INFORME

PROYECTO FINAL

BBDD

Javier Mendoza Guerrero

Bertrán Vidal Campos

ÍNDICE

Javier Mendoza Guerrero
Bertrán Vidal Campos

1. INTRODUCCIÓN: OBJETIVO DEL PROYECTO

Este proyecto tiene como objetivo principal el diseño, implementación y análisis comparativo de un sistema de almacenamiento y visualización de datos utilizando tanto bases de datos relacionales (MySQL) como no relacionales (MongoDB). Para ello, se ha trabajado con varios conjuntos de datos de reseñas de productos de Amazon, concretamente de las categorías *Toys and Games*, *Video Games*, *Digital Music* y *Musical Instruments*, todos ellos con estructura similar en formato JSON.

El trabajo se ha dividido en varias partes. En primer lugar, se ha realizado un esquema relacional y no relacional para diferenciar los datos que se almacenan en MySQL y en MongoDB, en función de su naturaleza estructurada o no estructurada. Posteriormente, se ha realizado la carga de datos a través de scripts de Python, realizando conexiones a las correspondientes aplicaciones.

A continuación, se ha implementado una aplicación para el acceso y visualización de los datos, que permite generar distintas gráficas relevantes a partir de la información almacenada. Para ello, se ha optado por una interfaz visual basada en Dash, mediante la cual se ha desarrollado un dashboard interactivo.

Además, se ha incorporado el uso de la base de datos de grafos Neo4J, en la que se modelan relaciones entre usuarios y productos para realizar análisis más avanzados, como la similitud entre usuarios o la identificación de comunidades de consumo. Finalmente, se ha explorado la aplicabilidad de técnicas de visualización con herramientas externas (como Tableau o Power BI) y se ha propuesto un posible enfoque de sistema de recomendación basado en aprendizaje automático (MLP), como ejemplo de integración de modelos de machine learning con los datos almacenados.

En conjunto, este proyecto combina técnicas y conceptos estudiados en la asignatura, como el diseño de esquemas, carga de datos, consultas en bases SQL y NoSQL y conceptos de otras asignaturas de la carrea como Visualización de Datos y Machine Learning.

2. DISEÑO Y CARGA DE DATOS

En primer lugar, se decidió qué parte de la siguiente información se almacenaría en MySQL y cuál en MongoDB. Contábamos con reseñas provenientes de las siguientes cuatro fuentes de información.

Tipo del producto	Nombre del fichero
toys	Toys_and_Games_5.json
video_ games	Video_Games_5.json
music	Digital_Music_5.json
instruments	Musical_Instruments_5.

Figura 1. Tabla de ficheros de datos

Todas las reseñas compartían la misma estructura:

```
{
"reviewerID": "A2HD75EMZR8QLN",  # Es el autor de la review
"asin": "0700099867",            # id del artículo
"reviewerName": "123",           # nombre del reviewer
"helpful": [8, 12],              # array conteniendo la utilidad
"reviewText": "texto",           # texto de la review
"overall": 5,                    # Nota
"summary": "texto",              # resumen de la review
"unixReviewTime": 1341792000,    # revisión unix (timestamp)
"reviewTime": "07 9, 2012"       # fecha de la review
}
```

Figura 2. Estructura proporcionada de una review

En primer lugar, cabe destacar que se ha creado un nuevo ID manual para cada review distinta. Este ID se comparte entre MySQL y MongoDB, lo que facilita futuras consultas que involucren ambos sistemas de gestión de bases de datos.

El criterio seguido para decidir qué información almacenar en la base de datos no relacional (MongoDB) se ha basado principalmente en incluir elementos sin una estructura rígida o de gran longitud, que no son fácilmente soportados por el sistema de datos estructurados como MySQL. En particular, se han almacenado en MongoDB los campos reviewText y summary.

Además, se consideró más eficiente almacenar directamente en MongoDB la lista helpful, en lugar de dividirla en dos elementos para adaptarla a MySQL. Esta decisión puede reducir el coste computacional y la complejidad del código al momento de cargar

los datos. En MongoDB se puede acceder a cada elemento de la lista individualmente si es necesario, aunque se almacenan en conjunto.

También es importante mencionar que, aunque contamos con cuatro tipos de productos, todos comparten la misma estructura. Dado que un mismo usuario puede haber valorado productos de diferentes categorías, se decidió almacenar todas las reviews en una única colección de MongoDB, con el fin de facilitar futuras consultas que involucren múltiples tipos de productos.

El resto de los campos —reviewerID, asin, reviewerName, overall, unixReviewTime, y reviewTime— se almacenan en MySQL, siguiendo el siguiente esquema relacional:

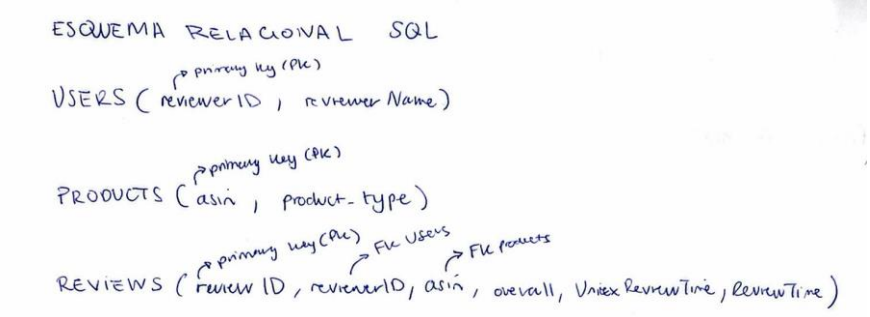

Figura 3. Esquema Relacional MySQL

Es importante destacar que, para cumplir completamente con la Tercera Forma Normal (3FN), se podría haber creado una cuarta tabla llamada REVIEWTIME(unixReviewTime, reviewTime). Esto se debe a que el campo reviewTime puede derivarse directamente de unixReviewTime, por lo que mantener ambos en la misma tabla introduce una dependencia transitiva en la tabla Reviews.

Al crear esta cuarta tabla, unixReviewTime actuaría como clave primaria, y en la tabla REVIEWS, unixReviewTime pasaría a ser clave foránea que referencia a REVIEWTIME.

Inicialmente, diseñamos la base de datos con estas cuatro tablas. Sin embargo, tras realizar varias consultas, observamos que el coste computacional derivado de los JOINs entre las tablas superaba el beneficio teórico de respetar al máximo la 3FN. Por ello, decidimos simplificar el diseño y aceptar una ligera desviación de la normalización en favor de una mayor eficiencia en las consultas.

