UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

ASESOR DE OUTFIT BASADO EN ANÁLISIS DE TICKETS

Máster Universitario en Ciencia de Datos

Autor: Rico Sánchez-Mateos, Rocío

Director: Jimeno Romero, Pablo
Codirector: Díaz Cortés, Javier

Ponente: Bellogin Kouki, Alejandro
Departamento de Ingenería Informática

FECHA: Junio, 2024

Agradecimientos

Después de la preparación y el esfuerzo, la suerte es el siguiente factor determinante a la hora de conseguir tus metas. Y yo considero que he sido muy afortunada con el equipo de profesionales con el que he tenido el privilegio de trabajar.

Primero, quería agradecer a mi director, Pablo Jimeno, por las reuniones bisemanales que tanto me han ayudado. No solo has invertido tiempo y dedicación, sino que has sabido guiarme perfectamente a través de los retos y callejones sin salida que se nos presentaron durante la realización de este TFM, siempre con una actitud amable.

En segundo lugar, quería agradecer a Alejandro Bellogín por su involucración desde el primer momento en esta aventura, tanto por responder rápidamente a todas mis dudas como por su buen criterio a la hora de dar feedback.

También quería agradecer a Javier Díaz por echarnos una mano cuando se necesitaba y por asegurarse de que, pese a mis despistes, siempre se cumplieran las condiciones de confidencialidad.

Por último, quería agradecer principalmente a mi hermana Celia, que me animó a buscar esta colaboración; pero también a Carlos Roldán, porque sin él no estaría donde estoy ahora; a Samuel Pascual por todo su apoyo; y no puedo olvidarme de mi familia, que está detrás de cada logro que consigo.

Muchas gracias a todos.

Resumen

Este proyecto se sitúa en un contexto donde la digitalización y el análisis de datos son fundamentales para la evolución del comercio minorista, especialmente en la moda. La motivación surge de la alta demanda de ropa en un mundo de tendencias rápidas y diversidad de artículos, buscando ofrecer una experiencia de compra más intuitiva y completa. Si bien actualmente se utilizan modelos computacionales para generar recomendaciones de productos similares a los que un cliente ha comprado, este trabajo busca ir más allá recomendando outfits completos. Basado en el análisis de tickets de compra, el proyecto pretende crear algoritmos que sugieran conjuntos de prendas que complementen un producto específico, aplicando técnicas de minería de datos, teoría de grafos, análisis de redes y procesamiento de imágenes para identificar productos comprados conjuntamente con frecuencia. Pese a la naturaleza subjetiva de este trabajo los resultados prometen ser de utilidad para futuras sugerencias dentro de los artículos seleccionados en la web o incluso para la distribución de estos en la tienda física.

Índice general

Capítulo 1

Introducción

1.1. MOTIVACIÓN DEL PROYECTO

El proyecto se sitúa en un contexto donde la digitalización y el análisis de datos son fundamentales para la evolución del comercio minorista, especialmente en la moda. Actualmente está muy presente el uso de modelos computacionales para generar recomendaciones de productos similares a los que uno compra, sin embargo, en este trabajo se busca dar un paso más allá hacia la recomendación de outfits completos. Esta propuesta se basa en la premisa de que el análisis de tickets de compra proporciona información valiosa sobre las preferencias de los consumidores, permitiendo la generación de algoritmos que sugieran conjuntos de prendas que complementen un producto específico. Se establece una relación con investigaciones previas en áreas como los sistemas de recomendación y el análisis de datos de compra aplicando técnicas de minería de datos, teoría de grafos, análisis de redes y procesamiento de imágenes para identificar conjuntos de productos que se adquieren conjuntamente con cierta frecuencia debido a su compatibilidad.

La motivación del proyecto responde a la gran demanda de ropa que hay en un mundo donde las tendencias cambian rápidamente y hay una diversidad mucho mayor de artículos. Este trabajo pretende ofrecer una solución alineada con las expectativas de los consumidores modernos proporcionándoles una experiencia de compra más intuitiva y completa.

El objetivo principal es desarrollar un algoritmo efectivo que, a partir del análisis de tickets de compra de Inditex, sea capaz de generar recomendaciones de outfits completos para productos específicos. Este algoritmo se basará en técnicas de análisis de datos y modelos de machine learning, considerando patrones de compra para ofrecer recomendaciones precisas y relevantes sobre cómo combinar la prenda en la que el usuario ha mostrado interés.

Adicionalmente, nos ayudaremos de un análisis de imágenes de catálogo para hacer más robusto el modelo inicial. El objetivo aquí es que el algoritmo mejore la señal reconociendo errores en la identificación de artículos.

1.2. OBJETIVOS Y ENFOQUE

El objetivo general es desarrollar un algoritmo de recomendación de outfits basado en el análisis de tickets de compra de Inditex con dos propósitos en mente: a) sugerir productos específicos que complementen los seleccionados en la web con la ayuda de desarrolladores web y b) mejorar la distribución de los artículos en tienda con la ayuda de diseñadores de

interiores comerciales. Para conseguirlo vamos a dividir el proceso en estos hitos parciales de manera progresiva y coherente.

- **Revisión, selección y filtrado de datos**: vamos a analizar y depurar los datos del *data lake* de Inditex sobre los que se tiene acceso para identificar y corregir posibles discrepancias o valores nulos. Además vamos a seleccionar y filtrar atributos relevantes, eliminar información innecesaria o redundante y categorizar por grupos los datos.
- **Planteamiento de un nuevo sistema de recomendación** basado en tickets de compra: vamos a diseñar un modelo que se adapte a las características de los datos previamente tratados para la generación de *outfits* donde el foco principal de la base sea el análisis de los tickets de compra.
- **Desarrollo de un framework**: vamos a integrar técnicas de minería de datos, teoría de grafos, análisis de redes y procesamiento de imágenes en la creación de varios notebooks utilizando la librería PySpark, empleada en en el ámbito del Big Data, para desarrollar el nuevo modelo de recomendación propuesto.
- **Procesado de imágenes**: vamos a hacer más robusto el modelo inicial al unificar elementos que, aun con distintos *IDs*, corresponden al mismo artículo mediante el análisis de embeddings de imágenes.

Capítulo 2

Estado del Arte

2.1. SISTEMAS DE RECOMENDACIÓN

Los tipos más comunes de algoritmos de recomendación son [6, 4]:

2.1.1. Filtrado colaborativo

El filtrado colaborativo [24] se basa en la hipótesis çoincidencia en el pasado implica coincidencia en el futuro". Es decir, si un usuario A tiene gustos en común con otro usuario B, es probable que también le gusten productos que aún no ha consumido, pero que B ya ha afirmado que le gustan. Una forma fácil de cuantificar los gustos es con calificaciones como por ejemplo los *ratings* asociados a películas.

Los algoritmos de filtrado colaborativo se pueden clasificar en dos métodos [1].

- **Método basados en la memoria**: utilizan técnicas estadísticas simples para medir la similitud entre usuarios o elementos y no requieren de pre-entrenamiento.
- **Método basados en modelos**: utilizan técnicas más avanzadas como la factorización matricial, las redes neuronales o técnicas de clústering para aprender características o patrones.

Además, cada uno de estos algoritmos tiene dos posibles enfoques:

- **Basado en usuarios**: encuentra usuarios con perfiles similares al usuario objetivo y recomienda los elementos que han sido del agrado del grupo.
- **Basado en elementos**: encuentra elementos que son similares a los que le gustaron al usuario objetivo y los recomienda.

Para obtener una idea más clara de cómo funcionan los sistemas de recomendación basados en filtrado colaborativo nos vamos a centrar únicamente en el método basado en memoria o método de k vecinos cercanos kNN (k *Nearest Neighbour*) por su sencillez. Concretamente, vamos a explicar solo el enfoque basado en usuario para no extendernos demasiado.

Filtrado colaborativo de vecinos cercanos basado en usuario

El proceso comienza con una matriz cuyos valores son los *ratings* generados por los usuarios (filas) sobre los productos consumidos (columnas). Primero, se aplica una función de similitud sobre la matriz para obtener las similitudes entre usuarios. Después, se

selecciona una cantidad k de usuarios similares (también conocidos como vecinos) para usar en la predicción de *ratings*. Una vez obtenida una lista de todos los artículos con su correspondiente calificación, se ordenan desde el valor predicho más alto al más bajo, para así recomendar solo los artículos más adecuados al usuario objetivo.

Una de varias maneras de calcular la predicción de los *ratings* r para un usuario u sobre un artículo x, basándose en las puntuaciones de sus vecinos v, es mediante una suma ponderada:

$$\hat{r}(u,x) = \frac{\sum_{v \in \eta_k(u)} sim(u,v)\ r(v,x)}{\sum_{v \in \eta_k(u)} |sim(u,v)|}$$

donde $\eta_k(u)$ representa el número de k vecinos más similares a u que ya han calificado el artículo x y donde $sim(u,v)$ es la función similitud entre el usuario u y su vecino v.

Una de las técnicas estadísticas más utilizadas para calcular la similitud entre usuarios es la similitud coseno:

$$sim(u,v) = \cos\left(r(u), r(v)\right) = \frac{\sum_{x \in I} r(u,x)\ r(v,x)}{\sqrt{\sum_{x \in I} r(u,x)^2\ \sum_{x \in I} r(v,x)^2}}$$

donde I denota el conjunto de elementos puntuados por los usuarios.

2.1.2. Filtrado basado en contenido

El filtrado basado en contenido [9] en vez de basarse en la interacción usuario-artículo trabaja con las descripciones de los artículos y el perfil de preferencias del usuario. La similitud vendrá dada por la coincidencia de propiedades entre elementos que le gustaron en el pasado al usuario u y los actuales.

El proceso comienza guardando el comportamiento pasado de un usuario en un vector de perfil o *profile vector* u y toda la información relacionada con los artículos en un vector de elemento o *item vector* x. En el caso de, por ejemplo, una película este último contendría características como el género, el director, la duración, etc. Desde aquí solo tenemos que calcular la similitud coseno del ángulo entre el vector de perfil y el vector de elemento:

$$\hat{r}(u,x) = sim(u,v) = \cos\left(r(u), r(v)\right) = \frac{\sum_{k \in K} u_k x_k}{|u||x|}$$

donde K denota el número total de características que definen el vector elemento, u_k es el componente que indica la importancia de la característica k en consumiciones pasadas para el usuario u del vector perfil, y x_k es la componente que indica la importancia de la característica k en el elemento x del vector elemento.

Una vez obtenida una lista de todos los artículos con su correspondiente valor de similitud asociado se ordenan desde el alto al más bajo para así recomendar los artículos más apropiados al usuario objetivo.

2.2. SISTEMAS DE RECOMENDACIÓN PARA MODA

La teoría presentada a continuación está basada en el revisión del estado del arte presentada en [10].

2.2.1. Mayores retos

La recomendación en el mundo de la moda es un reto complejo debido a la necesidad de modelos multimodales, a la escasez de grandes volúmenes de datos limpios, al carác-

ter subjetivo, cultural y local de las modas, y a la rapidez con la que aparecen nuevas tendencias.

Principalmente se distinguen estos desafíos:

- **Representación de artículos de moda**: la base de muchos sistemas de recomendación tradicionales, como el filtrado colaborativo o el filtrado basado en contenido, es débil porque, en el ámbito de la moda, es difícil encontrar datos detallados sobre la apariencia visual y grandes cantidades de datos de compras. Por esta razón, actualmente se tiende más hacia modelos que puedan usar descripciones de texto o reseñas de clientes.

- **Compatibilidad entre los artículos de moda**: el concepto de combinar prendas de ropa es complejo de traducir a modelos computacionales, más si tenemos en cuenta factores como las tendencias, la estacionalidad, la ubicación o los grupos sociales. Para poder abordar todas estas casuísticas los algoritmos actuales suelen recurrir a la información tanto de imágenes como de textos.

- **Ajuste y personalización de los artículos de moda**: el trasfondo socio-cultural no solo afecta a las compatibilidad entre artículos sino que también condiciona los productos a recomendar en función de la ocasión o circunstancia donde se usará el conjunto. Actualmente los modelos se ayudan de las redes sociales para sortear estos agentes externos, pero aún queda por resolver el problema de las tallas donde la forma del cuerpo puede influir en las elecciones estilísticas.

- **Interpretabilidad y explicación**: confiar ciegamente en las predicciones de los modelos puede no conducir a los mejores resultados a largo plazo, por lo que actualmente se recomienda crear modelos que permitan obtener explicaciones a través de regiones de imágenes destacadas o palabras clave. Además, la interpretabilidad permite a los equipos de gestión de negocios evaluar la idoneidad de las decisiones tomadas por un algoritmo.

2.2.2. Distintos objetivos finales

Existen muchos objetivos finales cuando hablamos de sistemas de recomendación de moda pero los cuatro más importantes son: la recomendación de artículos individuales que combinen con las preferencias del usuario; la recomendación de pares o conjuntos de artículos para conseguir un *outfit*; la recomendación de tallaje dependiendo del material del artículo o el tipo de cuerpo del usuario; y por último, la explicación detrás de las recomendaciones.

En este apartado, se detallará únicamente la recomendación de pares o conjuntos de artículos para conseguir un *outfit*, ya que es el propósito de este proyecto. En este tipo de recomendación los *outfits* son grupos de N artículos que conforman el atuendo de una persona. La forma más simple de un *outfit* es con $N = 2$ donde usualmente una de las prendas corresponde a la parte superior del cuerpo y la otra al tren inferior. Para $N > 2$ se pueden abarcar muchas más categorías como por ejemplo el calzado, los sombreros, los bolsos, etc.

Definimos un *outfit* como:

$$O = \{i_1, i_2, \ldots, i_N\} \in \mathcal{O}$$

donde $i_n \in \mathcal{I}$ es el n-ésimo artículo del *outfit* O, $\mathcal{O}$ es el conjunto de todos los posibles *outfit* y N la longitud del *outfit* cuyo valor no es fijo y puede cambiar para cada *outfit*.

La composición del *outfit* se formula de la siguiente manera: dada una función $s(O)$ que indica qué tan bien el *outfit* $O \in \mathcal{O}$ funciona, encuentra un *outfit* que maximice esta cualidad.

$$O^* = \arg\max_{O_j \in \mathcal{O}} s(O_j) \tag{2.1}$$

Dada esta definición general, la evaluación de la composición del *outfit* se realiza típicamente a través de la predicción de $s(O)$. Además, se pueden incluir diferentes objetivos en la función incorporando conocimientos como relaciones complementarias (por ejemplo, cuánto complementa una camisa blanca a unos pantalones azules) o la similitud (cuánto se parece visualmente un artículo del conjunto a otro artículo).

