PROYECTO FINAL BBDD:

Reviews de Amazon

Beltrán Sánchez Careaga

Ignacio Queipo de Llano Pérez-Gascón

Índice

1. Introducción

Este proyecto se centra en las reviews de Amazon, explorando la integración de bases de datos relacionales y no relacionales. El trabajo abarca el diseño y carga de datos en MySQL y MongoDB respectivamente, además de aplicaciones Python para acceso, visualización y análisis. Se explora la similitud entre usuarios, la relación entre usuarios y artículos utilizando Neo4J, y se expande con nuevas visualizaciones y la relación con Machine Learning para predicciones de puntuaciones de productos.

2. Diseño y carga de datos

En esta primera parte se lleva a cabo la integración de herramientas vistas a lo largo del curso, combinando bases de datos relacionales y no relacionales. Para ello, se ha diseñado cuidadosamente qué información almacenar en MySQL y qué información almacenar en MongoDB. Se ha diseñado un esquema relacional que define las relaciones entre las distintas tablas en MySQL, con el objetivo de mantener la integridad de los datos y facilitar las consultas complejas.

Con respecto a las tablas y colecciones creadas, teniendo en cuenta los 4 ficheros .json de los cuales teníamos que extraer los datos, se ha decidido crear 2 tablas, una llamada *Reviews*, la cual contiene datos pertenecientes a las mismas, y otra llamada *Reviewers*, conteniendo nombre e id de toda persona que haya hecho alguna review.

2.1. Esquema Relacional en MySQL

Se ha creado un esquema relacional en MySQL que refleja las relaciones entre las distintas entidades de datos. El diseño del esquema se ha basado en las características de los conjuntos de datos originales y en la naturaleza de las relaciones entre ellos. A continuación se presenta una descripción del esquema relacional de las tablas:

Reviews(
reviewID: INT PRIMARY KEY AUTO_INCREMENT,
reviewerID: VARCHAR(255),
asin: VARCHAR(255),
overall: FLOAT,
unixReviewTime: INT,
min_helpful: INT,
max_helpful: INT
category: VARCHAR(255)
FOREIGN KEY (reviewerID) REFERENCES Reviewers(reviewerID))

Reviewers(
reviewerID: VARCHAR(255) PRIMARY KEY,
reviewerName: VARCHAR(255))

2.2. Justificación del Diseño

Los datos se han distribuido entre MySQL y MongoDB considerando las siguientes consideraciones:

- MySQL: Se ha optado por almacenar los datos estructurados y altamente relacionales en MySQL. Esto incluye información como valores numéricos o cadenas de una sola palabra, que tienen una estructura fija y relaciones claras entre sí.
- MongoDB: Por otro lado, se ha decidido almacenar datos semi-estructurados y no relacionales en MongoDB. Esto incluye información como cadenas largas de caracteres, que pueden variar en estructura y longitud y no requieren relaciones complejas.

La razón de este diseño viene dado a que gracias a prácticas anteriores pudimos ver como MongoDB funcionaba muchísimo mejor que MySQL a la hora de filtrar y encontrar datos no estructurados, mientras que MySQL funciona muy bien siempre y cuando los datos sean numéricos, debido a la poca flexibilidad que presenta con respecto a MongoDB

2.3. Implementación

Toda la funcionalidad de creación de bases de datos, tablas/colecciones y carga de datos se ha realizado mediante Python. Se ha utilizado un archivo de configuración (*configuracion.py*) para almacenar las credenciales de conexión y detalles de las bases de datos, lo que facilita la configuración y la gestión de cambios mientras que el script principal *load_data.py* se encarga de la carga de los datos en ambas bases, asegurando una integración fluida y consistente de la información.

3. Aplicación Python para acceso y visualización

Para esta parte, nos hemos decantado por realizar un dashboard con la librería Dash de Python, ya que nos parecía la manera más práctica para el usuario de poder ver las diferentes visualizaciones de una forma dinámica y sencilla.