La carga de datos se realiza a través del archivo load_data.py, el cual utiliza las constantes definidas en el archivo configuración.py para establecer la conexión con MySQL y MongoDB mediante Python. Este script se encarga de cargar los cuatro ficheros especificados en la tabla de la Figura 1.

La siguiente imagen muestra un fragmento del código contenido en load_data.py, donde se crean las tablas definidas en el modelo relacional.

```python
def crear_tablas(cursor):
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Users (
            reviewerID VARCHAR(50) PRIMARY KEY,
            reviewerName VARCHAR(255)
        )
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Products (
            asin VARCHAR(50) PRIMARY KEY,
            product_type VARCHAR(50)
        )
    """)


    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Reviews (
            reviewID INT PRIMARY KEY,
            reviewerID VARCHAR(50),
            asin VARCHAR(50),
            overall INT,
            unixReviewTime INT,
            reviewTime DATE,
            FOREIGN KEY (reviewerID) REFERENCES Users(reviewerID),
            FOREIGN KEY (asin) REFERENCES Products(asin)
        )
    """)
```

Figura 4. Creación de las tablas MySQL en Python

3. ACCESO Y VISUALIZACIÓN DE LOS DATOS

Para la visualización de los datos, se diseñó un dashboard interactivo utilizando la librería Dash. El proceso comenzó con la implementación de las consultas necesarias para extraer la información desde MySQL y MongoDB, que se estructuraron en un único archivo conforme a las indicaciones del proyecto. A partir de estos datos, se generaron distintas gráficas con Plotly, incluyendo histogramas, evoluciones temporales y nubes de palabras. Finalmente, se organizó el layout del dashboard en pestañas temáticas, aplicando estilos personalizados en CSS para ofrecer una interfaz clara, profesional y fácil de utilizar.

Figura 5. Pestañas dashboard interactivo

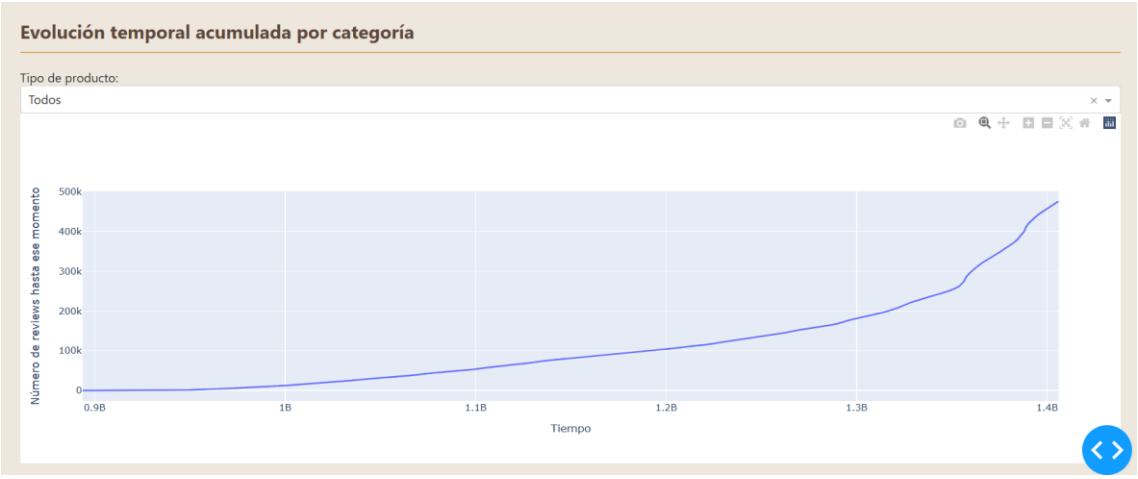

Figura 6. Evolución temporal de reviews filtrada por tipo de producto

Javier Mendoza Guerrero
Bertrán Vidal Campos

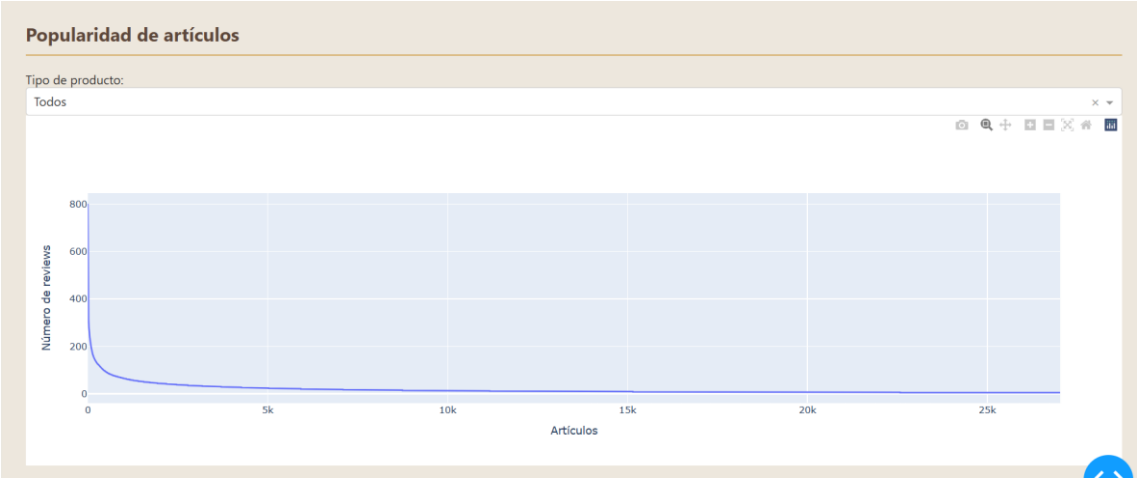

Figura 7. Popularidad de artículos filtrada por tipo de producto

Figura 8. Histograma por nota filtrado por tipo de producto y producto específico