2.2.3. Métodos habituales

Filtrados colaborativos con integración de información visual

Utilizan técnicas avanzadas de procesamiento de imágenes y aprendizaje automático para mejorar la precisión y relevancia de las recomendaciones. Métodos como VBPR (*Visual Bayesian Personalized Ranking*), DeepStyle, ACF (*Attentive Collaborative Filtering*) y DVBPR (*Deep* VBPR) incorporan características visuales extraídas de imágenes de productos mediante redes neuronales convolucionales (CNNs) para personalizar las recomendaciones según las preferencias visuales de los usuarios. Estos métodos superan a modelos tradicionales, como el BPR, al incorporar factores visuales mientras reducen su dimensionalidad para gestionar grandes volúmenes de información de manera eficiente. Además, consideran tanto la categoría como el estilo de los artículos, y en algunos casos, como en ACF, integran redes de atención para capturar los niveles de interés que los usuarios muestran hacia diferentes componentes del contenido multimedia. En conjunto, estas técnicas permiten abordar la naturaleza estética de los artículos y mejorar la experiencia de recomendación haciéndola más precisa y personalizada.

Modelos Generativos de Recomendación de Moda

Emplean tecnologías avanzadas como las GANs (*Generative Adversarial Networks*) para generar y recomendar prendas de vestir de manera realista y coherente. Ejemplos notables incluyen CRAFT (*Complementary Recommendation Using Adversarial Feature Transform*), que utiliza un proceso similar a un GAN para recomendar prendas complementarias basadas en características visuales, y DVBPR, que integra GANs en un marco de aprendizaje *end-to-end* para generar imágenes de moda personalizadas según las preferencias del usuario. MrCGAN (*Metric-regularized Conditional* GAN) y c^+GAN (*complementary* GAN) también destacan al emplear GANs para generar imágenes de prendas compatibles, evaluadas mediante encuestas de usuarios en línea. Estos modelos permiten explorar, inspirando tanto a clientes como diseñadores a través de recomendaciones visualmente atractivas y personalizadas.

2.3. MINERÍA DE REGLAS DE ASOCIACIÓN

Una regla de asociación es una implicación del tipo $A \cap B$ que se utiliza para descubrir relaciones comunes entre entidades en grandes bases de datos.

Se han investigado ampliamente diversos métodos para el aprendizaje de reglas de asociación resultando en el concepto de regla fuerte que permite descubrir regularidades de compra entre productos en datos de transacciones.

2.3.1. Métricas

Para seleccionar las reglas de asociación más interesantes del conjunto de todas las reglas posibles, se utilizan las siguientes métricas:

- **Soporte** (*support*): indicación de la frecuencia con la que aparece la combinación de los productos A y B en el conjunto de datos.

$$supp = P(A \cap B) = \frac{\text{número de transacciones que contienen A y B}}{\text{número total de transacciones}}$$

- **Confianza** (*confidence*): porcentaje de todas las transacciones que conteniendo A también contengan B

$$conf(A \Rightarrow B) = P(B|A) = \frac{supp(A \cap B)}{supp(A)} = \frac{\text{número de transacciones que contienen A y B}}{\text{número de transacciones que contienen A}}$$

- ***Lift***: grado en que los productos A y B dependen uno del otro.

$$lift(A \Rightarrow B) = \frac{supp(A \cap B)}{supp(A)\ supp(B)}$$

  Si $lift < 1$ indica que A y B son sustitutos el uno del otro.

2.3.2. Algoritmo apriori

El principio por el que se guía el algoritmo apriori [16] afirma que si un conjunto de elementos es frecuente, entonces todos sus subconjuntos también deben ser frecuentes .

Es decir, supongamos que $\{Bread,\ Diapers,\ Milk\}$ es un conjunto de elementos frecuente. Todas las transacciones que contienen $\{Bread,\ Diapers,\ Milk\}$ también contienen $\{Bread,\ Diapers\}$, $\{Bread,\ Milk\}$, $\{Diapers,\ Milk\}$, $\{Bread\}$, $\{Diapers\}$ y $\{Milk\}$. Por lo tanto, estos elementos también deben ser frecuentes para que $\{Bread,\ Diapers,\ Milk\}$ sea frecuente. Por otro lado, si un conjunto de elementos, supongamos $\{Cola,\ Eggs\}$, es infrecuente, entonces todos sus superconjuntos también deben ser infrecuentes, lo que significa que todas las transacciones, por ejemplo, $\{Cola,\ Eggs, Bread\}$, $\{Cola,\ Eggs,\ Diapers\}$, etc., también deben ser infrecuentes. Por lo tanto, todas las transacciones que contienen $\{Cola,\ Eggs\}$ pueden ser eliminadas inmediatamente.

Usando este argumento a la inversa es cómo se lleva a cabo la minería de datos. Consideramos la lista de elementos mostrados en la figura 2.1

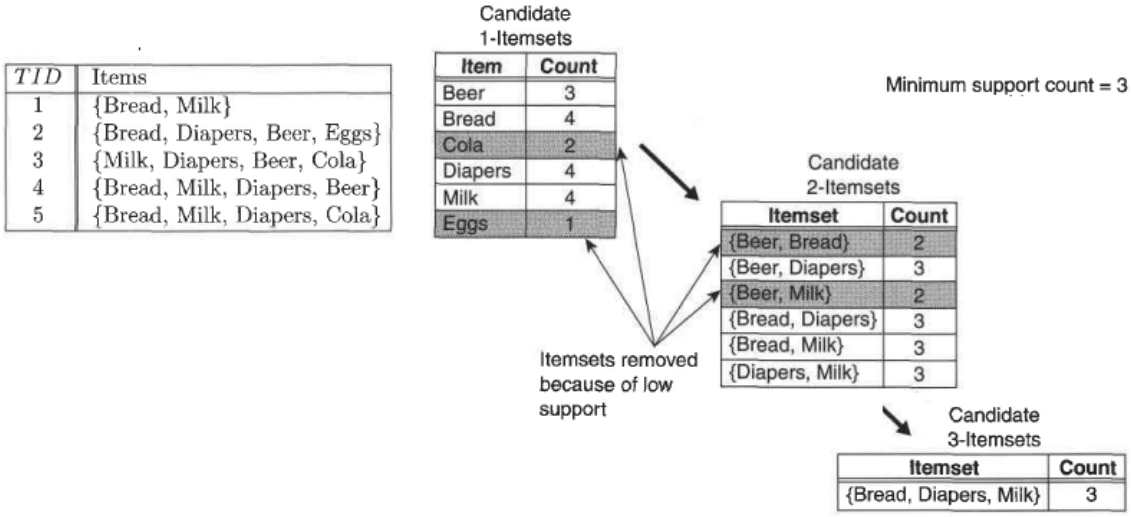


Figura 2.1: Ejemplo del uso del algoritmo apriori en una lista de la compra. Figura extraída de [16]

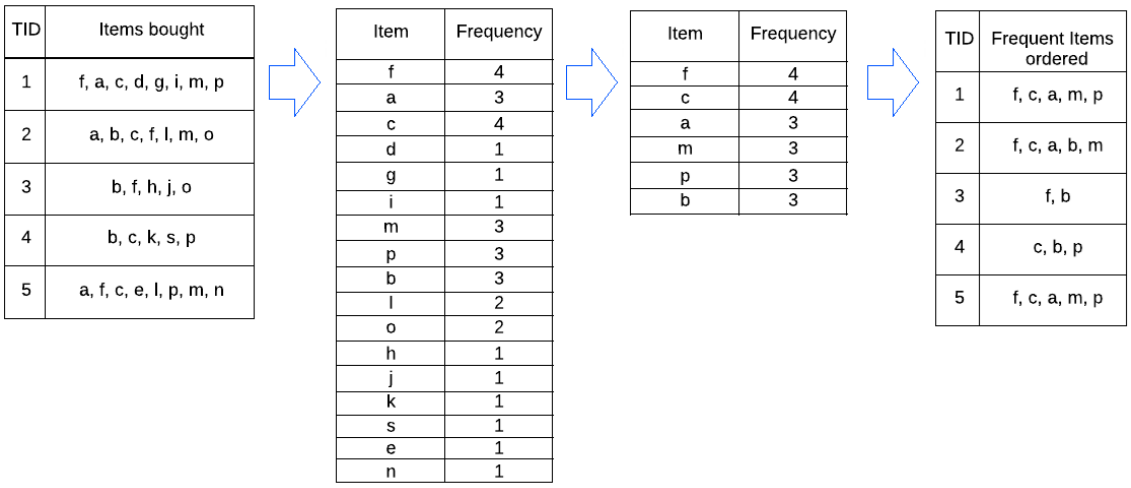
Supongamos que el valor de umbral mínimo es 3. Entonces, los conjuntos de elementos con una frecuencia de 3 o mayor que 3 se consideran frecuentes. Empezando por la agrupación de solo un elemento se observa que pueden eliminarse $\{Cola\}$ y $\{Eggs\}$ por infrecuentes, reduciendo la lista de elementos agrupados en pares de quince a seis conjuntos. Si repetimos el proceso y eliminamos $\{Beer, Bread\}$ y $\{Beer, Milk\}$ por infrecuentes reducimos a un único conjunto la lista de elementos agrupados en tríos.

Sin embargo, incluso después de mermar considerablemente la lista de posibilidades, el algoritmo Apriori para un conjunto de datos con una gran cantidad de elementos frecuentes o con valores de frecuencia bajos requiere mucha memoria para almacenarse, volviéndose lento e ineficiente. Esto es especialmente notable cuando la capacidad de la memoria es limitada y el número de transacciones es grande. Para hacer frente a estos problemas se propone el algoritmo FP-Growth (*Frequent Pattern Growth*).

2.3.3. Algoritmo FP-Growth

Frecuent Pattern Tree

Para poder entender el modelo FP-Growth primero hay que explicar el FP-Tree (*Frequent Pattern Tree*) [15]. Supongamos que existe un conjunto de datos como el de la figura 2.2: hay que eliminar las frecuencias inferiores a 3 (en este caso) y ordenar los restantes en orden descendente. Gracias a esto, los artículos remanentes en cada transacción pueden ser ordenados de manera adecuada.



TID	Items bought
1	f, a, c, d, g, i, m, p
2	a, b, c, f, l, m, o
3	b, f, h, j, o
4	b, c, k, s, p
5	a, f, c, e, l, p, m, n

Item	Frequency
f	4
a	3
c	4
d	1
g	1
i	1
m	3
p	3
b	3
l	2
o	2
h	1
j	1
k	1
s	1
e	1
n	1

Item	Frequency
f	4
c	4
a	3
m	3
p	3
b	3

TID	Frequent Items ordered
1	f, c, a, m, p
2	f, c, a, b, m
3	f, b
4	c, b, p
5	f, c, a, m, p

Figura 2.2: Ejemplo de cálculo de transacciones en un FP-Tree. Figura extraída de [15]

De los resultados obtenidos se construye en cada iteración el árbol FP por cada transacción. Se inicia con un nodo raíz y se hace crecer una rama por cada transacción, asignando a cada elemento un contador. De esta manera, en cada iteración se actualizan los contadores con la frecuencia total resultante de la combinación de ramas, como se muestra en la figura 2.3:

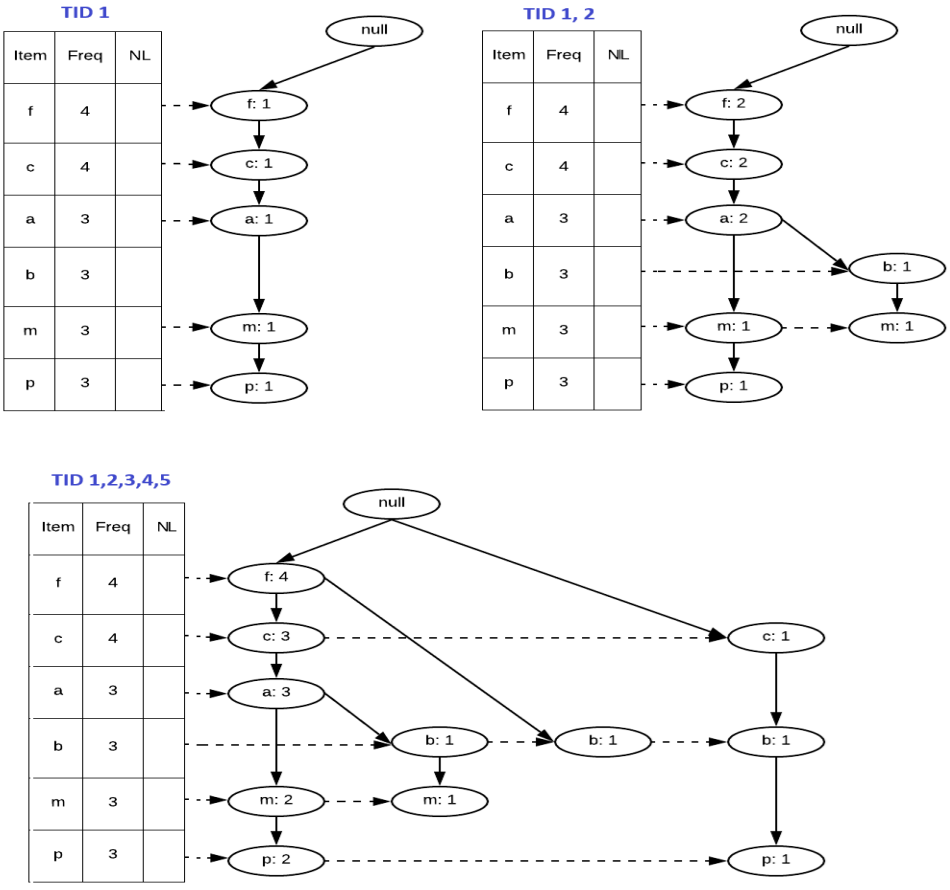


Figura 2.3: Ejemplo la creación de un árbol FP. Figura extraída de [15]

Algoritmo FP-Growth

FP-growth es un algoritmo que genera conjuntos de elementos frecuentes a partir de un árbol FP explorándolo de abajo hacia arriba [14, 17]. Es decir, en primer lugar encontrará todos los elementos frecuentes que terminen en p, luego m, b, a, c y finalmente f. La ruta del árbol que termina en el elemento frecuente dado es denominada como la base de patrones condicionados (*conditional pattern base*).

Ahora, para cada elemento, construiremos un árbol FP condicional. Se calcula identificando el conjunto de elementos comunes en todos los caminos de la base de patrones condicionados de un elemento frecuente dado y calculando su recuento como se muestra en la figura 2.4:

Item	Conditional Pattern Base	Conditional FP Tree	Frequent itemset
p	{f,c,a,m : 2}, {c,b: 1}	{c :3}	{c,p: 3}
m	{f,c,a : 2}, {f,c,a,b : 1}	{f,c,a : 3}	{f,m : 3}, {c,m : 3}, {a,m: 3}, {f,c,m: 3}, {f,a,m: 3}, {c,a,m: 3}, {f,c,a,m: 3}
b	{f,c,a: 1}, {f: 1}, {c: 1}		
a	{f,c: 3}	{f,c: 3}	{f,a: 3}, {c,a: 3}, {f,c,a: 3}
c	{f: 3}	{f: 3}	{f,c: 3}
f			

Figura 2.4: Ejemplo la creación de un árbol FP condicional. Figura extraída de [14, 17]

Es a partir del árbol FP condicional que se pueden generar los conjuntos de artículos frecuentes. Esta estructura permite que los patrones frecuentes se extraigan directamente sin necesidad de generar y contar candidatos repetidamente como en el algoritmo apriori, además, la estructura de árbol es más eficiente en términos de memoria. No solo eso sino que reduce el número de escaneos de la base de datos a dos.