Todo el código relacionado con la implementación de dicho dashboard se puede encontrar en el fichero *menu_visualizacion.py,* en el cual se puede observar tanto las queries utilizadas, como las distintas conexiones creadas y toda la parte relacionada con dash. Este dashboard viene dividido en 2 pestañas, una primera centrada en las evoluciones temporales, y otra centrada en los gráficos relacionados con el conteo como podrían ser los scatters, histogramas o incluso la nube de palabras, de manera que no tenemos una pestaña sobrecargada con todos los gráficos y permite a su vez una mejor actualización de las mismas en caso de que se desee cambiar cualquiera de los filtros permitidos.

Destacar en esta parte que hay veces en las que la aplicación de dash da error debido a un problema de conexión entre sql con python y dash. No pasa nada, no es un error de código como tal, volver a ejecutar cerrando vsc y sql hasta que funcione.

4. Aplicación Python junto con Neo4J

4.1. Obtener similitudes entre usuarios y mostrar los enlaces en Neo4J

Descripción:

En este apartado, se implementa la funcionalidad para calcular y visualizar las similitudes entre usuarios basadas en las reviews que han realizado. La similitud entre usuarios se calcula utilizando el índice de Jaccard, que mide la similitud entre dos conjuntos dividiendo el tamaño de su intersección por el tamaño de su unión.

Proceso:

1. Se seleccionan los usuarios con más reviews, donde el número de usuarios seleccionados es configurable.

```
Nodos eliminados correctamente.
Las 30 personas con más reviews son:
ReviewerID: A3V6Z4RCDGRC44, ReviewerNAME: Lisa Shea "be the change you wish to see in t..., Total de Reviews: 779
ReviewerID: AJKWF4W7QD4NS, ReviewerNAME: N. Durham "Big Evil", Total de Reviews: 671
ReviewerID: A3W4D8XOGLWUN5, ReviewerNAME: Michael Kerner "Michael Kerner", Total de Reviews: 652
ReviewerID: A9Q28YTLYREO7, ReviewerNAME: mistermaxxx08 "mistermaxxx08", Total de Reviews: 578
ReviewerID: AJGU56YG8G1DQ, ReviewerNAME: E. Kennedy, Total de Reviews: 550
ReviewerID: A2QHS1ZCIQOL7E, ReviewerNAME: Richard Baker "BinaryMessiah", Total de Reviews: 450
ReviewerID: A29BQ6B90Y1R5F, ReviewerNAME: S. Rhodes, Total de Reviews: 404
ReviewerID: A2582KMXLK2P06, ReviewerNAME: B. E Jackson, Total de Reviews: 403
ReviewerID: A3HU0B9XUEVHIM, ReviewerNAME: Andre S. Grindle "Andre' Grindle", Total de Reviews: 375
ReviewerID: AFV2584U13XP3, ReviewerNAME: Rich(Anime&Games Will Always Prevail!), Total de Reviews: 325
ReviewerID: A20DZX38KRBIT8, ReviewerNAME: Deimos ".", Total de Reviews: 325
ReviewerID: A3KJ6JAZPH382D, ReviewerNAME: Tim Brough "author and music buff", Total de Reviews: 301
```

2. Para cada par de usuarios seleccionados, se calcula la similitud de Jaccard basada en los artículos que han revisado en común.
3. Se almacenan las similitudes calculadas en un archivo auxiliar.
4. Se cargan las similitudes en Neo4J, donde cada nodo representa un usuario y los enlaces entre nodos representan la similitud entre usuarios.

Resultados:

Se obtiene una representación visual en Neo4J de las similitudes entre los usuarios seleccionados. Cada nodo en el grafo representa un usuario, y los enlaces entre nodos indican la similitud entre ellos. Una vez la carga de datos en Neo4J se ha realizado de manera exitosa, se permite la visualización y el análisis de las relaciones entre usuarios basadas en sus reviews al usuario.