Javier Mendoza Guerrero
Bertrán Vidal Campos

Figura 9. Valoración media filtrada por año

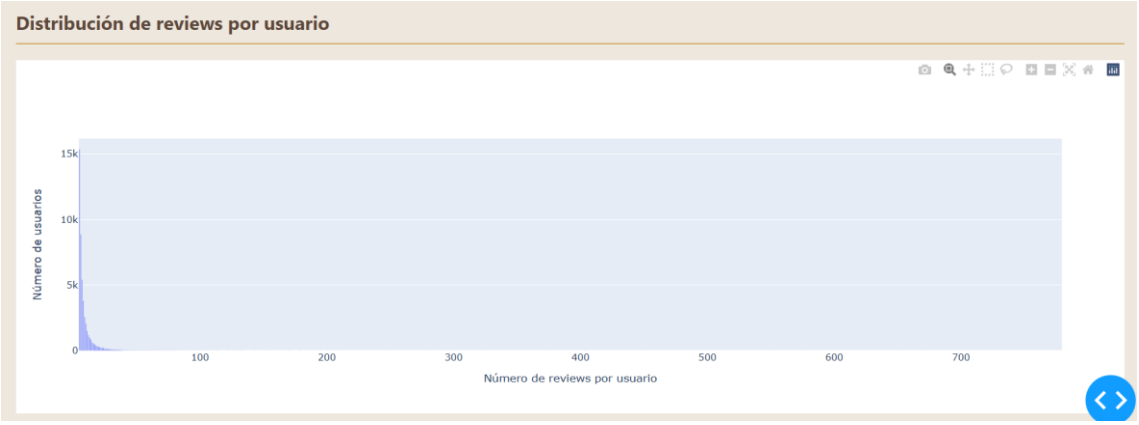

Figura 10. Distribución de reviews por usuario

Javier Mendoza Guerrero
Bertrán Vidal Campos

Figura 11. Nube de palabras filtrada por tipo de producto

Javier Mendoza Guerrero
Bertrán Vidal Campos

4. APLICACIÓN PYTHON JUNTO A NEO4J

En este apartado se crea el fichero neo4JProyecto.py, donde se definen las funcionalidades necesarias para analizar la base de datos de reviews guardada en MySQL, pero principalmente con el objetivo de visualizar los resultados en Neo4J.

Además, en cada uno de los subapartados de esta sección, para asegurar la coherencia y evitar conflictos con funcionalidades anteriores, se parte de una base de datos vacía en Neo4J, eliminando todos los nodos previamente creados antes de insertar los nuevos datos.

4.1. Similitudes entre usuarios y visualización de los enlaces en Neo4J

En primer lugar, se ha implementado una funcionalidad que, mediante una consulta a la base de datos creada en el apartado 1 (MySQL), permite obtener los n usuarios con mayor número de reviews (por defecto 30, aunque este valor es fácilmente configurable). A partir de esta lista, se calcula la similitud entre cada par de usuarios utilizando la correlación de Pearson, implementada manualmente a partir de la fórmula proporcionada, sin recurrir a librerías externas.

Solo se consideran pares de usuarios que hayan puntuado al menos un producto en común. Las similitudes obtenidas se almacenan y posteriormente se cargan en Neo4J, donde cada usuario se representa como un nodo y cada relación bidireccional indica el nivel de similitud entre dos usuarios.

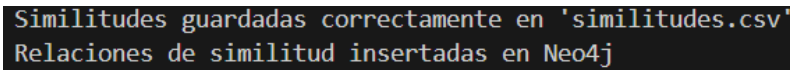

Figura 12. Output al insertar en Neo4J

Finalmente, se incluye una consulta en Neo4J para identificar el usuario con mayor número de conexiones (vecinos).

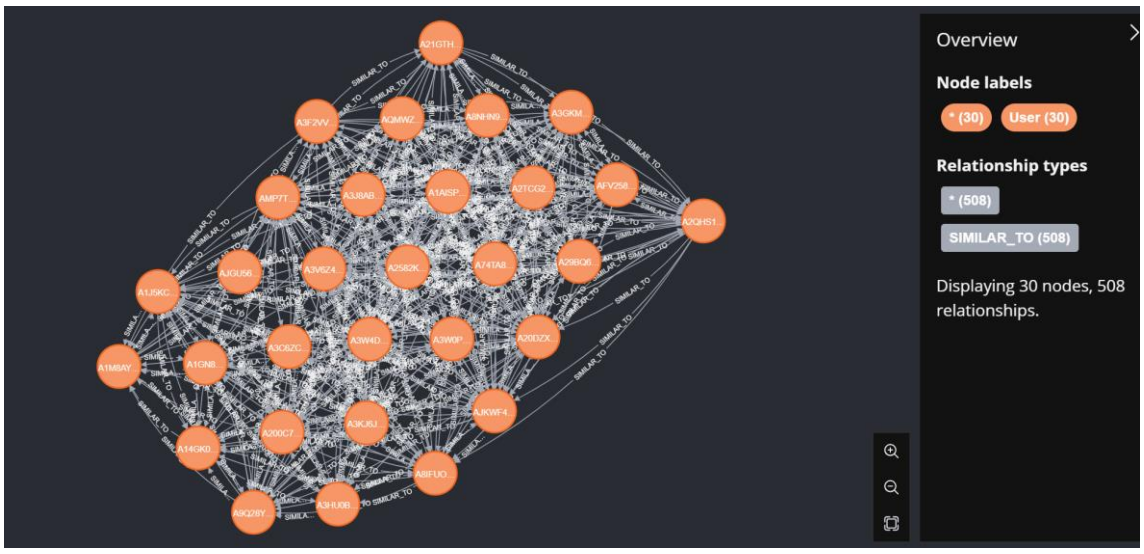

Figura 13. Grafo en Neo4J de Similitudes entre top30 Usuarios

4.2. Enlaces entre usuarios y artículos

En el archivo neo4JProyecto.py también se ha añadido una funcionalidad adicional que permite seleccionar un número determinado de artículos aleatorios de un tipo específico (por ejemplo, Videojuegos, Discos, etc.), que el usuario debe indicar por pantalla junto con la cantidad de artículos a seleccionar.

A partir de estos artículos seleccionados aleatoriamente, se visualizan en Neo4J tanto los artículos como los usuarios que los han puntuado. Cada relación entre usuario y artículo incluye como propiedades la nota otorgada y el momento en el que se realizó la review.

Para este apartado, se ha decidido cargar los datos desde el fichero porque era mucho más costos filtrar por la categoría en una consulta de MySQL para recuperar los artículos de un tipo específico.