Parallel FP-Growth

Todos los pasos en FP-Growth pueden ser paralelizados [19] . Así la base de datos se divide en varias particiones que se procesan en paralelo. Cada partición se utiliza para construir un FP-tree local. De cada FP-tree local se extraen patrones frecuentes de los cuales se generan sub-árboles condicionados. Por último, de todos los sub-árboles condicionados pertenecientes a las diferentes particiones se extraen sub-conjuntos de artículos frecuentes que se combinan para formar el conjunto global de artículos frecuentes.

Este último paso no es trivial dado que la división de la base de datos afecta la extracción de patrones frecuentes al alterar las frecuencias. Para solventar este problema, primero se realiza un escaneo inicial para determinar la frecuencia global de cada elemento en la base de datos completa. De esta forma, los patrones extraídos de los árboles FP locales están condicionados por los elementos frecuentes globales. Esto permite que los FP-trees condicionales pueden ser fusionados o sus resultados combinados para obtener las frecuencias reales mediante técnicas de reducción paralela.

El algoritmo *Parallel FP-Growth* nos permite trabajar en Spark con bases de datos masivas. Apache Spark es un framework de programación para elm procesamiento de datos distribuidos diseñado para ser rápido y de propósito general.

2.4. RECOMENDACIÓN BASADA EN GRAFOS

2.4.1. Qué es un grafo

Un grafo es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos. Permiten representar interacciones entre elementos de un conjunto y se representan visualmente como se indica en la figura 2.5:

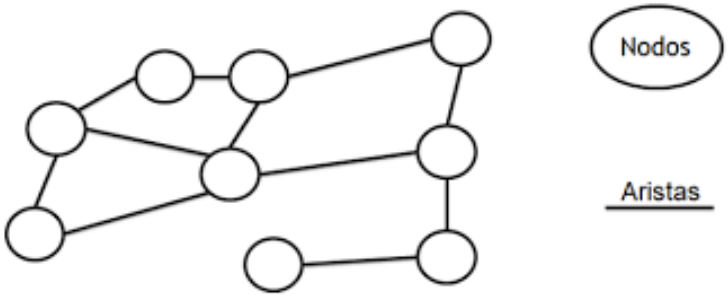


Figura 2.5: Ejemplo visual de la definición de grafo. Imagen extraída de [7].

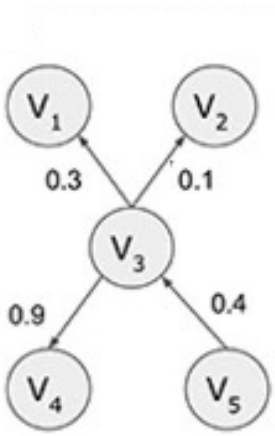
Hay dos categorías principales de grafos:

- **Grafo no dirigido**: la interacción entre elementos es simétrica, es decir, la interacción se realiza entre ambos nodos.

- **Grafo dirigido**: la interacción entre elementos es asimétrica, es decir, posee un sentido de dirección. La interacción se realiza de un nodo hacia otro.

Figura 2.6: Ejemplo visual de grafos dirigido y no dirigido. Imagen extraída de [3]

La representación matemática del grafo es la matriz de adyacencia, una matriz de tamaño $n \times n$ donde las filas y las columnas hacen referencia a los vértices para almacenar en cada casilla la longitud de la arista que los une como se indica en la figura 2.7:



	V_1	V_2	V_3	V_4	V_5
V_1	0	0	0	0	0
V_2	0	0	0	0	0
V_3	0.3	0.1	0	0.9	0
V_4	0	0	0	0	0
V_5	0	0	0.4	0	0

Figura 2.7: Ejemplo de como calcular la matriz de adyacencia dado un grafo dirigido ponderado. Imagen extraída de [12]

2.4.2. Algoritmos basados en teoría de grafos

Los algoritmos basados en grafos son fundamentales para mejorar la funcionalidad de los sistemas de recomendación y el análisis de redes. Estos se pueden agrupar de manera general en dos grupos principales.:

- **Detección de comunidades**: tienen como objetivo dividir una red en *clústeres* o comunidades donde los nodos dentro de la misma comunidad estén más densamente conectados entre sí que con los de otras. Un método particularmente famoso para conseguir esto es el **Método Louvain** [27, 11]. Funciona optimizando iterativamente la modularidad, una medida de la densidad de vínculos dentro de las comunidades en comparación con los vínculos entre comunidades. Se distribuyen los nodos iterativamente de tal manera que se genere el mayor aumento de modularidad hasta que se alcanza un máximo.

- ***Embedding* de grafos**: funcionan transformando los nodos a un espacio vectorial de baja dimensionalidad que preserva las propiedades del grafo, facilitando su aplicación en algoritmos de aprendizaje automático. Así es como trabaja **DeepWalk** [26, 18]: genera caminos aleatorios a partir de cada nodo del grafo, es decir, crea 'frases' donde los nodos son 'palabras' que el algoritmo **Word2Vec** puede procesar, logrando predecir los patrones de la red.

2.4.3. Detección de cliques y el algoritmo de Bron-Kerbosch

Un clique es un conjunto de nodos tal que todo par de vértices distintos son adyacentes, es decir, existe una arista que los conecta. En otras palabras, un clique es un subgrafo en el que cada nodo está conectado a todos los demás vértices del subgrafo. De esta forma el clique máximo es aquel que no puede expandirse añadiendo un vértice más del grafo sin dejar de ser un clique.

Encontrar cliques, particularmente cliques máximos, dentro de un grafo es una tarea computacionalmente costosa debido a la naturaleza combinatoria del problema. El algoritmo **Bron-Kerbosch** [5] es un método ampliamente utilizado para este propósito, conocido por su eficiencia para encontrar todos los cliques máximos [8].

Bron-Kerbosch

Este algoritmo amplía de forma recursiva cliques potenciales agregando vértices uno a uno a la vez que poda el espacio de búsqueda usando reglas específicas para evitar cálculos innecesarios.

La forma básica del algoritmo Bron-Kerbosch es una función recursiva que toma como argumentos los conjuntos de vértices R, P y X. El conjunto R representa el clique actual que se está construyendo, P representa el conjunto de vértices candidatos a unirse a R y X es el conjunto de vértices excluidos. La forma en la que este algoritmo funciona es agregando un vértice a R por cada iteración siempre y cuando este vértice sea válido, y guardando los ya procesados en X. Es decir se inicializa con los conjuntos R y X vacíos mientras que P contiene todos los vértices del grafo. Como bien indica el pseudocódigo de la imagen 2.8, los pasos a seguir son los siguientes

```
algorithm BronKerbosch1(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    for each vertex v in P do
        BronKerbosch1(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

Figura 2.8: Pseudocódigo del algoritmo Bron-Kerbosch. Imagen extraída de [2]

1. **if**: Comprobar si los conjuntos P y X están vacíos. En caso afirmativo R es nuestro clique máximo.

2. $\mathbf{R} \cup \{\mathbf{v}\}$: Añadir un vértice de P a R

3. $\mathbf{P} \cap \mathbf{N}(\mathbf{v})$: Reducir P a solo los nodos adyacentes al vértice del paso anterior.

4. $\mathbf{X} \cap \mathbf{N}(\mathbf{v})$: Redefinir X como los vértices comunes entre X y los adyacentes.

5. **BK**: Repetir desde el paso 1.

Para mayor claridad vamos a ilustrar este algoritmo con el siguiente ejemplo:

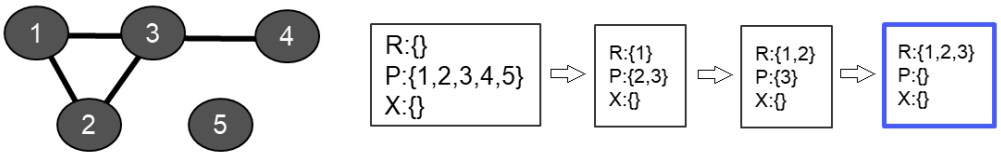


Figura 2.9: Ejemplo de uso del algoritmo Bron-Kerbosch

Después de la inicialización se añade el vector 1 a R y se reduce P a sus vecinos adyacentes los cuales son $\{2, 3\}$ y como la intersección de un conjunto vacío con cualquier cosa es de nuevo el vacío X se queda tal cual. Iterando añadimos el vértice 2 a R y reducimos P a los vecinos adyacentes existentes, en este caso, solo el nodo 3 cumple esas características. El último paso es añadir 3 a R cumpliendo así la condición de P y X vacías que señalan el fin de la búsqueda de un clique máximo.

Bron-Kerbosch con vértice pivote

La forma básica del algoritmo descrita anteriormente es ineficiente en el caso de grafos con muchos cliques no máximos, ya que se realiza una llamada por cada clique, sea máximo o no.

Para mejorar la eficacia del algoritmo se elige un vértice pivote u de P (o $P \cup X$) y se modifica su paso recursivo de forma que en lugar de probar recursivamente todos los vecinos de cada vértice en P, se centra en u y sus no vecinos. El razonamiento es que cualquier clique máximo que pueda formarse probando vecinos de u también se descubriría probando u o uno de sus no vecinos. Esta optimización reduce las comprobaciones redundantes.

```
algorithm BronKerbosch2(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch2(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

Figura 2.10: Pseudocódigo del algoritmo Bron-Kerbosch con vértice pivote. Imagen extraída de [2]

La diferencia principal es que para cada vértice pivote u de $P \cup X$ se itera sobre cada vértice v de P que no es vecino de u.

Para mayor claridad vamos a ilustrar este algoritmo con el siguiente ejemplo:

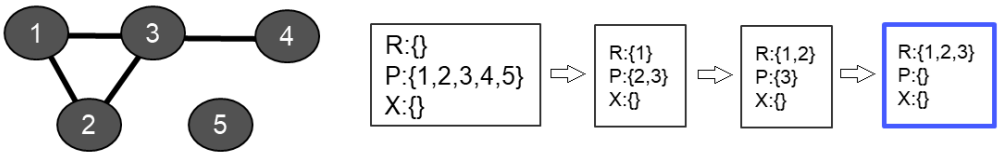


Figura 2.11: Ejemplo de uso del algoritmo Bron-Kerbosch con vértice pivote. Imagen extraída de [2]

Bron-Kerbosch con ordenación de vértices

El algoritmo de Bron-Kerbosch se puede mejorar eliminando el uso del pivote en el nivel más externo de la recursión. En su lugar, se elige cuidadosamente el orden en que se hacen las llamadas recursivas para minimizar los conjuntos de vértices candidatos en cada llamada recursiva.

En cada grafo existe un orden de degeneración, donde cada vértice tiene d o menos vecinos que vienen después de él en ese orden. La degeneración de un grafo G es el número más pequeño d tal que cada subgrafo de G tiene al menos un vértice con grado d o menos.

Cuando el algoritmo de Bron-Kerbosch recorre los vértices en un orden de degeneración, se garantiza que el conjunto P de vértices candidatos en cada llamada recursiva (los vecinos del vértice actual que vienen después en el orden) será pequeño, con un tamaño máximo de d. El conjunto X de vértices excluidos estará formado por todos los vecinos anteriores del vértice actual, que puede ser mucho mayor que d.

En las llamadas recursivas a niveles inferiores de la recursión, todavía se puede utilizar la técnica del pivote para optimizar la búsqueda de cliques máximos.

