidad. Esto, sin embargo, debería hacerse con cuidado. Si se reducen las categorías a un número muy pequeño puede empeorar la calidad de los datos que disponemos, puesto que puede que no se capture adecuadamente la diversidad que hay entre categorías, agrupando varias con diferencias importantes entre sí, empeorando también las posibles recomendaciones que se pudieran hacer sobre ellas. Por el otro lado, si se reduce mínimamente las

categorías se seguiría teniendo el mismo problema presentado anteriormente, siendo únicamente algo menos grave.

En cuanto a la calidad de los datos que nos ofrece *OpenStreetMap*, como se ha observado en apartados anteriores, presentan problemas sobre los valores de categorías así como la irregularidad en el número de ubicaciones sobre algunas ciudades. Dado lo anterior, sería buena idea emplear APIs de pago que no se han podido utilizar como *Google Maps*, a las que presuponemos una gran calidad de datos. Un punto a tener en consideración en la elección de estas sería el modelo de datos que emplean, especialmente el cómo estructuran las categorías comerciales, puesto que algunas emplean una lista de valores en lugar de un único valor; esto imposibilitaría la aplicación de los métodos de recomendación de este trabajo. De presentarse esto, podría aplicarse la normalización de categorías que se propone en el párrafo anterior para que el uso de los datos fuera factible.

La aplicación web no cuenta con muchas funcionalidades que se preveían inicialmente debido a los problemas presentados durante el desarrollo. Si bien están definidos roles, ninguno cuenta con permisos o utilidades especiales. Sería de bastante utilidad proveer a los administradores de una herramienta que les permita cargar datos de otras ciudades así como poder examinarlos y realizar los borrados y modificaciones que crean convenientes. Esto requeriría alojar la aplicación web en un sitio con la suficiente capacidad computacional como para aplicar los métodos realizados en este trabajo con objetivo de disponer de nuevas ciudades sobre las que realizar recomendaciones. En cuanto a los usuarios, se les podría dotar de la capacidad de guardar las ubicaciones sobre las que estén interesados, pudiendo acceder a ellas desde su perfil y realizar comparaciones sobre ellas.

En relación a los modelos utilizados para el sistema de recomendación solo se ha empleado *Random Forest*, por lo que podría incluir nuevos modelos que el usuario pueda elegir con cuales realizar las recomendaciones.

El único método de obtención de métricas de calidad que ha faltado por implementar ha sido *Rewiring*. Debido a los problemas que presenta este método no se utilizó en este proyecto, por lo que podría emplearse para desarrollos posteriores. Dada la complejidad que presenta y las dificultades de Neo4j para operar sobre grafos lo considero prácticamente irrealizable desde este, por lo que debería realizarse desde un lenguaje de programación.





---

## Bibliografía

---

- [1] Virginia Ahedo, Jose Ignacio Santos, and Jose Manuel Galan. Knowledge transfer in commercial feature extraction for the retail store location problem. *IEEE Access*, 9:132967–132979, 2021.
- [2] Rosa María Sánchez-Saiz, Virginia Ahedo, José Ignacio Santos, Sergio Gómez, and José Manuel Galán. Identification of robust retailing location patterns with complex network approaches. *Complex and Intelligent Systems*, 8:83–106, 2 2022.
- [3] Wikipedia contributors. Graph theory — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/w/index.php?title=Graph\\_theory&oldid=1171835383](https://en.wikipedia.org/w/index.php?title=Graph_theory&oldid=1171835383), 2023. [Online; accessed 28-August-2023].