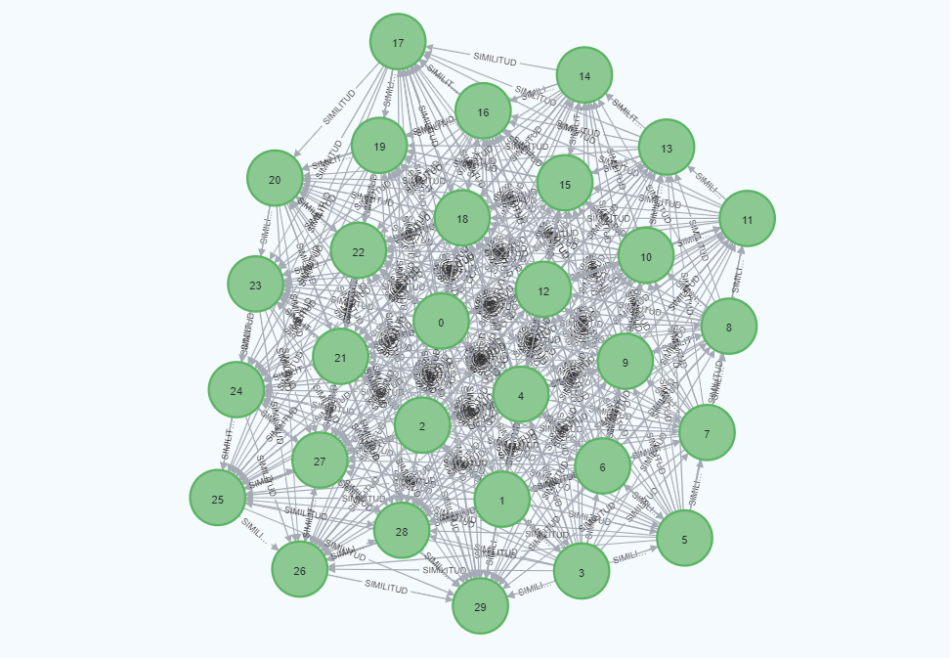

Imagen de las similitudes entre los usuarios obtenida en Neo4J.

Consulta adicional:

Se muestra por pantalla, mediante una consulta a Neo4J, al usuario que tiene más vecinos, lo que proporciona información adicional sobre la estructura de la red de usuarios en función de la similitud de sus reviews.

```
SIMILITUDES ENTRE 30 USUARIOS CARGADAS.
El nodo con más vecinos es: <Node element_id='4:dfcd32f9-c997-48d5-adee-927efa951577:524' labels=frozenset({
'Reviewer'}) properties={'reviews': 'B0000025BA,B000001EC1,B0000029B0,B0000024SN,B000002VUG,B000002L7G,B001Q
55D5K,B001SGZL2W,B001T8S62Q,B001QA2TYS,B0028ZNX68,B0028ZJ4O8,B002BS4JLA,B002BS47JE,B002BS47TE,B002BSA298,B00
2BSA20M,B0025QJU1C,B002B1TDV8,B002BRYIJY,B002BRUTT2,B002BRYXRQ,B002AB7TX8,B001O2J036,B001P1ZE68,B001L0XUGO,B
001TORSII,B001TOQ8JS,B004NRN5EO,B004PAGJN8,B004P6IVPQ,B004HHIB8U,B004FZ9822,B004HVKAA8,B004H0LFEY,B004FYEZMQ
,B004RIACMW,B004I5EE46,B004JN176K,B004WLRQAU,B0050SVGW8,B00897Z27C,B008HHTEXW,B006WQR3OM,B007BUA0AM,B0088MVO
ES,B0084380ZM,B002MFU290,B002MCG8MI,B002N42SDS,B0038PBF1W,B00347BRU8,B002XNW6T0,B0030F1DOO,B003O6DZSQ,B003IC
GL7I,B003JVKHEQ,B003JD7QYI,B003L8HQ7S,B003JZNDN4,B002SRP9VI,B002SU4QG4,B003O6FXFY,B003O6G6G4,B003VJNPPE,B003
YVK5T8,B00473Y6VA,B0045FCKVI,B003S2SQFS,B003P9C6QY,B0041RWIGW,B004BJ4HDC,B0049DYNNO,B0049PBOKW,B004EQCCI4,B0
05XOJ6NK,B005QF1OSG,B0053BG26C,B0053BQU4G,B005LBDOLK,B0050SYUSU,B0050SYYEK,B0055SWM08,B0058VI1SO,B006L1FAK2,
B006JSXVA8,B006ULENFG,B002I0H9WM,B002I0HBZW,B002I0IUM0,B002I0J4NE,B002I0J51U,B002I0KOLA,B', 'reviewer_name':
 'Lisa Shea "be the change you wish to see in t...', 'reviewer_id': 'A3V6Z4RCDGRC44', 'num_reviews': 779}>,
con 29 vecinos.
```