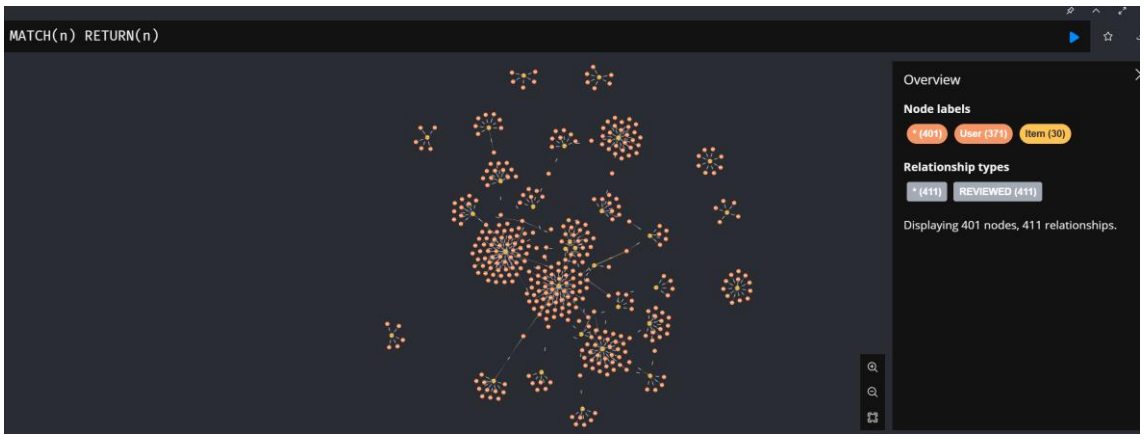

Figura 14. Ejemplo enlaces artículos y usuarios (n=30, tipo= instrumentos)

```
Base de datos de Neo4j limpiada correctamente.
Introduce el tipo de artículo (por ejemplo: [toys, video_games, music, instruments]): instruments
El tipo debe estar en la lista [toys, video_games, music, instruments]: 30
El tipo debe estar en la lista [toys, video_games, music, instruments]: instruments
Introduce la cantidad de productos a seleccionar (máximo 90): 30
Número de artículos elegido: 30
Artículos y reviews ya están cargados correctamente en Neo4j.
```

Figura 15. Output por consola de enlaces artículos y usuarios

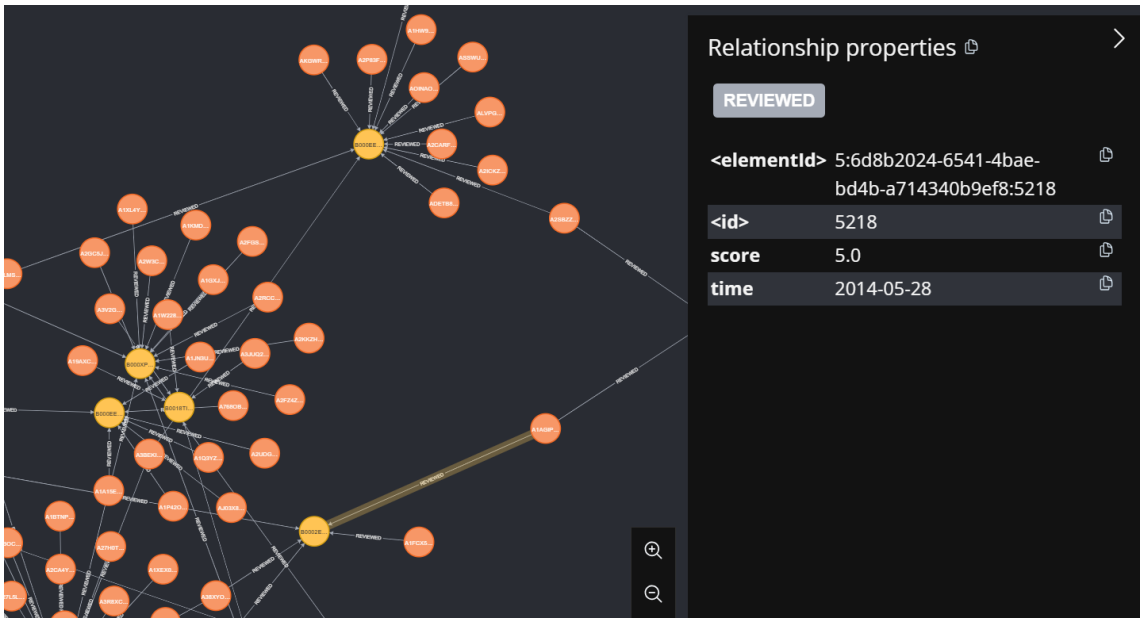

Figura 16. Propiedades score y time de las relaciones

Javier Mendoza Guerrero
Bertrán Vidal Campos

4.3. Usuarios que han visto más de un determinado tipo de artículo

A continuación, se seleccionan los primeros 400 usuarios ordenados alfabéticamente por su nombre. De ese conjunto, se filtran aquellos usuarios que hayan puntuado artículos pertenecientes a al menos dos tipos diferentes.

En Neo4J se representan mediante nodos tanto los usuarios como los tipos de artículos que hayan puntuado. Las aristas entre cada usuario y los distintos tipos de productos que consumieron incluyen una propiedad que indica el número de artículos puntuados de ese tipo. De nuevo aquí hemos decidido cargar los datos desde los ficheros por mayor rapidez que cargarlos desde MySQL filtrando por el tipo.

