Predicción de Compras Futuras a través de Patrones de Consumo

Brandon Uriel Garcia Sanchez

Agosto 9, 2025

Abstract

Este documento detalla el proceso de un algoritmo de Análisis de Cesta de Mercado (Market Basket Analysis) implementado en un entorno de Python utilizando librerías como Pandas y SQLite. El objetivo principal es identificar asociaciones entre productos comprados frecuentemente juntos en una base de datos de transacciones de comercio electrónico, para así poder generar recomendaciones de productos basadas en los hábitos de consumo de los clientes.

1 Introducción

El algoritmo está diseñado para descubrir patrones de compra y relaciones entre productos dentro de un gran volumen de datos de transacciones. A través del cálculo de métricas como el soporte, la confianza y el lift, es posible identificar qué productos tienden a ser comprados de manera conjunta, sentando las bases para un sistema de recomendación.

2 Configuración e importación de librerías

Para dar inicio al análisis, se importan las librerías necesarias para la manipulación y el análisis de datos.

```
# Algoritmo de consumo Market Basket Analyst.
# Consumo E-Commerce
# TPE
# CatoXP - Brandon Uriel Garcia Sanchez

import sqlite3 # importamos el sqlite3
import pandas as pd # libreria para manipulación de datos
```

3 Carga y preprocesamiento de datos

El primer paso consiste en cargar los datos desde una base de datos SQLite y prepararlos para el análisis.

3.1 Conexión a la base de datos

Se establece una conexión con la base de datos que contiene la información de las transacciones.

```
# Primero se conecta a la base de datos SQLite
conexion = sqlite3.connect("DBsanoyfresco.db")
```

3.2 Carga de datos a un DataFrame

La tabla de `tickets` es cargada en un DataFrame de Pandas para su manipulación.

```
# Cargar una tabla completa en un DataFrame
df = pd.read_sql_query("SELECT * FROM tickets", conexion)

# Mostrar los primeros registros del DataFrame
df
```

El DataFrame resultante contiene 4,975,718 registros y 11 columnas, entre ellas: `id_pedido`, `id_cliente`, `fecha`, `nombre_producto`, etc.

3.3 Verificación y limpieza de datos

Se inspecciona la calidad de los datos y se realizan las conversiones de tipo necesarias.

```
# verificamos la calidad de datos
df.info()
```

Se identifica que la columna `fecha` es de tipo `object` y se convierte a `datetime`.

```
# la columna fecha se encuentra en Dtype objeto, se tuvo que cambiar a datetime
df["fecha"] = pd.to_datetime(df["fecha"])
```

3.4 Preparación del DataFrame para el algoritmo

Para el análisis de la cesta de mercado, únicamente se necesitan las columnas `id_pedido` y `nombre_producto`.

```
# para el algoritmo no necesito todo el dataframe solamente el pedido y el nombre del producto
# asi lo llamaré como df_cesta = dataframe cesta
df_cesta = df[["id_pedido","nombre_producto"]]
df_cesta
```

4 Análisis de Asociación

Con los datos ya preparados, se procede a aplicar las técnicas de análisis de cesta de mercado.

4.1 Agrupación de productos por pedido

Se agrupan los productos por cada pedido, concatenando los nombres de los productos en una sola cadena de texto.

```
# agrupamos los productos por id_pedido con groupby
df_agrupado = df_cesta.groupby("id_pedido")["nombre_producto"].apply(lambda producto: ",".join(produ
df_agrupado
```

4.2 Creación de la matriz de transacciones

Se transforma el DataFrame agrupado en una matriz binaria (matriz de transacciones) donde cada columna representa un producto y cada fila una transacción. Un valor de 1 indica que el producto fue comprado en esa transacción, y 0 en caso contrario.

```
# aplicamos una funcion pandas .get_dummies() para transformar
# productos en columnas con 0 / 1
# si fue comprado en ese pedido tiene un 1 de lo contrario tiene un 0
df_transacciones = df_agrupado.str.get_dummies(sep=",")
df_transacciones
```

4.3 Cálculo del Soporte

El soporte de un producto es la proporción de transacciones en las que aparece. Se calcula como la media de cada columna en la matriz de transacciones.

$$\text{Soporte}(A) = \frac{\text{Número de transacciones con A}}{\text{Número total de transacciones}}$$

```
# soporte para cada producto, al ser solamente 1 y 0 podemos realizar una media * 100
# para poder realizar el algoritmo mátematico p(aUb)
soporte = df_transacciones.mean() * 100
soporte.sort_values(ascending=False)
```

4.4 Definición de las funciones de Confianza y Lift

Se definen dos funciones clave para medir la fuerza de las asociaciones entre productos.