En resumen, se utiliza un orden de degeneración en el algoritmo de Bron-Kerbosch que ayuda a reducir el tamaño de los conjuntos de vértices candidatos en cada llamada recursiva, como se indica en el pseudocódigo 2.12:

```
algorithm BronKerbosch3(G) is
    P = V(G)
    R = X = empty
    for each vertex v in a degeneracy ordering of G do
        BronKerbosch2({v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

Figura 2.12: Ejemplo de uso del algoritmo Bron-Kerbosch con vértice pivote. Imagen extraída de [2]

2.5. EMBEDDINGS DE IMÁGENES Y SIMILITUD COSENO

2.5.1. Embeddings de imágenes

Definición de *embedding*

El *embedding* de una imagen es la codificación vectorial de esa imagen. Sin embargo, hay que especificar que no contiene la misma información, es decir, va más allá de los números existentes en cada pixel representando la información del color en RGB [13]. Un *embedding* también dará información de qué hay contenido en la imagen gracias al entrenamiento de clasificación con imágenes y texto a los que se someten los modelos previamente. De esta forma en el espacio vectorial los *embeddings* de, por ejemplo, dos camisetas serán más cercanos que los de una camiseta y unos zapatos aunque las camisetas no se parezcan entre sí a nivel visual.

Métodos de generación de *embeddings*

Varias de las técnicas más prominentes actualmente son:

- **Redes Neuronales Convolucionales (CNNs)** [20]: son un tipo de arquitectura de redes neuronales profundas (*Deep Learning*), compuestas por capas que aprenden de forma automática. Principalmente son la combinación de: a) una capa convolucional para extraer características a través de aplicar un filtro o kernel que devuelva un mapa de activación donde si indique si la característica que buscaba se encontraba en la zona analizada o no; b) una capa de agrupamiento o *pooling* para reducir el número de parámetros quedándose solo con el valor máximo por zonas del mapa de activación; y c) una capa completamente conectada para clasificar los datos en categorías mediante transformaciones lineales aplicadas por cada neurona o perceptrón de la red. Dentro de las CNNs destaca la **ResNet50** con 50 capas distribuidas de tal manera que soluciona el problema del gradiente además de contar con atajos para saltarse capas.

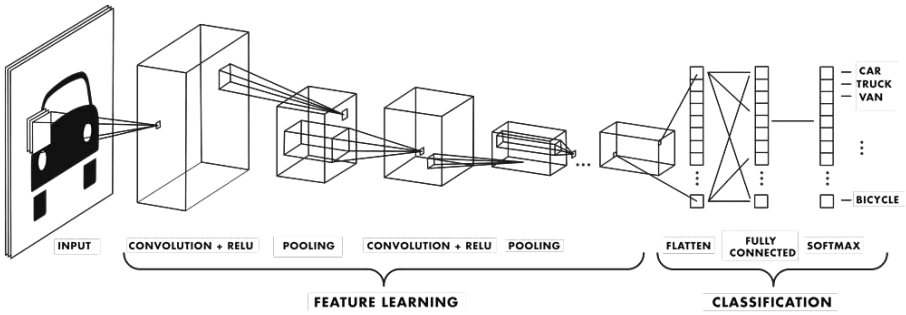


Figura 2.13: Ejemplo del funcionamiento interno de una CNN. Imagen extraída de [**?**]

- **Transformadores de visión** [25]: un ViT es es una arquitectura de aprendizaje profundo que adapta los principios de los transformadores o *transformers*, originalmente desarrollados para el procesamiento de lenguaje natural (*Natural Language Processing* o NLP). En NLP, el texto se divide en fragmentos conocidos como tokens. Luego, se asignan índices enteros de manera arbitraria pero única a cada token, y finalmente, se asocia un embedding al índice entero.

  El siguiente paso es que el mecanismo de autoatención simule cómo funciona la atención humana asignando diferentes valores de importancia a diferentes palabras en una oración. Estipula la relevancia de cada palabra calculando pesos "suaves" para cada embedding dentro de una sección específica de la oración llamada ventana de contexto. A diferencia de los pesos "duros", que están predeterminados y fijos durante el entrenamiento, los pesos "suaves" pueden adaptarse y cambiar con cada uso del modelo. Esta contextualización sirve para excluir datos irrelevantes de ser considerados y para amplificar aquellos tokens clave. Se realiza en las capas de codificación-decodificación que consisten en capas alternas de atención y feedforward.

  Con el mismo enfoque, un ViT descompone una imagen de entrada en una serie de píxeles (en lugar de dividir el texto en tokens), serializa cada píxel en un vector y lo mapea a una dimensión más pequeña con una única multiplicación de matrices [23]. Estos embeddings vectoriales luego son procesados por un codificador de transformer como si fueran embeddings de tokens.

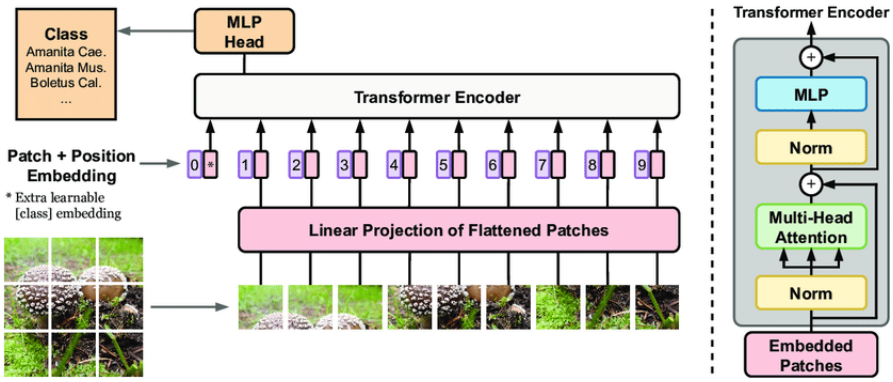


Figura 2.14: Ejemplo del funcionamiento interno de una ViT. Imagen extraída de [21]

- **Aprendizaje por transferencia** [22]: implica aprovechar modelos previamente entrenados en grandes conjuntos de datos para adaptarse a nuevas tareas con conjuntos de datos potencialmente más pequeños. Destacan plataformas como TensorFlow

Hub, PyTorch Hub, Keras Applications y Hugging Face con modelos robustos pero este último ofrece además un repositorio intuitivo de usar, centrado en NLP y ViTs, con facilidad para cargar modelos y una comunidad activa, convirtiéndolo en la plataforma más usada actualmente en la industria.

2.5.2. Métricas para la similitud

- **Producto escalar**: se define como $\vec{u} \cdot \vec{v} = ||u||||v|| \cos\theta$. Depende de las magnitudes de los vectores. Si estamos trabajando en un sistema de recomendación, donde la longitud de los vectores es importante, la similitud del producto escalar es la mejor a elegir.

- **Similitud coseno**: es define a partir del producto escalar ya que es el coseno del ángulo comprendido entre dos vectores $\cos\theta = (\vec{u} \cdot \vec{v})/(||u||||v||)$. No depende de las magnitudes de los vectores, solo del ángulo entre ellos. Dos vectores con el mismo ángulo entre ellos tendrán la misma similitud de coseno independientemente de sus magnitudes. Si estamos trabajando en un espacio de alta dimensionalidad, con embeddings normalizados, la similitud por coseno es la que mejor funciona. Además su rango de valores solo se encuentra en [-1,1] mientras que el producto escalar puede resultar en cualquier número real.

- **Distancia euclídea**: es la distancia en línea recta entre dos puntos que se calcula con el teorema de Pitágoras $d(P_1, P_2) = \sqrt{(x_2 – x_1)^2 + (y_2 – y_1)^2}$. Si estamos trabajando en un sistema de clasificación, donde la distancia entre dos clases es importante, la similitud por distancia euclídea será la que mejores resultados nos de.

Capítulo 3

Diseño e Implementación

3.1. DISEÑO

3.1.1. Flujo de Trabajo

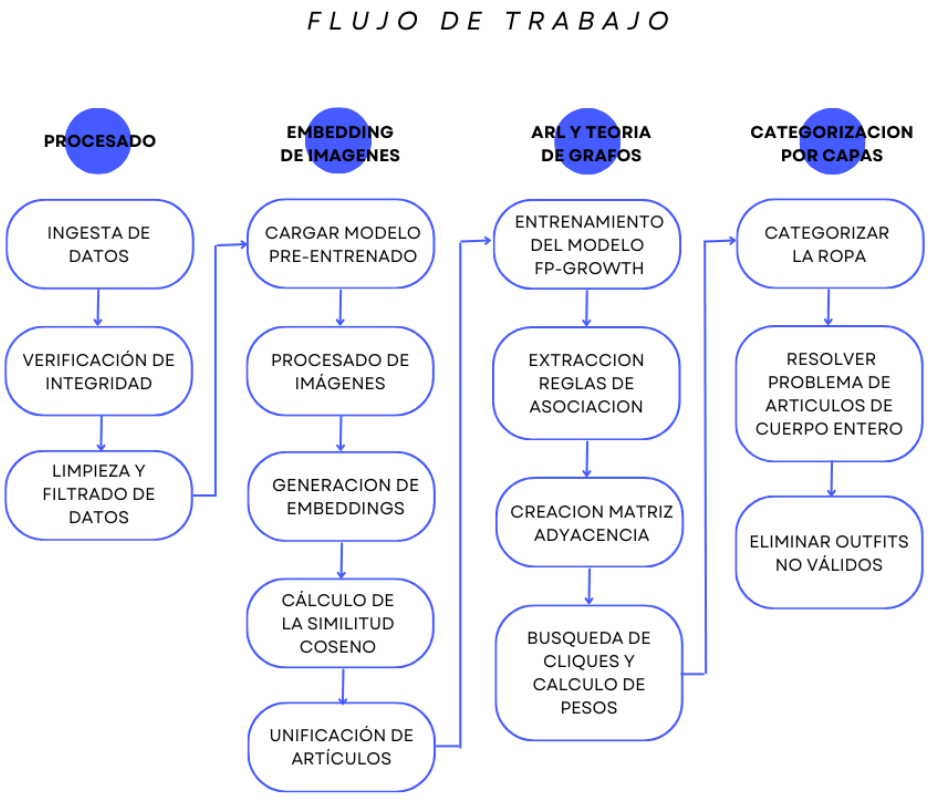


Figura 3.1: Diagrama del flujo de trabajo

El desarrollo de este proyecto se ha dividido en cuatro etapas. En primer lugar, se procesaron los datos enfrentándonos a un data lake con con información proveniente de distintas fuentes en momentos diferentes, lo que generó inconsistencias o valores nulos. Sin embargo, la limpieza de los datos no concluyó aquí; también se empleó el procesamiento de imágenes para identificar todos aquellos artículos que, siendo el mismo elemento, se

encontraban bajo diferentes identificadores. Es importante corregir estos errores, dado que la siguiente etapa requiere utilizar la frecuencia de compra para encontrar las reglas de asociación entre los diferentes artículos de ropa.

Una vez encontradas y calculadas las métricas de *support*, *confidence* y *lift* asociadas, obtenemos información cuantitativa sobre las relaciones entre pares de elementos. Con estos datos, podemos crear una matriz de adyacencia para encontrar los cliques máximos gracias a la teoría de grafos.

Por último, estos resultados nos llevan a la etapa final, que consiste en categorizar los grupos encontrados en partes de arriba, abajo, calzado, abrigo y complementos. Con estas categorías, podemos identificar qué cliques forman un outfit válido, excluyendo aquellos que solo contienen artículos de una sola categoría o que carecen de al menos una prenda de arriba y abajo.

3.1.2. Fuentes de datos

El conjunto de datos que se ha utilizado para este proyecto está sacado de las bases de datos en Snowflake. Principalmente se han usado:

- Un registro de todos los artículos vendidos desde el 2015 ocupando 1.6 TB en una tabla de 19.8 billones de filas por 148 columnas. Hay mucha información almacenada pero para lo que nos concierne solo nos vamos a centrar en las columnas donde se archivan todos los identificadores de los artículos vendidos y sus correspondientes tickets de compra. Esto incluye cualquier país donde se haya producido la transacción y todas las cadenas pertenecientes al grupo Inditex. Como solo nos interesan las prendas que se pueden comprar juntas se ha acotado este dataset a Zara (una de las cadenas comerciales de Inditex) desde octubre del 2022 hasta el abril del 2024 en España. Cada ticket individual se genera agrupando la localización, la fecha, la caja y el tipo de ticket para formar así un identificador único y anónimo. Esto quiere decir que vamos a operar con tickets de compra aislados y no en relación al historial personal de ningún usuario. Cada uno se considerará independiente del anterior.

- Un registro de la parte comercial de todos los artículos existentes ocupando 492.9 MB en una tabla de 6.9 millones de filas por 47 columnas. De esta tabla solo nos interesa la columna de descripción del producto, tanto para aplicar filtros como para poder realizar un análisis multimodal al agregar texto además de imágenes en futuras pruebas.

- Un registro de la parte visual de todos los artículos existentes en Zara ocupando 151.2 MB en una vista de 1.5 millones de filas por 28 columnas. De esta tabla sobre todo nos interesan los *url* estáticas de las imágenes pero también el modelo, el color y la familia.

La primera tabla mencionada es aquella sobre la cual se basará nuestro algoritmo, ya que se fundamenta en la frecuencia con la que los artículos se compran juntos. Sin embargo, igual de necesarias son las otras dos tablas para asegurarnos de que los resultados finales sean recomendaciones de *outfits* válidos y coherentes.

3.1.3. Descripción del entorno de trabajo

Se ha trabajado en la PaaS (*Platform as a Service*) Databricks debido a que permite el uso de Spark, es compatible con Snowflake y además gestiona la conexión con los clústeres.

Este proyecto ha requerido el uso de dos clústeres: uno principal para casi la totalidad del algoritmo y otro exclusivamente solo para el procesado de imágenes.

Clúster principal

Figura 3.2: Esquema de las características del clúster principal

Es decir, se está usando un entorno de ejecución *Long Term Support* (LTS) con la versión 3.4.1 de Apache Spark. Para el *driver* se ha especificado la máquina virtual tipo *Standard_DS4_v2*, que cuenta con 28 GB de memoria RAM y 8 núcleos de CPU, para controlar la ejecución del programa principal y coordinar la distribución de las tareas a los *workers*. A su vez, para los *workers* se ha especificado la máquina virtual tipo *Standard_DS3_v2*, que cuenta con 14 GB de memoria RAM y 4 núcleos de CPU.

Clúster para el procesado de imágenes

Figura 3.3: Esquema de las características del clúster usado para el procesado de imágenes

Es decir, se está usando un entorno de ejecución *Long Term Support* (LTS) optimizado para *Machine Learning* (ML) soporte para unidades de procesamiento gráfico (GPU). Para el *driver* se especifica la máquina virtual tipo *Standard_N8as_T4_v3* que cuenta con una GPU *Nvidia T4*, con 56 GB de memoria RAM y una CPU con 8 núcleos de procesamiento.

3.1.4. Librerías externas utilizadas

Para el desarrollo del proyecto se han utilizado varias librerías externas con el fin de ahorrar tiempo. A parte de librerías base como lo son **numpy**, **pandas**, **torch** o **pyspark** también se han usado:

- **SnowflakeUtils**: para poder ejecutar instrucciones o *querys* SQL con el método *executeQuery*. Necesario para extraer dataframes de las bases de datos existentes en SnowFlake al notebook de Databricks donde estemos trabajando.

- **pyspark.ml.fpm**: para poder importar un modelo FP-Growth creado para paralelizarse. Distribuye la tarea de minería de datos de forma que cada trabajador ejecuta un grupo de artículos independientes.

- **transformers**: para importar tanto la clase *CLIPModel*, que representa el modelo CLIP en sí, un modelo ya desarrollado capaz de entender imágenes, como la clase *CLIPProcessor*, que se utiliza para preprocesar los datos antes de pasarlos al modelo. Se utiliza porque permite cargar un modelo pre-entrenado desde la plataforma de HuggingFace.

- **networkx**: para poder utilizar las funciones *from_pandas_adjacency* y *find_cliques*. La primera regresa un grafo al introducir un dataframe y la segunda utiliza el algoritmo de Bron-Kerbosch para encontrar los cliques del grafo.