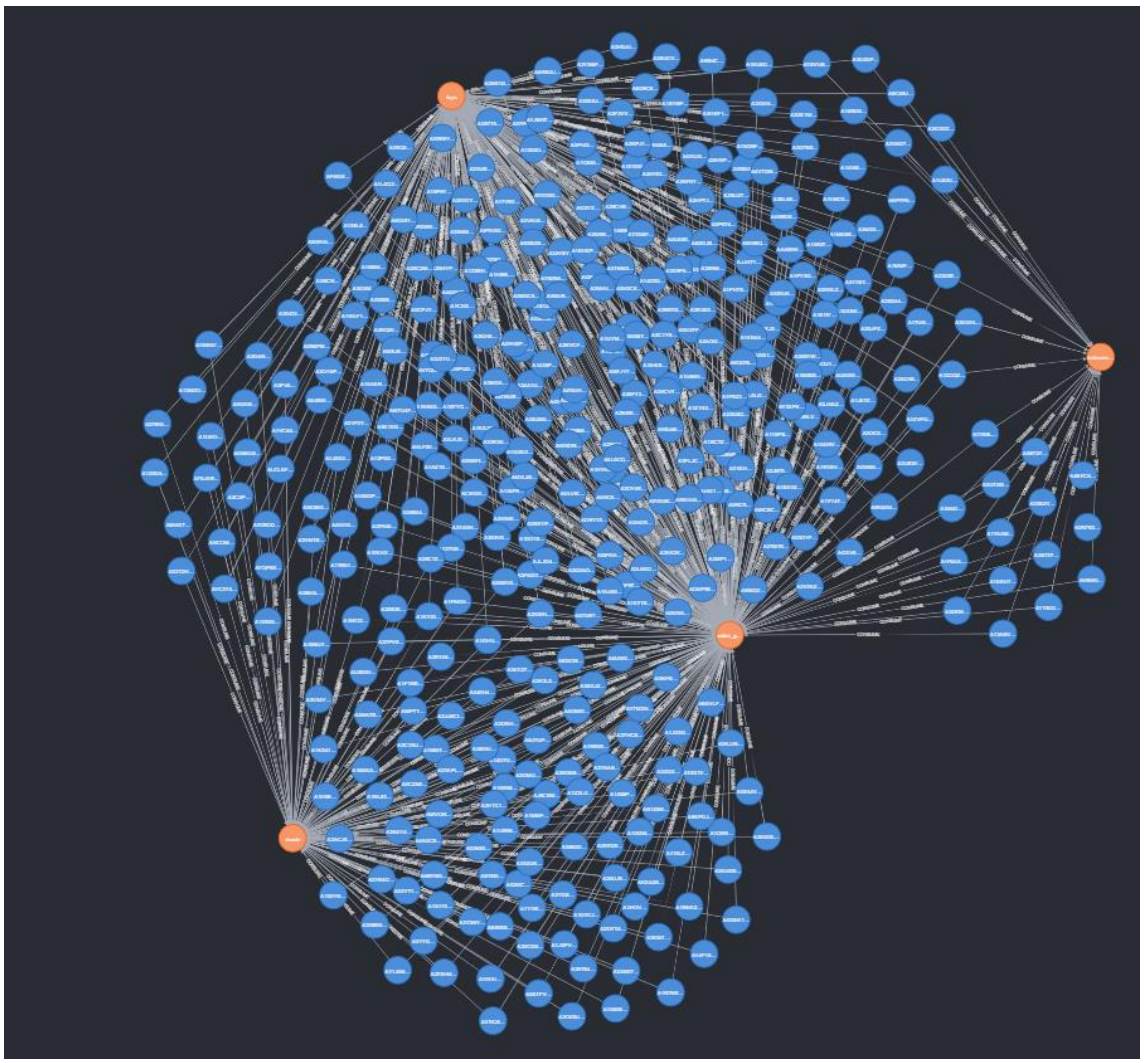

Figura 17. Usuarios con reseñas de más de 2 categorías

Javier Mendoza Guerrero
Bertrán Vidal Campos

4.4. Artículos populares y artículos en común entre usuarios

Por último, se ha implementado una funcionalidad que permite seleccionar los 5 artículos más populares (es decir, con más valoraciones) que tengan menos de 40 reviews. Estos artículos se visualizan en Neo4J junto con todos los usuarios que los hayan puntuado. Para realizar este filtrado primero se ha utilizado MySQL.

Además, para el resultado obtenido con el filtrado, se calculan los enlaces entre usuarios en función del número de artículos que hayan puntuado en común. Estas relaciones se representan en Neo4J con una propiedad que indica la cantidad de artículos compartidos entre cada par de usuarios.