4.2. Obtener enlaces entre usuarios y artículos

Descripción:

En esta parte del proyecto, se desarrolla la funcionalidad para seleccionar un número de artículos aleatorios de un tipo específico y mostrar en Neo4J los usuarios que han consumido o puntuado esos artículos. Además, se incluyen propiedades en los enlaces entre usuarios y artículos para representar la nota de la review y el momento en el que se realizó la review.

Proceso:

1. Se selecciona un número especificado de artículos aleatorios de un tipo determinado, donde el usuario proporciona el tipo de artículo y el número deseado de artículos.
2. Se recuperan las reviews asociadas a los artículos seleccionados, incluyendo la información del revisor, el artículo, la puntuación y la fecha de la review.
3. Para cada review, se crean nodos en Neo4J para representar tanto al usuario como al artículo, y se crean enlaces entre ellos con propiedades que incluyen la puntuación y la fecha de la review.
4. Se carga la información en Neo4J para su visualización y análisis.

Resultados:

Se obtiene una representación visual en Neo4J de los usuarios que han revisado los artículos seleccionados, junto con la información de la puntuación y la fecha de las reviews. Cada nodo en el grafo representa un usuario o un artículo, y los enlaces entre nodos muestran las relaciones entre ellos basadas en las reviews realizadas.

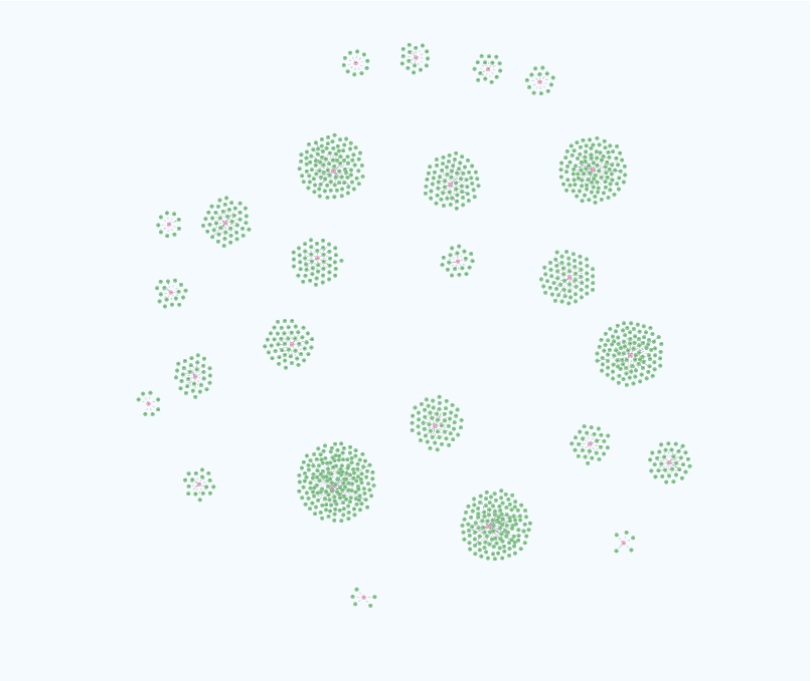

Figura de la imagen obtenida al estudiar 25 artículos aleatorios de la categoría ‘VideoGames’