- **IPython.display**: para poder renderizar los resultados como HTML, ya que es la forma más rápida de visualizar las imágenes dentro del propio notebook, ahorrando así una gran cantidad de tiempo y espacio de memoria.

En un segundo plano nos hemos beneficiado de las funciones *combinations* de **itertools** para el cálculo de los pesos de los cliques y *cosine_similarity* de **sklearn.metrics.pairwise** para el cálculo de la matriz similitud coseno.

3.2. IMPLEMENTACIÓN

3.2.1. Descripción del script

En esta sección se va a hablar de forma más tangible pero resumida de cómo se avanza de una etapa a la siguiente en el diagrama de trabajo.

- **Procesado**: computacionalmente hablando lo que nos interesa del procesado de las bases de datos es extraer dos tablas. La primera es aquella sobre la que aplicaremos análisis de imagen y minería de datos. Esta tabla se obtiene del registro de todos los artículos vendidos y contiene una columna con todos los artículos comprados, asociados en otra columna con el código identificativo de cada ticket. La segunda tabla se obtiene de la unión del registro comercial con el registro visual, y abarca todas las características relevantes de cada artículo individualmente. Esta última, a la que denominaremos master de aquí en adelante, no tiene un propósito concreto más allá de almacenar la URL de la foto, el modelo, la calidad, el color, la familia y la descripción de cada elemento en una tabla maestra organizada y limpia para un uso cómodo en futuras operaciones.

- **Embedding de imágenes y similitud coseno**: dado que el cálculo de los embeddings de imágenes es costoso, optimizamos el proceso enfocándonos únicamente en los productos vendidos. Primero, filtramos los artículos duplicados de la tabla obtenida previamente y asociamos cada elemento con su imagen utilizando la columna URL de la tabla master. Luego, estos datos se introducen en nuestro transformador de visión para obtener los embeddings necesarios para calcular la similitud coseno. Por último, obtenemos el resultado real de esta etapa que es un diccionario donde cada clave agrupa elementos con una similitud aproximada de 1, indicando que son esencialmente el mismo artículo. Este diccionario nos permite renombrar en la tabla inicial y la master todos los productos que aparentaban ser distintos bajo un mismo nombre, el nombre de la clave.

- **ARL**: Una vez renombrados todos los artículos procedemos a agrupar todos los elementos pertenecientes a una misma transacción ayudándonos del código identificativo de cada ticket de compra. Después, aplicamos el método FP-Growth a estos datos, obteniendo una tabla que incluye una columna para cada artículo antecedente y otra para cada producto consecuente, así como las métricas de confianza (*confidence*), soporte (*support*) y *lift*.

- **Teoría de grafos**: Teniendo una medida cuantitativa de la relación a pares entre artículos podemos crear la matriz de adyacencia con los datos obtenidos de la métrica

lift, rellenando con ceros las relaciones sobre las que no tenemos información. Usamos la librería networkx para calcular los cliques máximos, generando un dataframe con dos columnas: una que contiene arrays con los elementos de cada clique y otra que muestra el peso asociado a cada clique. Para poder hablar de *outfits* subdividimos la columna array en las siguientes categorías: parte de arriba (abrigos, chaquetas, sudaderas,camisetas, etc), parte de abajo (pantalones, faldas, etc), cuerpo entero (vestido, traje y mono) y extras (calzado, accesorios y ropa interior) con la ayuda de la propiedad 'familia' de la tabla máster. Pese a la necesidad de filtrar y resolver el problema que presentan las prendas de cuerpo entero este dataframe es ya el formato en el que se presentan los resultados finales.

A continuación se van a desarrollar en mayor profundidad cada una de estas etapas.

3.2.2. Tratamiento de datos

Tipo de *input*

El tipo de datos disponible determina el tipo de sistema de recomendación a utilizar. En nuestro caso, debemos descartar tanto los filtrados colaborativos como los basados en contenido porque no contamos con las calificaciones (*ratings*) de los usuarios sobre nuestros productos. Podríamos forzar una solución asumiendo una calificación máxima positiva para los artículos comprados y una calificación media para los devueltos. Sin embargo, dado que cada transacción está asociada a un código identificativo anónimo para cada ticket, incluso suponiendo que cada ticket pertenece a un usuario distinto, el perfil del usuario sería en el mejor de los casos de tamaño 20 frente a la inmensa variedad de productos ofertados. Esto se debe a que hemos impuesto un límite de 20 artículos comprados en el filtrado para asegurarnos de seleccionar solo particulares. Siguiendo con el tipo de sistema de recomendación a utilizar, también descartamos el resto de métodos habituales descritos en los modelos generativos ya que la mayoría se basan en imágenes y en las preferencias de usuario, mientras que nuestro enfoque pretende centrarse en el análisis de tickets de compra.

Comprobación

Vamos a trabajar con data lakes donde encontramos la información que necesitamos, pero estructurada de forma desorganizada en múltiples tablas y donde es fácil encontrarse con valores nulos o genéricos. Esto significa que, al relacionar primary keys, será necesario verificar las correspondencias debido a posibles discrepancias causadas por errores humanos o falta de consenso en los términos. Además, al eliminar las filas incompletas, será necesario realizar un estudio para discernir con buen criterio si el porcentaje de información eliminada es significativo, si se puede recuperar de algún modo y su impacto en los resultados finales. En conclusión, extraer datos requerirá de un análisis detallado y exhaustivo.

Procesado

Inditex guarda mucha información en sus bases de datos, desde la hora en la que se realizan las compras, hasta en qué caja se efectuó el pago, cuánto fue el dinero que se cobró, etc. No todo el contenido es relevante para nuestro algoritmo, por lo que debemos realizar una selección de los atributos que nos interesan.Dentro de estos aún tenemos que filtrar ya que, por ejemplo, los tickets de compra no siempre hacen referencia a ropa: también

pueden contener muebles, peluches o incluso tickets de ventas internas para las bolsas de la compra. Nuestro objetivo es obtener tablas limpias, actualizadas y categorizadas por grupos para analizar por separado.

3.2.3. Creación de embeddings y cálculo del coseno

Al extraer los embeddings de las imágenes y calcular la distancia entre vectores con el coseno nos damos cuenta de que existen elementos con distinto *id* que coinciden exactamente. Es decir, al volcar en las bases de datos la información, se han usado distintos nombres para referirse al mismo artículo. Esta repartición involuntaria del número de veces que se compran estos artículos es un problema directo para nuestro algoritmo basado en la frecuencia de compra. Usamos esta información para redefinir bajo un mismo nombre todos aquellos elementos que se consideren el mismo.

Embedding de imágenes

Para el análisis de imágenes se ha hecho uso del modelo pre-entrenado *clip-vit-base-patch32* de la plataforma *Hugging Face* que utiliza un transformador *ViT-B/32* como codificador de imágenes. Esta elección se ha hecho existiendo las siguientes variantes de modelos en la red neuronal CLIP: *ViT-B/16*, *ViT-L/14*, *RN50*, *RN101* o *ViT-H/14*. Hay un motivo: para tareas simples o que no requieren detalles muy finos, modelos como *ViT-B/16* o *ViT-L/14* podrían funcionar, de lo contrario, *ViT-B/32* o RN50 serían mejores opciones. En la generación de *embeddings*, donde la velocidad y la eficiencia de los recursos son fundamentales, *RN50* o *ViT-B/32* son buenas opciones, mientras que *ViT-B/16* y *ViT-L/14* son más adecuados para la recuperación de imágenes. Es por ello que optamos en nuestro proyecto por *clip-vit-base-patch32* con un modelo *ViT-B/32*. Este modelo se ha entrenado con los pies de foto de imágenes disponibles públicamente mediante el rastreo de un puñado de sitios web y el uso de conjuntos de datos de imágenes preexistentes de uso común, como YFCC100M. Garantizando así un entrenamiento con un dataset lo suficientemente grande y variado para obtener resultados fiables.

Coste computacional de calcular los embeddings

Teniendo en cuenta el volumen de datos (aproximadamente 47mil imágenes) el procesado es computacionalmente muy costoso. El problema principal es el tiempo de ejecución requerido y lo hemos sorteado procesando los datos en tandas (*batch*) de 100 elementos, guardándolos de manera incremental en archivos tipo *parquet*. Este formato de almacenamiento en columnas está optimizado para la eficiencia del almacenamiento. Guardar los datos de manera incremental significa que, en lugar de procesar y almacenar todos los datos de una vez, vamos añadiendo los nuevos datos a medida que se procesan, adjuntándolos a los datos ya existentes. Esto nos permite manejar grandes volúmenes de datos sin sobrecargar la memoria y facilita el procesamiento continuo y eficiente. Además, permite la interrupción de la ejecución sin conllevar la pérdida de los datos, ya que se puede continuar desde donde se dejó, ahorrando el tiempo de recalculación en caso de errores.

Similitud coseno

Teniendo en cuenta que estamos trabajando en un espacio de alta dimensionalidad y con vectores normalizados la métrica que mejor se adapta a nuestro caso es la similitud coseno. Importando de la librería *sklearn.metrics.pairwise* la función *cosine_similarity* se encarga de calcular la matriz de similitud. Para reducir el tiempo de iteración, como los

elementos que son en realidad el mismo artículo van a presentar un coseno cercano a uno van a manifestarse en tantas filas y columnas como número de elementos iguales haya, se convierte la matriz similitud en una matriz diagonal con la ayuda de la función *tril_indices* de *numpy*. Así cuando se obtienen los índices de los elementos iguales de la matriz similitud, es decir, los pares de artículos que cumplen la condición $> 0{,}99999$ solo obtenemos $[A, B]$ o $[B, A]$ pero no ambos solventando la redundancia. Se usa esa condición ya que el redondeo de los resultados puede traducirse en no conseguir un 1 exacto pasándose tanto por abajo como por arriba. Lo último por conseguir ahora es agrupar todos aquellos elementos que aparecen en más de una pareja bajo una misma clave en un diccionario. El nombre de esta clave se genera concatenando todos los identificadores que componen su array de valores.

Coste computacional de buscar los artículos con similitud aproximada 1

La matriz de similitud tiene aproximadamente 47,000 filas y 47,000 columnas. Para evitar provocar un error de memoria al tratar de obtener las posiciones de los valores aproximados a 1, hemos recurrido a la menor precisión del formato *float16* y a la eficiencia de la función *argwhere* de *numpy*. Por el redondeo ocurrido al convertir nuestros datos a formato *float16* nuestra condición pasa a ser 0,998.

3.2.4. Cálculo de las reglas de asociación

Extracción de patrones de compra: hay que aplicar técnicas de minería de datos como las reglas de asociación para extraer las combinaciones frecuentes de prendas y con ello identificar patrones y similitudes en los tickets de compra.

El cálculo de las reglas de asociación es muy fácil gracias a la librería *pyspark.ml.fmp* de donde importamos la función *FPGrowth*. Los atributos requeridos son: el nombre de la columna donde se encuentran los artículos agrupados por transacción, un valor mínimo de *support* que permita identificar cuando un conjunto de elementos se considera frecuente y el mínimo valor de *confidence* para generar una regla de asociación. De esta forma cualquier patrón que aparezca más de $minSupport \times$tamaño del conjunto de datos veces se calificará como conjunto de elementos frecuentes. Una vez entrenamos al modelo con nuestros datos el método *associationRules* devolverá un dataframe con los artículos hipótesis asociados a aquellos que representan la conclusión de las reglas de asociación, y las métricas *support*, *confidence* y *lift*.