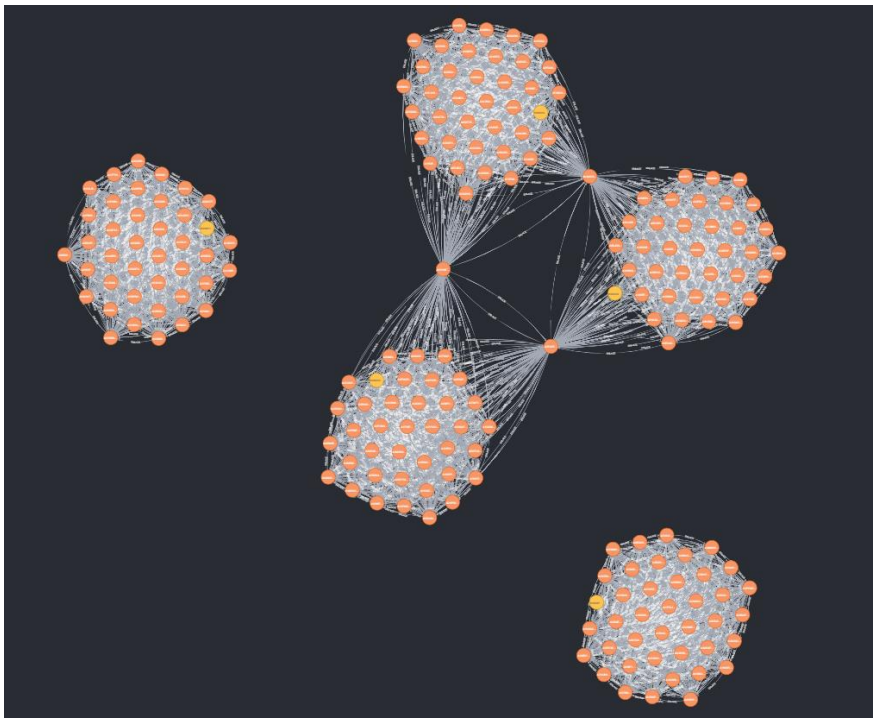

Figura 18. Top 5 artículo más populares

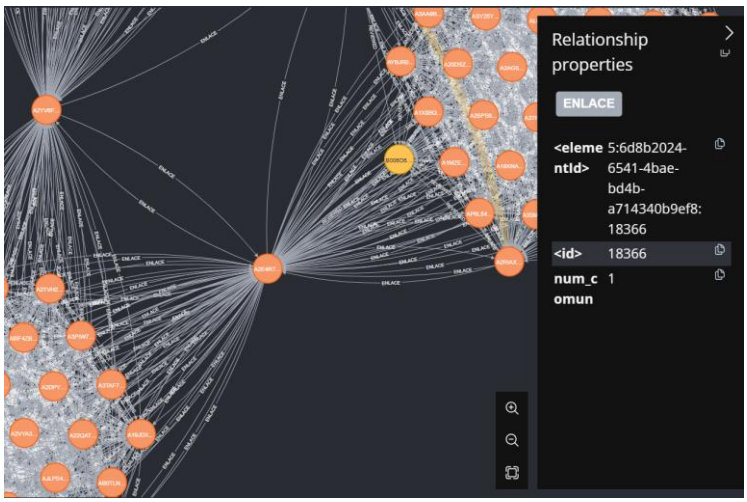

Figura 19. Propiedad relaciones num_comun

Javier Mendoza Guerrero
Bertrán Vidal Campos

5. NUEVOS DATOS

Hemos insertado un archivo adicional a la base de datos, distinto a los 4 con los que hemos estado trabajando, para ampliar la base de datos de reviews. El mecanismo de inserción ha sido casi idéntico al que utilizamos al crear la base de datos inicial, con la ligera diferencia de que el ID para cada review parte desde el último índice creado, es decir, desde la longitud de las reviews que ya existían. En particular, la base de datos que hemos decidido añadir es 'Pet_Supplies_5.json'."

```
#obtener el número total de reseñas en la tabla Reviews
cursor_mysql.execute("SELECT COUNT(*) FROM Reviews")
result = cursor_mysql.fetchone()
total_reviews = result[0]

#se inicia el contador de reviewID a partir del total de reseñas
autoincrement = total_reviews + 1
```

Figura 20. Cálculo del primer ID de las reviews del archivo *Pet_Supplies_5.json*