Consideraciones:

Es fundamental garantizar que la base de datos en Neo4J esté limpia antes de cargar los nuevos datos para evitar conflictos o duplicaciones. Por ello, antes de ejecutar cualquier consulta, se le llama a una función que limpia la base de datos de Neo4j para no distorsionar la salida.

4.3. Obtener algunos usuarios que han visto más de un determinado tipo de artículo:

Descripción:

En esta sección del proyecto, se desarrolla la funcionalidad para seleccionar a los primeros 400 usuarios ordenados por nombre que hayan puntuado artículos de al menos dos tipos diferentes. Luego, se muestra en Neo4J nodos representando a estos usuarios y a los diferentes tipos de artículos que han puntuado.

Proceso:

1. Se seleccionan los primeros 400 usuarios ordenados alfabéticamente por nombre que han puntuado artículos de al menos dos tipos distintos.
2. Para cada usuario seleccionado, se recuperan los tipos de artículos que ha puntuado y se crean nodos en Neo4J para representar tanto al usuario como a cada tipo de artículo.
3. Se crean enlaces entre los nodos de usuario y tipo de artículo, con una propiedad que indica el número de artículos puntuados de ese tipo por el usuario.
4. Se carga la información en Neo4J para su visualización y análisis.

Resultados:

Se obtiene una representación visual en Neo4J de los usuarios seleccionados y los diferentes tipos de artículos que han puntuado. Cada nodo en el grafo representa un usuario o un tipo de artículo, y los enlaces entre nodos muestran las relaciones entre ellos basadas en las reviews realizadas por los usuarios.

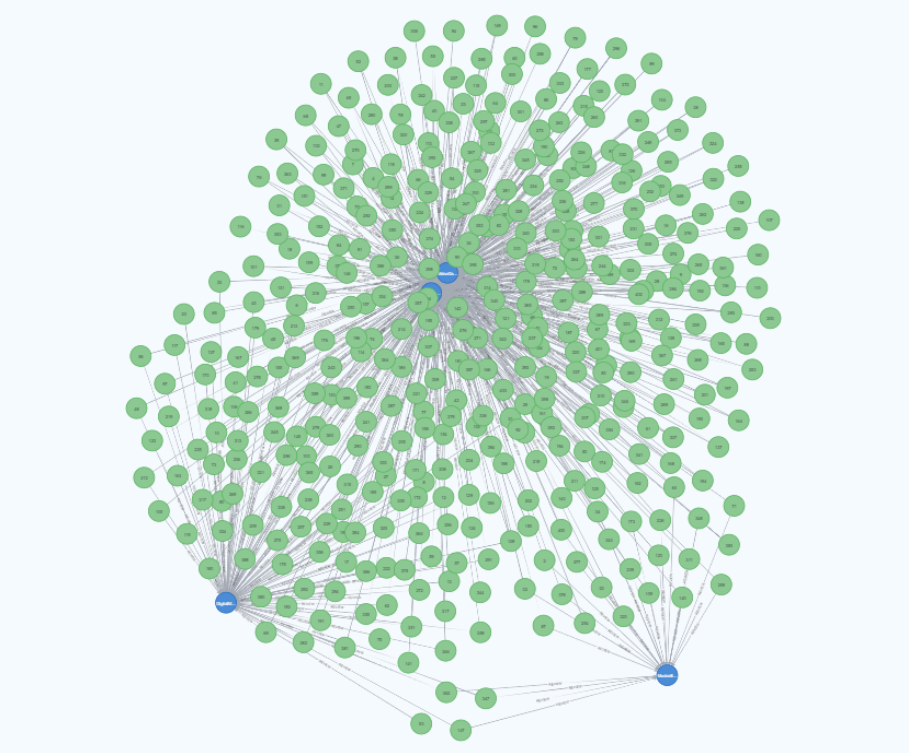

Imagen obtenida al consultar los usuarios que han puntuado elementos de más de 2 tipos donde las aristas contienen el número de elementos que se han puntuado.