3.2.5. Creación de la matriz grafo y búsqueda de cliques

Con las métricas obtenidas de las reglas de asociación, vamos a construir una matriz de adyacencia para identificar grupos de artículos que estén todos relacionados entre sí, es decir, cliques. Primero, creamos una lista con todos los artículos únicos obtenidos de las reglas de asociación. Luego, realizamos un *cross join* sobre esta lista, generando dos columnas que representan todas las combinaciones posibles del conjunto de datos. En este formato, podemos aplicar un *join* para incorporar la columna de la métrica deseada (por ejemplo, *lift*) y rellenar con ceros los pares de elementos que no tienen relación entre sí. Finalmente, utilizando las funciones *groupby* y *pivot*, transformamos estos resultados en una matriz para obtener la matriz de adyacencia deseada.

Grafo no dirigido

Debido a cómo se definen las reglas de asociación, si tenemos $A \Rightarrow B$, no necesariamente tenemos $B \Rightarrow A$. Esto se debe a que, para aparecer en nuestros resultados, ambos

artículos deben ser considerados frecuentes y, además, deben tener un alto porcentaje de compras conjuntas para ser considerados una combinación frecuente. Es fácil imaginar que la probabilidad de que una camisa llamativa se compre junto con unos vaqueros básicos sea alta, pero la proporción de veces que esos vaqueros se compran con esa camisa llamativa sea baja, ya que los vaqueros son un básico de armario que combina con muchos otros artículos, apareciendo así en combinación con una mayor variedad de productos. Esto hace que nuestro grafo sea dirigido. Sin embargo, el porcentaje de productos en nuestros resultados donde se cumple $A \Leftrightarrow B$ es muy alto. De manera más visual, observamos que un gran porcentaje de filas en nuestro dataframe con antecedente A y consecuencia B están seguidas de filas con antecedente B y consecuencia A, con aproximadamente los mismos valores de *support*, *confidence* y *lift*. Gracias a este comportamiento, podemos asumir que, en la práctica, nuestro grafo es no dirigido. Poder realizar esta aproximación nos permite usar la función *find_cliques* de la librería *networkx* ya que se basa en el método Bron-Kerbosch mencionado previamente.

Detección de comunidades o cliques

La mayor parte de la teoría de grafos se dedica a la detección de comunidades pero consideramos que aunque exista una combinación de artículos que puedan dar lugar a un *outfit* dentro de un clúster altamente conectado no implica que la combinatoria de este sea buena ya que la relación entre todos los elementos no es obligatoria. Esto podría dar lugar a *outfits* donde pese a que la camiseta y los zapatos estén muy estrechamente ligados a una pieza central, como pueden ser unos pantalones, no lo estén entre ellos dando lugar a una combinación disonante estéticamente. Esta es la razón por la que descartamos el uso de algoritmos habituales en la teoría de grafos como el método Louvain o el DeepWalk. Esa también es la misma razón por la que usamos el algoritmo de Bron-Kerbosch con ayuda de la librería *networkx* para encontrar los cliques.

Pesos de los cliques

Los pesos de los cliques no pueden basarse en las métricas de soporte, confianza y lift, ya que estas requieren conocer la frecuencia con la que todos los elementos del clique aparecen juntos en las transacciones, lo cual no necesariamente ocurre. Por ejemplo, consideremos un clique ABCD derivado de las transacciones AB, AC, AD, BC, BD y CD. El número de transacciones que contienen ABCD es nulo. No obstante, solo disponemos de estas métricas como valores cuantitativos de las relaciones entre los elementos del clique. Dada esta limitación, la decisión tomada para calcular los pesos ha sido combinar los valores individuales de las relaciones, a sabiendas de que al hacerlo perderemos el significado original de estas métricas. Aunque estas combinaciones no representen un concepto específico, seguirán indicando en cierto grado la fuerza de las conexiones dentro del clique. Para este proyecto, se ha optado por definir los pesos de la siguiente manera:

$$(\prod lift_n)^{1/n} = \left(\prod_{i=0}^{N_I[n], N_{II}[n]} \frac{supp(N_I[i] \cap N_{II}[i])}{supp(N_I[i])\ supp(N_{II}[i])} \right)^{1/n}$$

donde N es la lista tal que si los artículos que conforman un clique son ABCD la combinatoria de ellos es AB AC AD BC BD CD y n la longitud de N. Sobre el productorio se ha hecho un promedio geométrico para que aquellos cliques con mayor número de prendas no salgan beneficiados por cantidad en vez de calidad.

Al centrarnos en el uso de cliques la función $s(O)$ que indica que tan bien funciona el *outfit* se define en este caso como el peso del clique. Se calcula como la suma de todas las relaciones existentes dentro del propio clique. Maximizar esta cualidad (ecuación 2.1) es el objetivo final de nuestro recomendador de *outfits*.

3.2.6. El problema de las prendas de cuerpo entero

Antes de dar por finalizados los resultados hay que categorizar cada clique por partes de arriba, abajo, cuerpo entero y extras (calzado y complementos) para así: a) poder eliminar aquellos que solo pertenezcan a una sola familia, b) garantizar la coherencia en los *outfits* evitando uniones de prendas de cuerpo entero, como los vestidos, con partes de arriba o abajo, como camisetas o faldas.

Categorización

Hemos optado por la clasificación manual de la distintas prendas en categorías. No se ha optado por un enfoque más avanzado como puede ser entrenar un modelo de clasificación supervisada utilizando algoritmos de machine learning como Redes Neuronales porque la casuística es simple y el número total de familias distintas reconocidas en las bases de datos de Inditex no es demasiado alto y sirve para este propósito.

Las categorías que hemos elegido son:

- **Parte de arriba**: abrigo, gabardina, impermeable, cazadora, chaleco, chaqueta, sudadera, jersey, blusa, camisa, camiseta, polo, etc
- **Parte de abajo**: falda, bermudas, leggings, pantalon, peto y shorts.
- **Cuerpo entero**: vestido, mono y traje.
- **Extras**: bota, sandalia, zapato, tacón, deportivas, gafas, gorros, bolsos, cinturones, albornoz, calcetines, ropa interior, pijamas, bañador, etc

Outfits válidos

Vamos a considerar un *outfit* válido a aquel que contenga como mínimo una parte de arriba y una de abajo o una prenda de cuerpo entero junto con extras. Se definen estas reglas para poder emparejar camisetas con pantalones y vestidos con sandalias pero no cometer errores en los que los elementos se superpongan como al emparejar un mono con una falda o errores de incompletitud al emparejar una blusa con unos zapatos dejando la zona de las piernas sin recomendación.

Para conseguir este objetivo desdoblamos todas las filas que contienen una prenda de cuerpo entero en dos nuevas filas de manera que en una de ellas se encuentre el clique original sin el artículo de cuerpo entero y en la otra el clique original sin las prendas de las partes de arriba y abajo. Este paso extra nos permite aprovechar recomendaciones que en un primer momento presentan una combinación incompatible de elementos. Por último, filtramos por la definición de *outfit* válido para llegar a los resultados finales de este proyecto.

3.2.7. Evaluación

Los valores de los pesos de nuestros cliques son una herramienta esencial para evaluar el rendimiento de nuestro algoritmo. Valores más altos indican un mejor desempeño. Para

asegurar la eficacia de nuestro método, llevamos un registro histórico de las magnitudes obtenidas y comparamos constantemente los datos originales con aquellos obtenidos tras la aplicación de nuevos filtros y mejoras. Este enfoque nos permite identificar y cuantificar mejoras de manera precisa y sistemática.

Reconocemos que los resultados de este trabajo pueden tener una componente subjetiva. Para validar la efectividad de nuestros resultados, contamos con la colaboración del equipo de diseño de Inditex. La evaluación se realiza mediante un test sencillo, donde expertos en estilismo de moda comparan nuestros resultados con conjuntos generados de manera pseudoaleatoria. La opinión de estos estilistas de moda nos permite determinar la validez y calidad de los outfits generados, asegurando que nuestro algoritmo produce combinaciones de alta calidad y relevancia en el mundo de la moda.

Capítulo 4

Resultados

4.1. DESCRIPCIÓN Y ANÁLISIS DE LA IMPLEMENTACIÓN

En esta sección vamos a analizar los resultados de la implementación en sus distintas etapas.

4.1.1. Procesado de datos

Del registro de todos los artículos vendidos queremos quedarnos solo con aquellos que puedan ser comprados juntos en una misma tienda. Es por ello que seleccionamos solo la cadena de Zara en España para mujeres. Además, establecemos un intervalo de tiempo desde enero de 2022 hasta abril de 2024 para considerar únicamente artículos actuales. Esta selección reduce a un 0,014 % la matriz inicial. Sin embargo, aunque el contenido de esta tabla es el adecuado, aún no son datos limpios; es necesario filtrar los datos. Restringimos aquellos tickets de compra con más de 20 elementos para asegurarnos de contar solo con ventas a particulares. También acotamos el tipo de producto a solo ropa excluyendo peluches, decoraciones navideñas, perfumes, cosmética, etc. Por último, eliminamos los artículos cuya 'familia' cae en la categoría de 'Genérico' ya que, para poder crear outfits válidos, necesitamos diferenciar entre camisetas, pantalones, zapatos, etc., y la categoría 'Genérico' no nos lo permite. Tras todo este procesado nos quedamos con un 63,29 % de los datos seleccionados. Esta bajada tan brusca se debe principalmente a la gran cantidad de elementos identificados como 'Genérico'. Pese a no ser despreciable la cantidad de valores perdidos consideramos que: a) el volumen de datos restante sigue siendo lo suficientemente grande y variado como para asegurar que el buen funcionamiento del modelo no se verá entorpecido; y b) que los datos faltantes son completamente al azar (*missing completely at random* or MCAR) porque, a primera vista, no parecen depender de ninguna variable. Especificamos 'a primera vista' porque es posible que pertenezcan a alguna familia (camisetas, pantalones, bufandas, gorros, accesorios, etc) pero la única forma que habría de determinarlo sería entrenar un clasificador, que con ayuda de las imágenes, categorizara las prendas. Sin embargo, con una rápida evaluación de nuestras prioridades y el tiempo disponible nos damos cuenta de que las ventajas que traería hacer esto no superan el coste asociado. En conclusión, asumimos que los datos son MCAR y, por lo tanto, eliminarlos es seguro, ya que no introducen ningún sesgo.

Del registro de la parte comercial y de la parte visual, queremos quedarnos con una tabla que no tenga valores nulos en las URLs de las fotos, ya que sin las imágenes no podemos observar los resultados. De esta selección, vamos a limpiar nuevamente eliminando aquellos que tienen el valor 'Genérico' en los atributos 'Descripción' y 'Familia', además

de excluir las líneas de productos de perfumería, cosmética y maquillaje. Esta depuración solo se lleva a cabo para reducir, en este caso en un $4,62\,\%$, las dimensiones de la tabla master para ahorrar recursos de memoria al guardarla y tiempo al cargarla.

4.1.2. Embedding de imágenes y similitud coseno

El diccionario resultante de esta etapa muestra que hemos unificado 11,326 elementos (valores) bajo 3,740 nombres (claves). Esta es una reducción muy significativa, considerando que hemos modificado el nombre del $24\,\%$ de nuestros artículos. Esto implica que nuestros resultados se verán afectados por esta etapa.

4.1.3. Cálculo de reglas de asociación

Al aplicar reglas de asociación obtenemos 166 mil relaciones entre pares de artículos. Podríamos ser más selectivos ajustando los parámetros del umbral, pero nos interesa mantener el mayor número posible de relaciones (siempre garantizando un mínimo de calidad). De este modo, en la próxima etapa, aumentamos las posibilidades tanto de encontrar cliques como de que estos sean de mayor tamaño.