6. VISUALIZACIÓN Y ML

6.1. Visualización

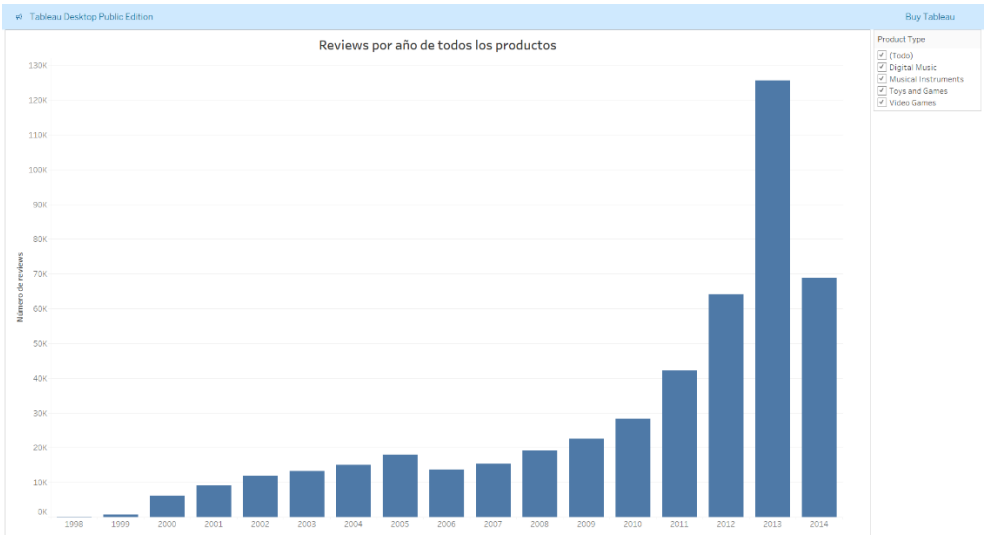

Figura 21. Gráfica reviews por año en Tableau

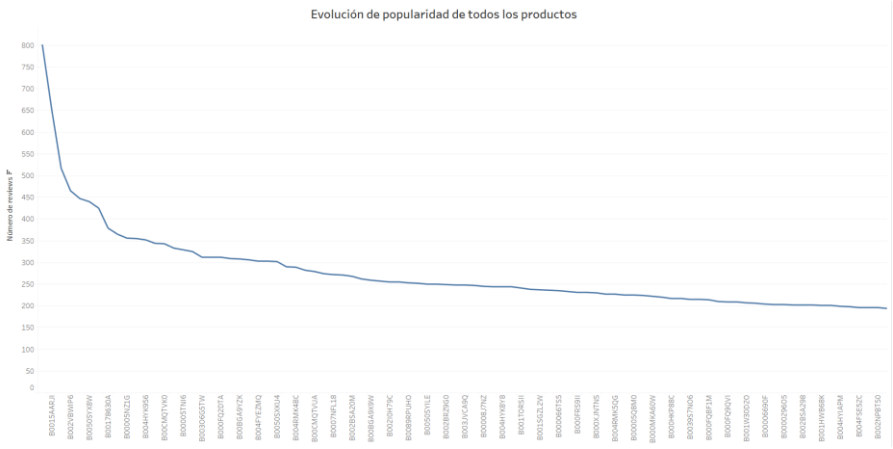

Figura 22. Gráfica evolución de popularidad en Tableau

Javier Mendoza Guerrero
Bertrán Vidal Campos

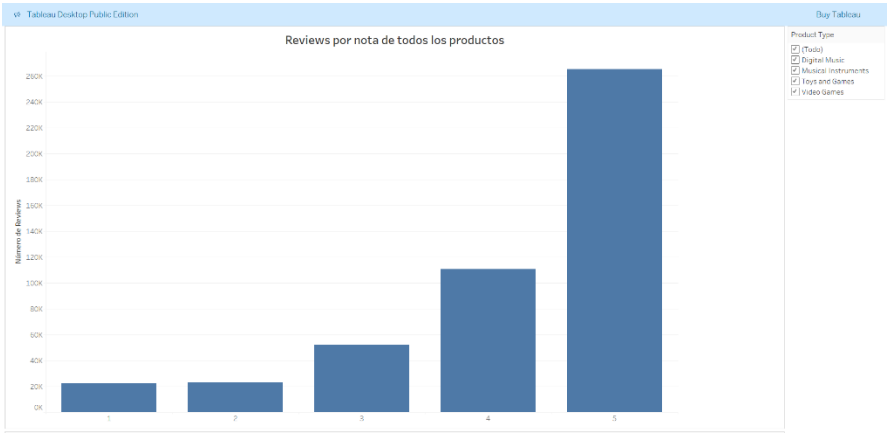

Figura 23. Gráfica reviews por nota en Tableau

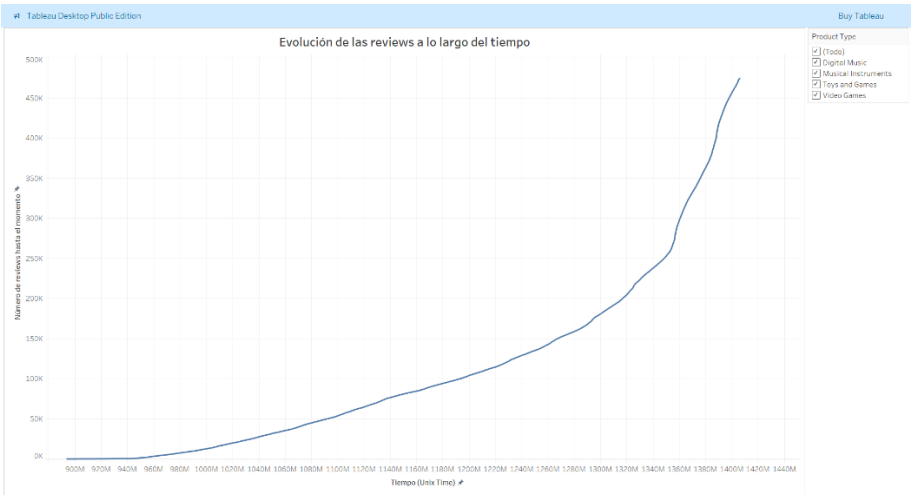

Figura 24. Gráfica evolución de las reviews en Tableau

Javier Mendoza Guerrero
Bertrán Vidal Campos

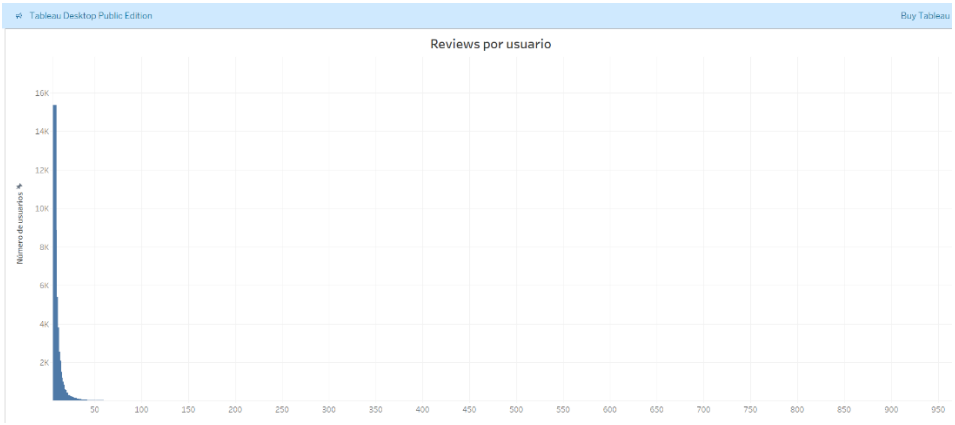

Figura 25. Gráfica reviews por usuario en Tableau

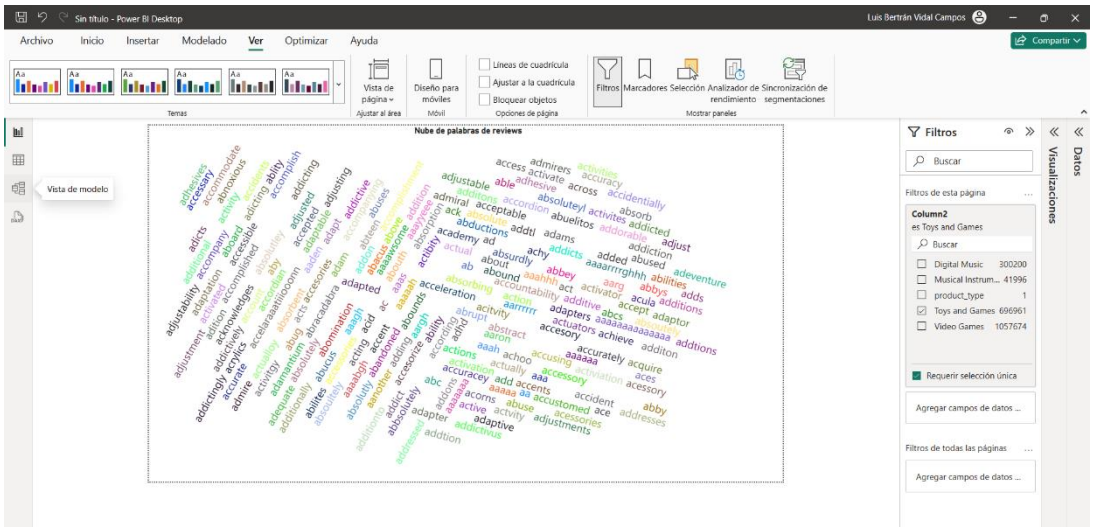

Figura 26. Nube de palabras en PowerBI

Javier Mendoza Guerrero
Bertrán Vidal Campos

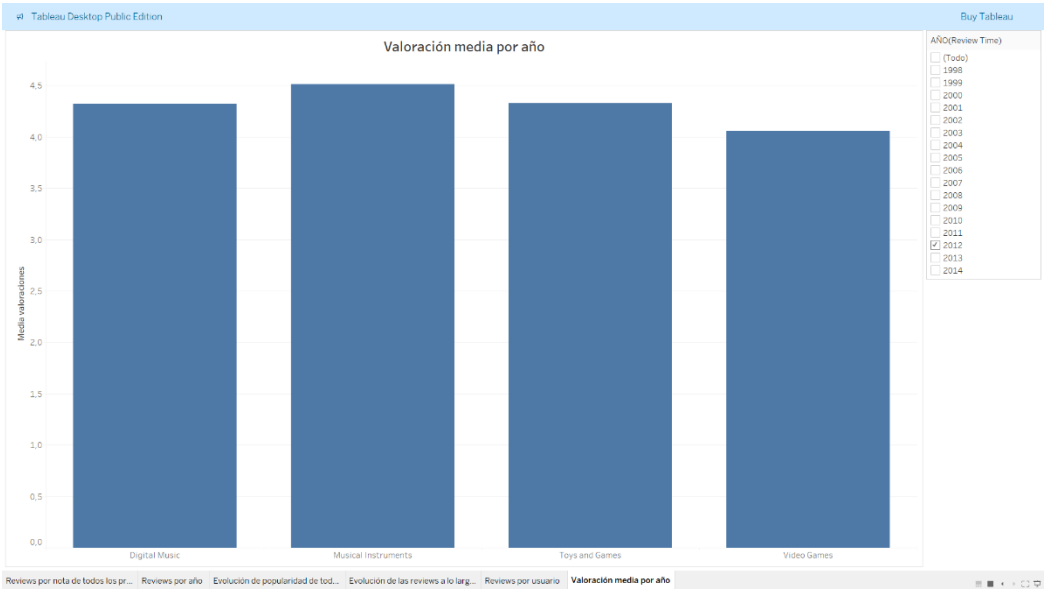

Figura 27. Gráfica valoración media por año en Tableau

6.2. Machine Learning

(Breve descripción de como usaríamos Red Neuronal MLP para efectuar recomendaciones de artículos a usuarios.)