4.4. Artículos populares y artículos en común entre usuarios

Descripción:

En esta parte del proyecto, se desarrolla la funcionalidad para seleccionar los 5 artículos más populares que tengan menos de 40 reviews y mostrar en Neo4J los usuarios que los han puntuado. Además, se calculan los enlaces entre los usuarios mostrando cuántos artículos han puntuado en común.

Proceso:

Se seleccionan los 5 artículos más populares que tengan menos de 40 reviews.
Para cada uno de estos artículos, se recuperan los usuarios que los han puntuado y se crean nodos en Neo4J para representar tanto a los usuarios como a los artículos.
Se crean enlaces entre los nodos de usuario y artículo, y se calcula el número de artículos que han puntuado en común los usuarios.

Se carga la información en Neo4J para su visualización y análisis.

Resultados:

Se obtiene una representación visual en Neo4J de los usuarios que han puntuado los 5 artículos más populares seleccionados. Cada nodo en el grafo representa un usuario o un artículo, y los enlaces entre nodos muestran las relaciones entre ellos basadas en las reviews realizadas por los usuarios. Además, se muestra el número de artículos que han puntuado en común los usuarios.

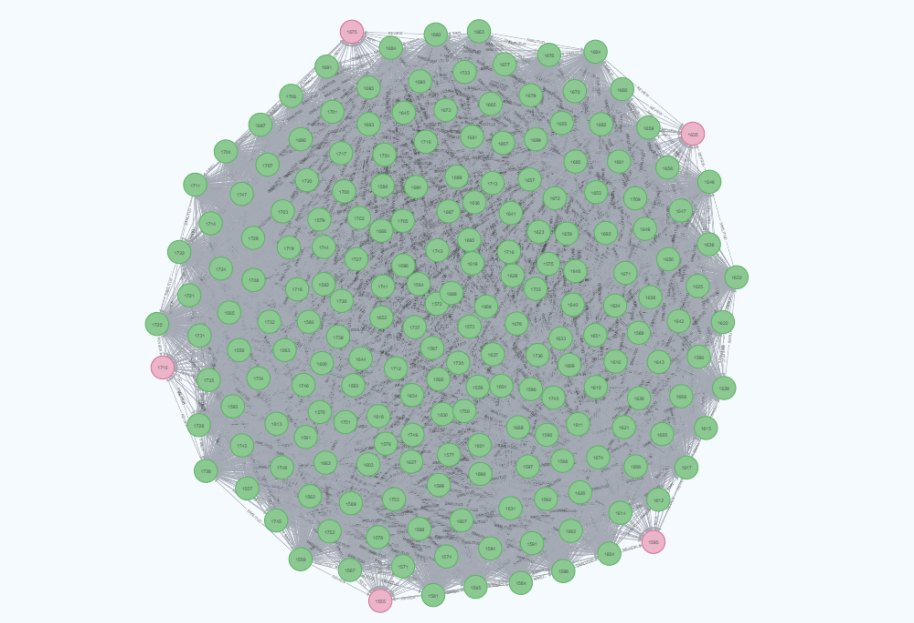

Imagen obtenida al consultar todos los nodos en Neo4J después de cargar los datos.

Mensaje de finalización:

Se muestra un mensaje por pantalla al finalizar la carga en Neo4J para informar al usuario que los datos están disponibles para su consulta en Neo4J.