- **Confianza:** Mide la probabilidad de comprar el producto B si ya se ha comprado el producto A.

$$\text{Confianza}(A \to B) = \frac{\text{Soporte}(A \cup B)}{\text{Soporte}(A)}$$

- **Lift:** Mide cuánto más probable es comprar A y B juntos de lo que sería si fueran independientes. Un lift ¿ 1 indica una asociación positiva.

$$\text{Lift}(A, B) = \frac{\text{Soporte}(A \cup B)}{\text{Soporte}(A) \times \text{Soporte}(B)}$$

```
# Aplicamos lógica con funciones para calcular la confianza entre dos productos en la muestra
def confianza(antecedente, consecuente):
  # casos donde se compran ambos productos
  conjunto_ac = df_transacciones[(df_transacciones[antecedente] == 1) &
                                 (df_transacciones[consecuente] == 1)]
  # Confianza - compras conjuntas / compras de producto A
  return len(conjunto_ac) / df_transacciones[antecedente].sum()

# Funcion para calcular el lift entre dos productos en la muestra
def lift(antecedente, consecuente):
  soporte_a = df_transacciones[antecedente].mean() # P(A)
  soporte_c = df_transacciones[consecuente].mean() # P(B)
  conteo_ac = len(df_transacciones[(df_transacciones[antecedente] == 1) &
                                   (df_transacciones[consecuente] == 1)]) # número de transacciones
  soporte_ac = conteo_ac / len(df_transacciones) # P(A  B)
  return soporte_ac / (soporte_a * soporte_c) # P(A  B) / (P(A) * P(B))
```

4.5 Generación de Reglas de Asociación

Se generan combinaciones de todos los productos y se calculan la confianza y el lift para cada par. Se establece un umbral de confianza mínimo (0.05) para filtrar las reglas más relevantes.

```
from itertools import combinations
# definir un umbral para la confianza minima
umbral_confianza = 0.05
asociaciones = []

# generar combinaciones de productos y calcular confianza y lift
for antecedente, consecuente in combinations(df_transacciones.columns,2):
  # soporte del antecedente
  soporte_a = df_transacciones[antecedente].mean()

  #calcular confiaza
  conf = confianza(antecedente, consecuente)
  if conf > umbral_confianza:
    asociaciones.append({
        'antecedente': antecedente,
        'consecuente': consecuente,
        'soporte_a' : round(soporte_a *100,1),
        'confianza' : round(conf * 100,1),
        'lift': round(lift(antecedente, consecuente),1)
    })

# convertir las asociaciones en un DataFrame
df_asociaciones = pd.DataFrame(asociaciones)
```

```
#ordenar las asociaciones por confianza de mayor a menor
df_asociaciones.sort_values(by='lift', ascending=False, inplace=True)
df_asociaciones # show
```

5 Enriquecimiento y Exportación de Resultados

Finalmente, las reglas de asociación encontradas se enriquecen con información adicional de los productos y se exportan a un archivo CSV.

5.1 Creación de tabla de productos únicos

Se genera un DataFrame con la información única de cada producto para facilitar el enriquecimiento de los datos.

```
# crear una tabla con los productos únicos y las columnas correspondientes
productos_unicos = df[['id_producto','id_seccion','id_departamento','nombre_producto']].drop_duplica
productos_unicos
```

5.2 Enriquecimiento del DataFrame de asociaciones

Se fusiona el DataFrame de reglas de asociación con la tabla de productos únicos para añadir detalles como el ID del producto, sección y departamento.

```
df_asociaciones_enriquecido = df_asociaciones.merge(productos_unicos, left_on='antecedente', right_o
df_asociaciones_enriquecido.columns = ['antecedente', 'consecuente', 'soporte_a', 'confianza', 'lift
df_asociaciones_enriquecido
\end{rinted}

\subsection{Exportación de los resultados}
Las reglas de asociación finales y enriquecidas se guardan en un archivo CSV para su uso posterior.

\begin{minted}{python}
df_asociaciones_enriquecido.to_csv("reglas.csv", index=False,sep=";",decimal=",")
```

6 Conclusión

El algoritmo implementado permite procesar un gran conjunto de datos de transacciones para extraer - ***reglas*** de asociación significativas entre productos. Los resultados, encapsulados en el archivo `reglas.csv`, proporcionan una base sólida para la implementación de un sistema de recomendación de productos, con el potencial de mejorar la experiencia del cliente e incrementar las ventas a través de sugerencias personalizadas y relevantes.