Elección del modelo

El modelo que consideramos mejor se ajustaría a los datos sería un MLP (Multilayer Perceptron) un tipo de red neuronal que tiene al menos una capa oculta y en el que las conexiones entre neuronas están completamente conectadas, es decir, cada neurona en una capa está conectada a todas las neuronas de la siguiente capa.

En cuanto a la elección de este modelo por encima de otros como Regresión Lineal y Logística, arboles de clasificación o Máquinas de vector soporte (SVM) , se debe a que las redes neuronales son poderosas para modelar relaciones complejas y no lineales, lo que las hace ideales para sistemas de recomendación con datos de alta dimensionalidad, como calificaciones de productos. Son escalables y pueden manejar grandes volúmenes de datos como es el caso de nuestra base de datos. Por supuesto

este modelo tiene limitaciones ya que los MLP requieren una gran potencia computacional y además son difíciles de interpretar debido a la gran cantidad de hiperparámetros y su naturaleza de "caja negra", es decir que podemos saber exactamente cómo funciona un perceptrón individual pero no el conjunto MLP.

Pasos para implementar MLP:

1. **Preprocesamiento de datos**
   Primero, separamos los datos en conjuntos de entrenamiento (train) y prueba (test). Luego, tratamos los datos faltantes; dado que este es un dataset grande, podemos eliminar directamente las filas con datos faltantes, a menos que esos datos sean importantes para la predicción. Para detectar outliers, utilizamos boxplots o QQ-plots, y de nuevo, eliminamos aquellos que no aporten información relevante sobre algún producto. Posteriormente, realizamos el encoding de las variables categóricas. Por ejemplo, para variables en texto, asignamos puntuaciones a aquellos comentarios que mencionan ciertas palabras clave como "buena" o "útil". Este es un proceso muy complejo y costoso, así que para sacra información útil de las variables Summary o ReviewTest se podría considerar un modelo probabilístico alternativo al MLP, el modelo de Naive Bayes que es especialmente útil para clasificar mediante textos.

2. **Elegimos los parámetros: Descenso de gradiente y backpropagation**
   En cada capa de la red neuronal, aplicamos una combinación lineal de las características de entrada y, luego, a la salida de esa función lineal, aplicamos una función no lineal. (función de activación)

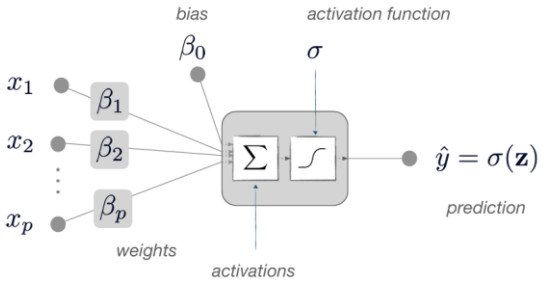

Figura 28. Diagrama de procesamiento en una capa particular

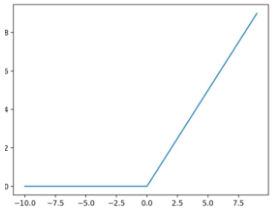

Figura 29. Diagrama función ReLU

Javier Mendoza Guerrero
Bertrán Vidal Campos

En este caso, es útil elegir la función ReLU debido a las propiedades de su derivada durante el descenso de gradiente. Al ser un problema de clasificación, es recomendable usar la función de pérdida log loss o binary cross-entropy. Esta función de pérdida se utiliza para actualizar los parámetros mediante el descenso de gradiente.

$$\theta^{t+1} = \theta^t - \alpha \nabla_\theta \mathcal{C}(\theta^t, \mathcal{D})$$

Figura 30. Función descenso gradiente

Para calcular el gradiente, necesitamos usar el algoritmo de backpropagation.

$$\frac{\partial \mathcal{C}_d}{\partial w_{ij}^k} = \underbrace{\frac{\partial \mathcal{C}_d}{\partial a_j^k}}_{\delta_j^k} \underbrace{\frac{\partial a_j^k}{\partial w_{ij}^k}}_{o_i^{k-1}} = \delta_j^k o_i^{k-1}$$

$$\delta_j^k = \sum_{\ell=1}^{n_{k+1}} \frac{\partial \mathcal{C}_d}{\partial a_\ell^{k+1}} \frac{\partial a_\ell^{k+1}}{\partial a_j^k}$$

Figura 31. Regla de la cadena en Backpropagation

Si implementáramos esto desde cero, tendríamos que calcular analíticamente la derivada del gradiente utilizando la regla de la cadena e implementarla en el código. Sin embargo, este proceso puede simplificarse utilizando librerías de Python como Keras (específicamente diseñada para MLP) o Scikit-learn, que son ampliamente usadas en muchos modelos de aprendizaje automático.

1. **Determinar el número de capas**
   Comenzamos con un número de neuronas que se encuentre entre el número de entradas y salidas. Dado que este es un problema complejo, se utilizarán varias capas, pero estas estarán limitadas con regularización (por ejemplo, utilizando regularización Ridge, que penaliza los parámetros de la red para evitar que tomen valores demasiado altos). Esto es esencial para prevenir el overfitting en el modelo.

$$Loss_{Ridge} = L(\hat{y}, y) + \lambda \sum_{i=1}^{M} \beta_i w_i^2$$

Figura 32. Regularización Ridge sobre parámetros MLP

2. **Evaluación en datos de prueba**
   Después de realizar la validación cruzada (cross-validation) en los datos de entrenamiento, si estamos satisfechos con el rendimiento del modelo (es decir, si el error es bajo y no hay overfitting ni underfitting), entonces probamos el modelo en los datos de test.