5. Nuevos datos

Para incorporar nuevos datos, se ha seguido un proceso similar al utilizado en la carga de datos inicial, reciclando gran parte del código y modificando lo necesario para crear un nuevo fichero llamado *inserta_data.py,* con el cual se pueden insertar tantos ficheros como se deseen, almacenando de forma correcta los ficheros y sus directorios en el primer diccionario llamado: “PATH_DATASETS”.

Debido a la gran similitud entre ambas partes, consideramos innecesaria ningún tipo de explicación extensa sobre cómo se ha llevado a cabo, puesto que nuestra base de datos sigue teniendo el mismo diseño y lo único que se ha tenido que considerar es el punto en el que empiezan los nuevos reviewsID, ya que son la PK de nuestra tabla principal (*reviews*). Al ser esta clave una que va incrementando, teníamos que tener en cuenta cuál fue la última review insertada, y viendo esa última

review, accedemos a su reviewID para saber que la próxima review insertada, debe tener un reviewID igual al antiguo más uno, lo cual se puede observar implementado en esta parte del código:

```
conexion = pymysql.connect(
        host=MYSQL_HOST, user=MYSQL_USER, password=MYSQL_PASSWORD
    )
    cursor = conexion.cursor()
    conexion.select_db(MYSQL_DATABASE)
    cursor.execute(f"SELECT MAX(reviewID) FROM REVIEWS")
    review_id = cursor.fetchone()[0]
```

Donde el review_id es el ID de la última review insertada en la base de datos hasta la fecha. Así, con esta pequeña modificación, se consigue insertar tantos nuevos archivos como queramos, siempre y cuando las columnas de estos archivos sean las mismas y cumplan el diseño de nuestra base de datos.

6. Más visualización y relación con Machine learning

6.1 Más visualización

Una vez hechas todas las visualizaciones con dash, hacemos lo mismo con PowerBi y así ver la capacidad del mismo para conectarse a bases de datos. Estos son los resultados:

1. Evolución de reviews por años: Histograma del número de reviews a lo largo de los años, con opción de seleccionar tipo de producto o visualizar todos:

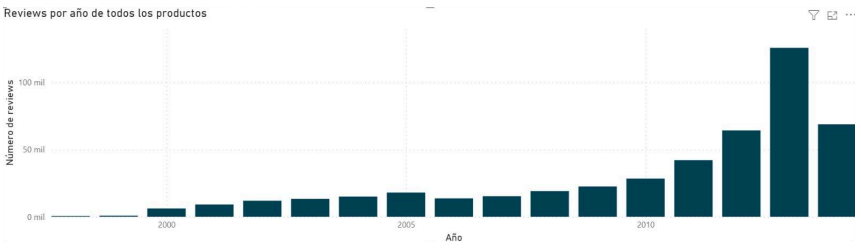

Figura 1: Evolución de reviews por años

2. Popularidad de los artículos: Curva que ordena los artículos por el número de reviews, segmentable por tipo de producto:

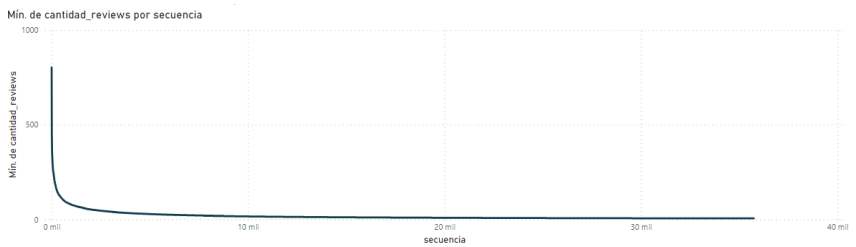

Figura 2: Evolución de la popularidad de los productos

3. Histograma por nota: Distribución del número de artículos según su puntuación, basado en 'overall':

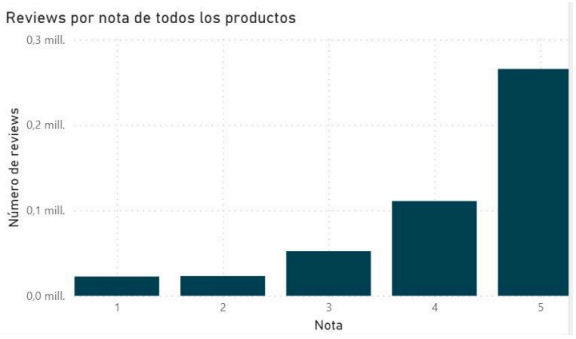

Figura 3: Histograma por nota

4. Evolución de reviews por categorías: Muestra la evolución de reviews por fecha para cada categoría de producto:

Figura 4: Evolución de las reviews a lo largo del tiempo

5. Histograma de reviews por usuario: Distribución del número de reviews por usuario:

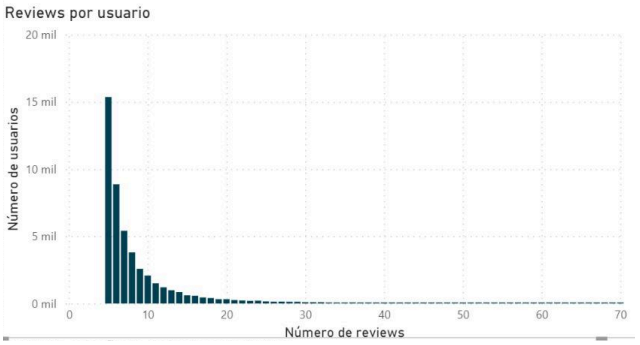

Figura 5: Número de reviews por usuario

6. Evolución de reviewed activos: Muestra el número de reviewers que realizan reviews a lo largo del tiempo:

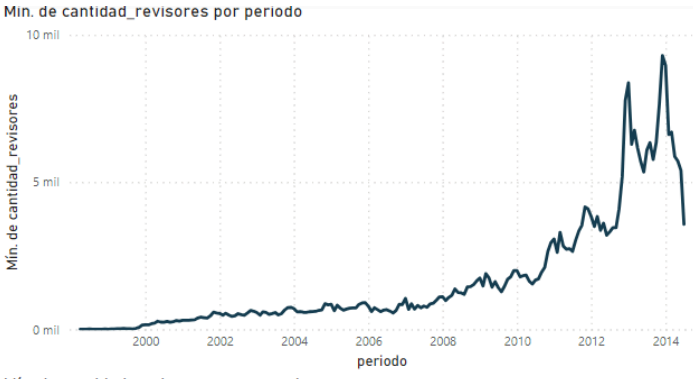

Figura 6: Evolución de reviewers activos

6.2 Relación con ML

Para el modelo de Machine Learning, deberíamos primero elegir cuales de los campos son los que vamos a emplear en el modelo, realizando el procesamiento de datos correspondiente. Así, tomaríamos como características relevantes reviewerID, asin, category (utilizando codificación de variables dummy), overall (normalizada entre 0 y 1) y helpful, donde helpful sería la proporción entre el número de reviews que han ayudado (min_helpful) y el número de reviews totales (max_helpful).

Una vez realizado el procesamiento y la normalización de los datos, pasamos a la creación de las matrices "X" (para los inputs) e "y" (para el output a predecir):

- X.columns: reviewerID, asin, variables dummy para category y helpful
- Y.columns: overall

Con esta parte hecha, dividimos los datos en un conjunto de *train* y otro conjunto de *test* para con el primero entrenar al modelo y con el segundo comprobar la eficiencia del mismo. En nuestro caso, nos hemos decantado por usar un modelo de Random Forest, debido a su capacidad de gestionar datos complejos y la robustez que presenta frente a datos que sólo aportan ruido.

Finalmente, una vez entrenado el modelo, calculamos su accuracy o su RMSE para ver cómo de eficiente es y si hay que hacer algún reajuste, como encontrar los parámetros óptimos con Cross Validation.