4.1.4. Búsqueda de cliques

De una matriz de adyacencia de dimensión $461261529 \times 461261529$ encontramos $57k$ cliques máximos. Sin embargo, tras filtrar aquellos cliques cuyos elementos corresponden únicamente a una de las categorías "parte de arriba", "parte de abajo", "cuerpo entero" y "complementos" porque no pueden conformar un *outfit*, reducimos el número a $32k$. Adicionalmente, al solucionar el problema de las prendas de cuerpo entero disminuimos las opciones a $25k$. Por último, eliminando aquellos *outfit* que son sets establecidos y aquellos no válidos conservamos tan solo $9k$ de los iniciales, es decir, un $15.7\,\%$. Filtramos los conjuntos establecidos, como los bikinis o los trajes de chaqueta y pantalón, ya que están diseñados para tener una relación desde el inicio. En este proyecto, nos enfocamos en buscar asociaciones de artículos que surgen de manera natural por parte de los usuarios.

En conclusión, de las 166 mil reglas de asociación a pares se encuentran 57 mil cliques máximos de los cuales solo 9 mil se corresponden con un *outfit* válido. Se podría considerar un porcentaje pequeño si no se tuviera en cuenta que son *outfits* completos y frecuentes, es decir, con un éxito ya asegurado por sus previas compras. No estamos generando conjuntos basados en una predicción, sino en un conocimiento sobre el comportamiento de los consumidores. Es decir, se ha conseguido la cantidad de 9263 *outfits* que podrían ser recomendados a usuarios online para posiblemente aumentar el número de ventas o a diseñadores de interiores comerciales para mejorar la distribución de los artículos en tienda.

4.2. RESULTADOS

El volumen de resultados es demasiado grande para ser presentado en esta memoria y no se puede mostrar en formatos más compactos como gráficos. Por lo tanto, se procederá a explicar el análisis de los resultados utilizando ejemplos representativos de los mismos.

4.2.1. Análisis de los conjuntos preestablecidos

Vamos a hablar de qué valor aportan dentro de nuestro algoritmo los conjuntos establecidos.

Conjuntos preestablecidos indetectables

Los conjuntos preestablecidos siempre destacan en las métricas que cuantifican las relaciones entre artículos. Por ello, al ordenar los resultados en orden decreciente según el peso de los cliques, no es sorprendente que lo primero que resalte sea la sospecha de que no se han eliminado correctamente todos estos conjuntos, como se observa en los ejemplos 4.1.

FOTO				
DESCRIP.	b-camisa stpd posicionado cenefa hoja m/larga	b-set pantalon estampada	b-l1-blazer corta cuadro vichy	b-skort tablas cuadro vichy
M	2815	2864	2353	1971
C	80	40	64	84

Cuadro 4.1: Se muestran la descripción, el modelo y el color de dos conjuntos preestablecidos mostrados con fotos

En teoría, los componentes de los conjuntos preestablecidos pueden filtrarse porque presentan los mismos valores de modelo y color. Sin embargo, en la práctica, nos damos cuenta de que algunos de estos conjuntos no se filtran adecuadamente. Existe también la condición suficiente, aunque no necesaria, de que compartan la misma descripción.

Pese a que estos fallos son muy pocos y se presentan en un número reducido, son un buen indicativo del buen funcionamiento del modelo. Sin embargo, no es lo que nos interesa y puede llevarnos a cuestionar la legitimidad de otras combinaciones, como las presentadas en 4.2.

Asimismo, aunque estamos convencidos de que estos *outfits* no son conjuntos preestablecidos, es una buena señal que no podamos estar completamente seguros. Esto indica que nuestro algoritmo incluye combinaciones tan buenas como aquellas que fueron diseñadas con ese propósito. En conclusión, aunque estos fallos nos quitan algo de precisión, también aportan validez a nuestros resultados.

FOTO				
DESCRIP.	devo-046-body elastico estampado rayas c/polo	short falda tablas	b-camisa saten cruzada manga campana	devo-b falda satinada plisada larga
M	4661	5427	8143	3564
C	46	630	402	425

Cuadro 4.2: Se muestran la descripción, el modelo y el color de las fotos

Completitud de los conjuntos preestablecidos

Lo segundo que observamos es que la existencia de sets preestablecidos no impide la combinación con otras prendas para completar el outfit. Para mayor claridad, al mencionar que se filtraron los conjuntos programados, nos referimos a aquellos cliques que solo contenían elementos con los mismos valores de modelo y color. Sin embargo, sí nos interesa identificar artículos que aporten novedad a conjuntos diseñados para llevarse juntos, como en el caso 4.3:

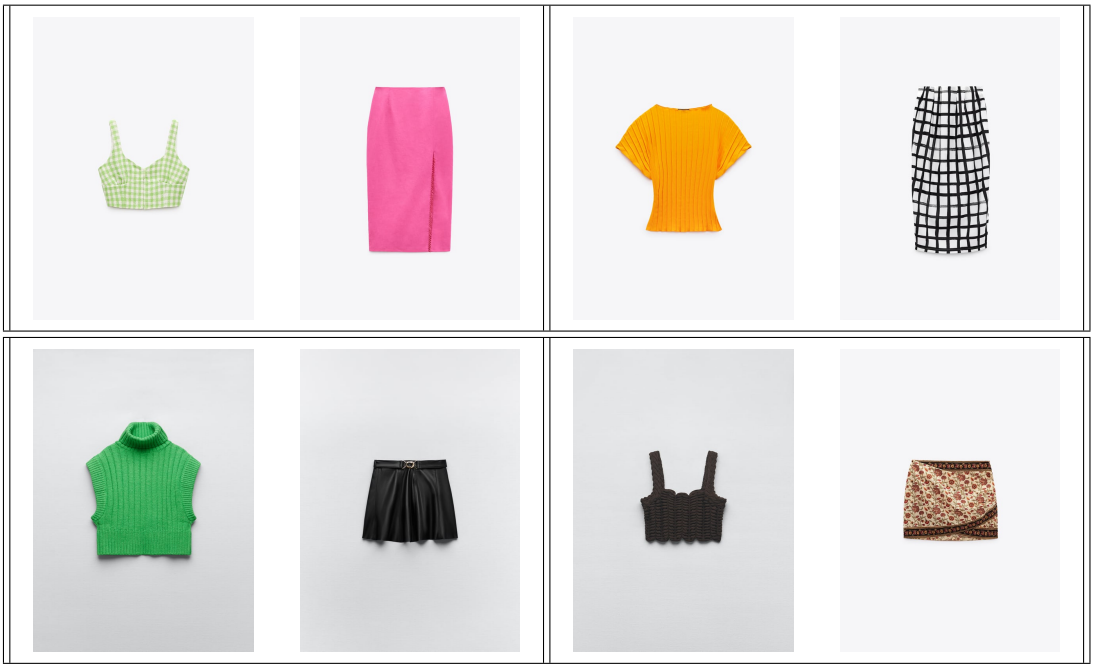
Como podemos observar, hay dos conjuntos preestablecidos a los que se añade un tercer elemento que no pertenece a ellos. Estos resultados demuestran la capacidad de nuestro algoritmo para completar conjuntos. En otras palabras, no solo genera combinaciones nuevas, sino que también las amplía, dotando al modelo de una mayor versatilidad. No está restringido por limitaciones o caminos predefinidos.

4.2.2. Análisis visual

La tercera observación que surge al analizar los resultados es que no existen restricciones por color, textura o forma, ya que el algoritmo no se basa en características visuales. Esto se evidencia en los ejemplos mostrados en 4.4.

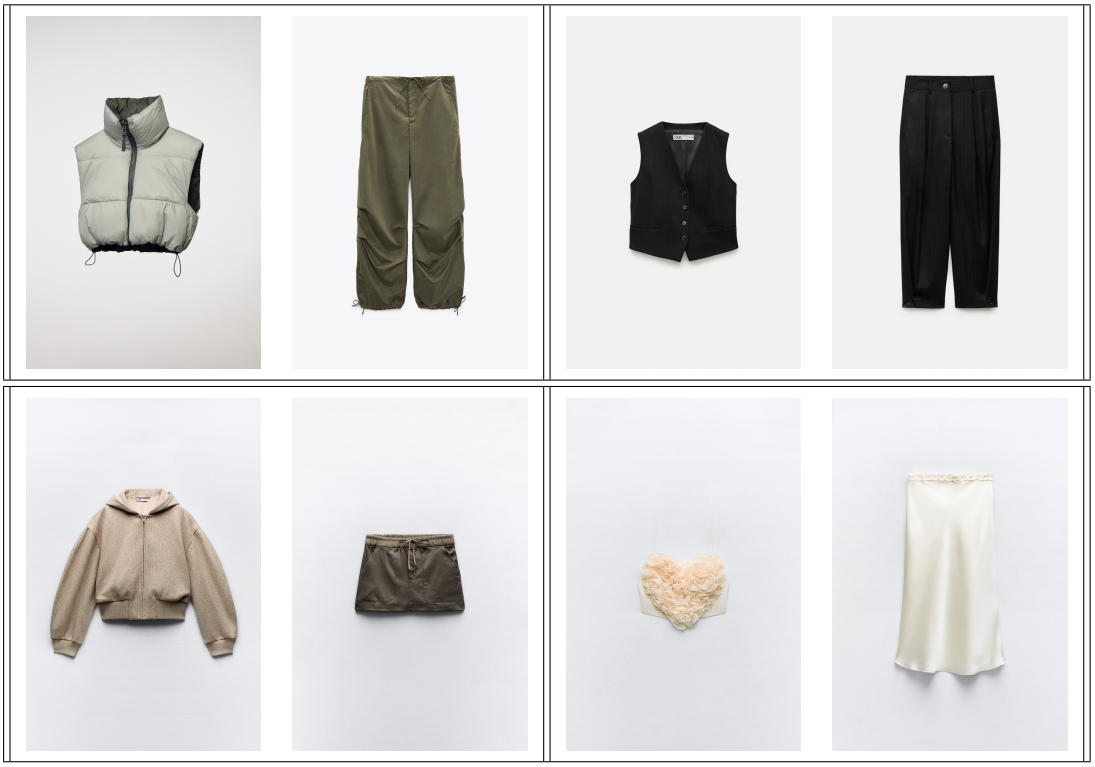
FOTO			
DES.	b- blusa estampada detalle gomas escote	b- falda maxi paneles estampada	cint piel 5.5cm tachas espejo
M	2459	2457	1736
C	330	330	700
FOTO			
DES.	ct-l1 sudadera capucha	ct-l1 pantalon ancho costuras	ct-l2 top bandeau ba
M	8417	8417	4174
C	807	807	485

Cuadro 4.3: Se muestran la descripción, el modelo y el color de dos conjuntos preestablecidos y el nuevo artículo que los complementa.



Cuadro 4.4: Se muestran cuatro ejemplos de *outfit* no afectados por la variación de color, textura y forma de sus componentes

Esta flexibilidad es una gran ventaja porque permite al algoritmo descubrir combinaciones innovadoras y no convencionales que podrían pasar desapercibidas si solo se consideraran atributos visuales. Además, esta característica amplía el alcance del algoritmo, permitiéndole adaptarse a diferentes estilos y preferencias sin necesidad de ajustes específicos. Al basarse en la combinación de artículos que se compran juntos, el algoritmo también incorpora un cierto sentido de la ocasión. Es decir, se puede observar ?? cómo no mezcla ropa informal con deportiva, o ropa de fiesta con elegante, lo que añade coherencia y relevancia a las recomendaciones.

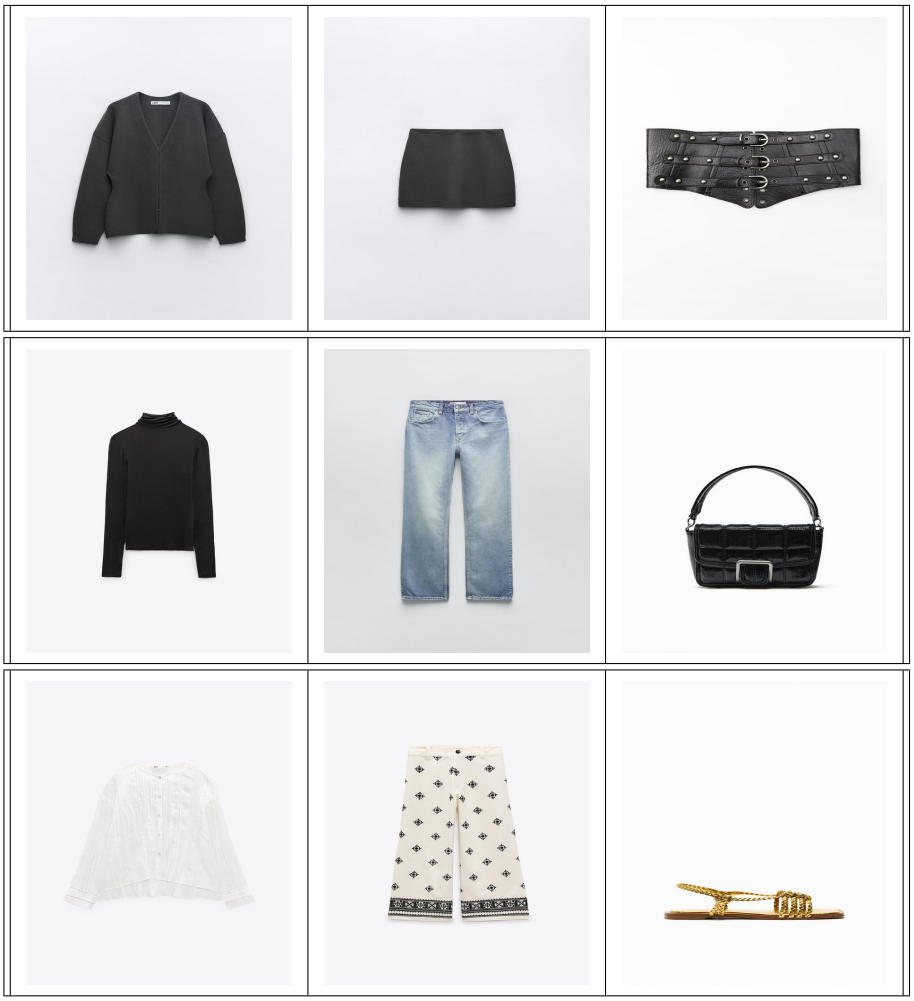


Cuadro 4.5: Se muestran cuatro ejemplos de *outfit* que mantienen coherencia formal dentro de las partes

4.2.3. Análisis de complementos

A pesar de haber clasificado todo el calzado, los accesorios e incluso la ropa interior bajo la categoría de " extras", apenas se observan cliques que incluyan estas componentes. Esto simplemente refleja la distribución real en las tiendas, donde los otros tipos de artículos son mucho más abundantes. Aun así, nos encontramos ejemplos 4.6.

No se deben subestimar estos resultados, ya que, a pesar de que estos artículos no son el foco principal en las tiendas, han logrado destacar en el ranking de elementos frecuentes. Además, productos como un bolso negro, considerado un básico de armario, tienen un rango más amplio de emparejamiento con multitud de outfits. Para aparecer en una regla de asociación, no solo deben ser artículos populares, sino que también deben comprarse frecuentemente en conjunto con otros productos. Es decir, entre todas las combinaciones posibles, este bolso se ha registrado en caja muchas más veces junto a esa camiseta y ese pantalón que con cualquier otra combinación. Siempre, pero especialmente en combinaciones tan específicas, se debería analizar si ha habido algún detonante de esta relación, como los maniquíes en tiendas, el vestuario de famosos, etc.



Cuadro 4.6: Se muestran tres *outfits* con artículos 'extra' como los cinturones, los bolsos o los zapatos.

4.2.4. Análisis de las prendas de cuerpo entero

En este análisis hay que aclarar que aunque hemos definido como *outfits* válidos aquellos que incluyen prendas de cuerpo entero junto con extras, excluyendo combinaciones de partes superiores e inferiores, hay excepciones. Aunque un vestido no pueda combinarse con un top o unos shorts, sí puede combinarse con chaquetas, chalecos o abrigos. Permitiendo estas excepciones nos encontramos con *outfits* como estos 4.7.:

Sin embargo, como se mencionó en la sección anterior, los complementos son escasos, obligando a prendas de cuerpo entero como los vestidos a asociarse casi únicamente con estas excepciones. Esto supone un problema estacional para nuestro algoritmo porque, aunque los vestidos son muy demandados, su uso está más asociado al verano. Esto significa que el emparejamiento con chaquetas, chalecos y abrigos se da principalmente en invierno y entretiempo. En invierno, aunque hay menos vestidos disponibles, las posibilidades de combinarlos con otros artículos aumentan. Por el contrario, en verano, hay más vestidos, pero las combinaciones son menos variadas debido al calor y la falta de complementos como sandalias, bolsos o cinturones. No por ninguna razón se reducía el número de cliques de 32 mil a 25 mil cuando se trataba de resolver el problema de las prendas de cuerpo entero.

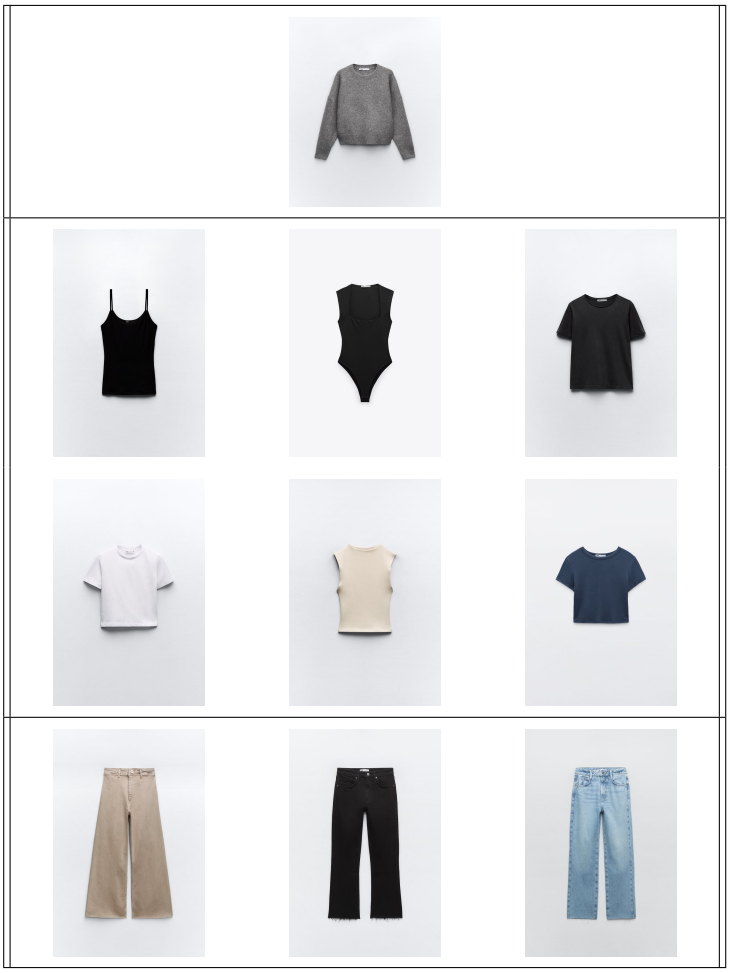
Cuadro 4.7: Se muestran cuatro ejemplos de *outfit* que muestran combinaciones con prendas de cuerpo entero

4.2.5. Análisis del número de prendas

Para mayor comodidad hasta ahora solo se habían enseñado *outfits* de dos o tres prendas pero la realidad es que el tamaño de nuestros cliques es ilimitado. Esto no quiere decir que el número de prendas de un solo *outfit* es ilimitado, sino que tenemos múltiples variaciones del mismo *outfit*. Es decir, dentro de un mismo estilo de *outfit* disponemos de varias opciones sobre las que elegir como por ejemplo 4.8.

No importa la combinación que elijas dentro de este mismo clique, ya que todas las posibles combinaciones funcionan bien. Este tipo de resultados podría visualizarse en un futuro como 4.1

Este formato da la oportunidad de elegir qué variación del *outfit* te gusta más, añadiendo interacción con el usuario y aumentando el margen de error del recomendador. Esto hace que el sistema sea más flexible y tolerante a pequeñas variaciones o incertidumbres en los datos o en el entorno. Es beneficioso cuando no se busca la precisión absoluta y se prioriza la capacidad de adaptación.



Cuadro 4.8: Variaciones de un mismo *outfit*

Figura 4.1: Posible futuro diagrama de la recomendación de varias variaciones de un mismo outfit

4.2.6. Análisis del sistema de cliques

Por último, a medida que descendemos por la lista ordenada de outfits, observamos la repetición de cliques casi idénticos. En cliques masivos, es común encontrar un grupo central de artículos compartidos por varios cliques. Esto ocurre porque, en comunidades tan interconectadas, es fácil encontrar elementos relacionados con estos clusters pero aislados del resto de los productos. Para entenderlo mejor observemos 4.9.

Esta macrocomunidad surge debido a dos razones: primero, estos artículos son considerados básicos, lo que significa que se compran con frecuencia y combinan prácticamente con todo. Segundo, hemos incluido en el grafo los elementos con lift menor que uno, es decir, aquellos que se consideran sustitutos uno del otro.

En el ejemplo propuesto, podemos observar cómo ambos argumentos son ciertos. La mayor parte de los cliques está compuesta por ropa básica, como camisetas blancas o negras y vaqueros. Además, dentro de estos cliques, encontramos elementos muy similares entre sí. Por ejemplo, hay dos camisetas blancas de tirantes que solo varían en el grosor de los mismos y dos camisetas de manga corta que difieren únicamente en la longitud de la manga.

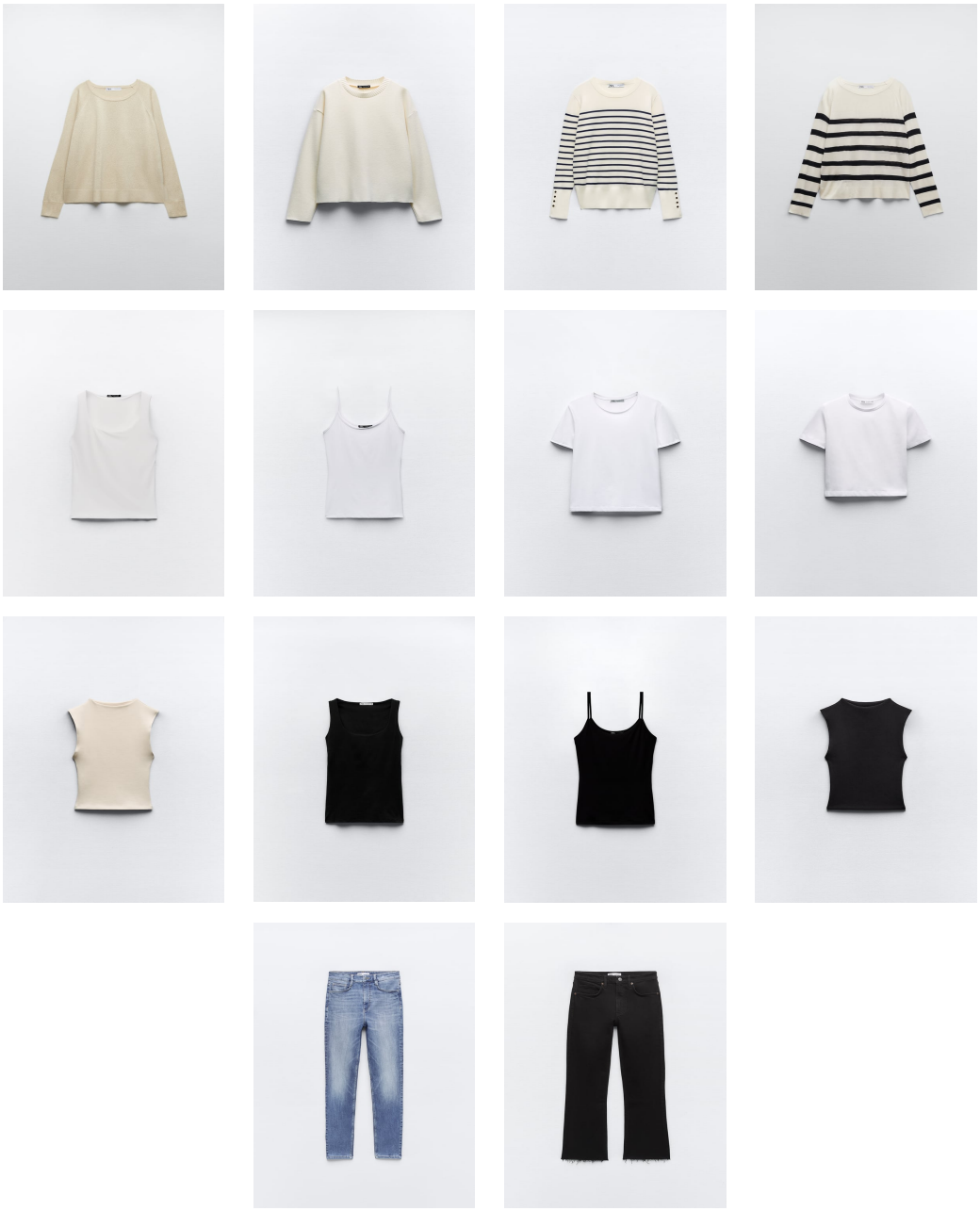
No existe un clique máximo que englobe ambos conjuntos porque nuestro algoritmo no ha detectado una asociación lo suficientemente fuerte entre el top del clique A y el jersey y vaquero oscuro del clique B como para considerar un conjunto compuesto por los tres frecuente. Estas pequeñas variaciones multiplican el número de cliques sin aportar información especialmente relevante. Sin embargo, no son malos resultados; simplemente requieren un tratamiento de datos posterior que permita combinar varios de estos cliques en uno solo cuando se pueda conservar la coherencia. Si se consiguiese, habríamos conseguido localizar una comunidad muy grande, y por tanto, muy relevante llena de posibles recomendaciones de *outfits* válidas. Sería una herramienta potente para cubrir rápidamente grandes conjuntos de datos.

4.2.7. Análisis adicionales

- Como era de esperar, los cliques con mayor peso se encuentran en pares o tríos de artículos. Esto se debe a que la relación entre artículos se basa en la frecuencia con la que se compran juntos. Cuanto menos se divida la frecuencia de un artículo al relacionarse con otros, mayores serán las métricas de asociación encontradas.

- En los cliques masivos, hay una mayor prevalencia de partes de arriba, especialmente camisetas, en comparación con partes de abajo. Esto se debe a la naturaleza duradera y atemporal de los pantalones. Con esta información, sabemos que no solo debemos enfocarnos en la recomendación exacta de conjuntos, sino también en ofrecer opciones que se renuevan con mayor frecuencia en nuestro armario.

- Los productos de temática solo se relacionan entre ellos. Por ejemplo, debido al éxito de la película de Barbie se han venido muchos productos asociados a esta, y pese a que sería fácil combinarlos con cualquier otro artículo rosa, no hay signos de ello.

Artículos comunes a los cliques A y B	
	
Artículo aislado del clique A	Artículos aislados del clique B
	

Cuadro 4.9: Cliques en torno a un mismo clúster

Capítulo 5

Conclusiones y trabajo futuro

5.1. CONCLUSIONES

En este proyecto se ha avanzado en la exploración del área de los sistemas de recomendación basados en el análisis de tickets. La búsqueda de *outfits* ha requerido de análisis de imágenes, técnicas de minería de datos, teoría de grafos y análisis de redes. Gracias a la combinación de estas diferentes técnicas hemos obtenido resultados donde se destacan las siguientes ideas:

- La existencia de conjuntos preestablecidos aporta dos beneficios clave: primero, valida nuestro algoritmo al demostrar que puede generar combinaciones tan efectivas como las diseñadas específicamente para ese propósito; segundo,evidencia su capacidad para generar y extender conjuntos, lo que le otorga mayor flexibilidad y evita seguir rutas predeterminadas.
- El algoritmo no limita las recomendaciones según características visuales como color, textura o forma. Además, al analizar combinaciones de artículos comprados juntos, el algoritmo también considera el contexto de uso, asegurando que las sugerencias sean coherentes y pertinentes según la ocasión.
- Los accesorios, el calzado e incluso ropa interior, clasificados bajo la categoría de 'extras', poseen casi nula relevancia en nuestro algoritmo. Sin embargo, esto no se considera un problema ya que solo replica la distribución de artículos existente en las tiendas.
- El esfuerzo puesto por conseguir *outfits* válidos con prendas de cuerpo entero solo se ve plasmado en combinaciones de vestido con chaquetas, chalecos y abrigos.
- El futuro de la recomendación incluirá variaciones de un mismo *outfit* con las que el usuario podrá interactuar para que se ajuste mejor a sus gustos personales. Los resultados con un gran número de prendas hacen que nuestro algoritmo sea capaz de permitir una mayor personalización.
- Uno de los problemas actuales en nuestros resultados es la repetición de conjuntos grandes con mínimas variaciones. Disemina la información realmente valiosa sobre la localización de un vecindario altamente conectado.

Dicho esto, podemos afirmar que los resultados son positivos al proporcionar combinaciones coherentes y válidas, elevando el potencial de convertirse en un modelo viable para implementaciones prácticas en la empresa.

Todo esto sería inviable si no se cumpliera la premisa fundamental de que los usuarios adquieren conjuntos de ropa combinados, en lugar de simplemente prendas individuales que puedan combinar con su armario en casa.

En conclusión, la recomendación de conjuntos basada en el análisis de tickets de compra es un campo prometedor que ha generado resultados significativos e interesantes en nuestro proyecto.

5.2. TRABAJO FUTURO

Este proyecto tiene un gran potencial y, a pesar de las horas dedicadas, existen numerosas áreas de mejora que podrían abordarse en el futuro:

- Si disponemos de las URL de los artículos categorizados como 'Genéricos', podríamos entrenar un clasificador para asignar estos valores perdidos a sus respectivas familias, lo cual nos permitiría aumentar significativamente el volumen de datos inicial ya que pasarían a ser válidos para nuestro algoritmo. No solo eso, sino que podríamos aprovechar este clasificador entrenado para directamente no tener que depender de la familia que aparece en los datos.

- Se podría realizar una labor de investigación para explorar las implicaciones que conlleva repetir este estudio pero aplicando el análisis de imagen para detectar elementos únicos mal identificados como distintos artículos después de encontrar las reglas de asociación. Si las tiendas físicas tienden a desviarse de la idea de tienda ideal debido a la falta consistente de stock en algunos productos o a distribuciones no óptimas, nuestro algoritmo podría beneficiarse de calcular los embeddings de las imágenes después de realizar la agrupación de artículos y ejecutar el aprendizaje de reglas de asociación, y no antes como hemos hecho nosotros. Por ejemplo, si dos tiendas tienen el mismo pantalón registrado bajo nombres diferentes, mientras que en tiendas ideales la proporción con la que se relaciona con unas botas y una camisa con las cuales combina es la misma, en tiendas imperfectas podría suceder que en una tienda solo se asocie con las botas y en otra solo con la camisa. Este cambio en la frecuencia de asociación nos proporciona una señal más fuerte o significativa si mantenemos este producto separado al aplicar minería de datos.

- Deberíamos experimentar con los parámetros de umbral seleccionados para determinar la frecuencia de los conjuntos al calcular las reglas de asociación. Es crucial encontrar el rango óptimo donde, considerando nuestros recursos computacionales, podamos descubrir la mayor cantidad de reglas de asociación posible sin comprometer la significancia de lo que consideramos como frecuente.

- En nuestro proyecto hemos usado la métrica lift para construir los pesos de los grafos pero habría que averiguar cómo cambiarían los resultados si se usasen soporte o confianza.

- Uno de los cambios más significativos sería la expansión del algoritmo mediante la incorporación del análisis de imagen en la fase final. Calcular los embeddings es un proceso costoso, por lo que podría resultar más efectivo aprovechar los embeddings y la matriz de similitud previamente calculados para completar los *outfits* finales. Por ejemplo, si tenemos un conjunto inicial compuesto por tres camisetas y un pantalón, podríamos completarlo añadiendo los dos pantalones más similares hasta alcanzar un total de tres conjuntos completos.

- Por último, y aprovechando de nuevo los embeddings y la matriz de similitud ya calculada, sería beneficioso condensar la repetición de cliques en uno solo, asegurando la coherencia visual de la comunidad. Podríamos incluir elementos aislados siempre y cuando cumplan con las características suficientes para formar parte de la unidad cohesiva.

Con esto queremos recordar que siempre hay margen de mejora y que nuestro objetivo siempre ha sido buscar la mayor calidad de nuestros resultados.

Bibliografía

[1] Algoritmos De Recomendación Basados en Contenido - FasterCapital — fastercapital.com. https://fastercapital.com/es/tema/algoritmos-de-recomendaci%C3%B3n-basados-%E2%80%8B%E2%80%8Ben-contenido.html. [Accessed 10-06-2024].

[2] Bron–Kerbosch algorithm - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm, . [Accessed 17-06-2024].

[3] Grafo - Wikipedia, la enciclopedia libre — es.wikipedia.org. https://es.wikipedia.org/wiki/Grafo, . [Accessed 01-06-2024].

[4] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, junio 2005. doi: 10.1109/TKDE.2005.99.

[5] C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, septiembre 1973. doi: 10.1145/362342.362367.

[6] P. B.Thorat, R. Goudar, and S. Barve. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications*, 110:31–36, 01 2015. doi: 10.5120/19308-0760.

[7] ConectaSoftware. Neo4j. https://www.conectasoftware.com/magazine/neo4j/. [Accessed 01-06-2024].

[8] A. Conte and E. Tomita. On the overall and delay complexity of the cliques and bron-kerbosch algorithms. *Theoretical Computer Science*, 899:1–24, enero 2022. doi: 10.1016/j.tcs.2021.11.005.

[9] M. de Gemmis, P. Lops, C. Musto, F. Narducci, and G. Semeraro. *Semantics-Aware Content-Based Recommender Systems*, pages 119–159. 01 2015. ISBN 978-1-4899-7636-9. doi: 10.1007/978-1-4899-7637-6_4.

[10] Y. Deldjoo, F. Nazary, A. Ramisa, J. Mcauley, G. Pellegrini, A. Bellogin, and T. D. Noia. A review of modern fashion recommender systems, 2023.

[11] H. Eroy. An Alternative Method of Community Detection — medium.com. https://medium.com/smucs/

an-alternative-method-of-community-detection-850f9578fb4a, 2022. [Accessed: 30-05-2024].

[12] J. A. Fernando Carazo. Métricas descriptivas de grafos y redes. https://cienciadedatos.net/documentos/pygml04-metricas-grafos-redes-python, 2024. [Accessed 01-06-2024].

[13] J. Gallagher. What is an Image Embedding? — blog.roboflow.com. https://blog.roboflow.com/what-is-an-image-embedding/, 2023. [Accessed 32-05-2024].

[14] A. Jain. Frequent itemset generation with the fp-growth algorithm. https://blog.knoldus.com/machinex-frequent-itemset-generation-with-the-fp-growth-algorithm/, 2018. [Accessed 10-06-2024].

[15] A. Jain. Understanding fp-tree construction. https://blog.knoldus.com/machinex-understanding-fp-tree-construction/, 2018. [Accessed 10-06-2024].

[16] A. Jain. Why no one uses apriori algorithm for association rule learning? https://blog.knoldus.com/machinex-why-no-one-uses-apriori-algorithm-for-association-rule-learning/, 04 2018. [Accessed 10-06-2024].

[17] R. Jodha. FP Growth Algorithm in Data Mining - Scaler Topics — scaler.com. https://www.scaler.com/topics/data-mining-tutorial/fp-growth-in-data-mining/. [Accessed 10-06-2024].

[18] T. Kumawat. Deep Walk and Node2Vec: Graph Embeddings — tejpal.abhyuday. https://medium.com/@tejpal.abhyuday/deep-walk-and-node2vec-graph-embeddings-faf02d369442, 2023. [Accessed 30-05-2024].

[19] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang. Pfp: Parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, pages 107–114, Lausanne, Switzerland, 2008. ACM. doi: 10.1145/1454008.1454027. URL https://doi.org/10.1145/1454008.1454027.

[20] K. O'Shea and R. Nash. An introduction to convolutional neural networks, 2015.

[21] L. Picek, M. Sulc, J. Matas, J. Heilmann-Clausen, T. Jeppesen, and E. Lind. Automatic fungi recognition: Deep learning meets mycology. *Sensors*, 22:633, 01 2022. doi: 10.3390/s22020633.

[22] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, and e. a. Sastry, Girish. Learning transferable visual models from natural language supervision. arXiv, febrero 2021. http://arxiv.org/abs/2103.00020.

[23] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021.

[24] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009:1–19, October 27 2009. doi: 10.1155/2009/421425. URL https://doi.org/10.1155/2009/421425.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. arXiv, agosto 2023. http://arxiv.org/abs/1706.03762.

[26] W. Zhang, R. Shang, and L. Jiao. Complex network graph embedding method based on shortest path and moea/d for community detection. *Applied Soft Computing*, 97: 106764, December 2020. doi: 10.1016/j.asoc.2020.106764. URL https://doi.org/10.1016/j.asoc.2020.106764.

[27] Z. Zhang, P. Pu, D. Han, and M. Tang. Self-adaptive louvain algorithm: Fast and stable community detection algorithm based on the principle of small probability event. *Physica A: Statistical Mechanics and Its Applications*, 506:975–986, septiembre 2018. doi: 10.1016/j.physa.2018.04.036.