DESARROLLO DE UN SISTEMA DE RECOMENDACIÓN PARA UN CASO REAL

by

Enrique Villamarín Fonseca

A thesis submitted in conformity with the requirements
for the MSc in Economics, Finance and Computer Science

University of Huelva & International University of Andalusia

September/November 2018

Desarrollo de un sistema de recomendación para un caso real

Enrique Villamarín Fonseca

Máster en Economía, Finanzas y Computación

Gonzalo A. Aranda Corral
Universidad de Huelva y Universidad Internacional de Andalucía

2018

Abstract

The recommendation systems are at its peak, this is due to the number of platforms that use them in their day to day operations. Furthermore, they have also become important as tools that facilitate the decision-making process. This Master's Dissertation carries out a collaborative filtering recommendation system based on Formal Concept Analysis to improve the sales process and to maximize the sales of the company that uses this system.Key words: FCA, recommendation system, collaborative filtered, association rules

Key words: FCA, recommendation system, collaborative filtered, association rules

Resumen

Los sistemas de recomendaciones están en auge, esto se debe a la cantidad de plataformas que lo utilizan en su funcionamiento día a día. A su vez, también han tomado importancia como herramientas que facilitan el proceso de toma de decisiones. Este Trabajo Fin de Máster lleva a cabo un sistema de recomendación de filtrado colaborativo basado en el análisis de conceptos formales para facilitar el proceso de venta y así intentar maximizar las ventas de la empresa para la cual se ha realizado.

Palabras Claves: ACF, sistema de recomendación, filtrado colaborativo, reglas de asociación

Agradecimientos

En primer lugar, me gustaría darle las gracias al profesor Dr. D. Gonzalo A. Aranda Corral por dejarme trabajar con él, enseñarme y no desesperarse conmigo. También me gustaría darles las gracias a mi familia y amigos por apoyarme y animarme a continuar.

ÍNDICE

ÍNDICE DE FIGURAS

1 Introducción.

"El conocimiento es poder" (Francis Bacon, s.XVI) pero en pleno siglo XXI, con la llegada del "Big Data", el internet de las cosas y la multitud de nuevos sensores, la cantidad de millones de datos e información que se genera en la red dificulta el acceso al conocimiento sin un minucioso análisis de los mismos. Así, por ejemplo, empresas como IBM afirman que generan 2,5 pentabillones de bytes al día. Aguilar. F. J. (2013) en su libro "Big Data, Análisis de grandes volúmenes de datos en organizaciones" confirma que en 2012 se crearon 2,8 zettabytes (1 zettabytes = 1 billón de gigabytes) y anuncia que para 2020 la estimación es de cerca de 40 zettabytes.

Esta gran acumulación de datos ha producido que distintos profesionales de diversos ámbitos del saber se especialicen en la captación y generación de conocimiento, es decir, que surgiese la figura del *Data Scientist* o analista de datos. Uno de los ámbitos del conocimiento de mayor impacto de los analistas de datos es el área de comercialización e investigación de mercados (marketing). En este ámbito, el correcto tratamiento de los datos genera conocimiento que permite realizar una oferta personalizada en función de los gustos, los criterios de compras o incluso poder adquisitivo de los clientes, tanto actuales como potenciales. Este tipo de análisis se puede basar en sistemas de recomendación como herramientas de filtrado de datos que facilitan la obtención de los datos específicos solicitados por las empresas. Por todo ello, centraremos este trabajo en el desarrollo de un sistema de recomendación basado en un filtrado colaborativo que permite una mejor selección cuando existe un determinado número de afinidades.

Este trabajo se estructura fundamentalmente en dos partes: el marco teórico y la experimentación. En el marco teórico se revisan los conceptos básicos relativos al análisis formal de conceptos y a los sistemas de recomendaciones. El apartado de experimentación describe los distintos pasos realizados para la creación y desarrollo del sistema de recomendación de filtrado colaborativo.

2 Marco Teórico: Conceptos básicos para la creación del sistema de recomendación.

2.1 Análisis formal de conceptos (FCA).

El análisis formal de concepto (FCA) es una teoría matemática introducido por Rudolf Willie en 1984 y basada en la desarrollada por Garret Birkhoff y otros en los años treinta (teoría de retículos). Como pone de manifiesto Rodríguez Jiménez, J.L (2017) en su tesis doctoral en referencia al FCA, "se trata de un entorno conceptual para estructurar, analizar, minimizar, visualizar y revelar conocimiento a partir de la información proporcionada en términos de una relación binaria entre los elementos de dos conjuntos: objetos y atributos".

Esta sección es una breve introducción al FCA, sus nociones básicas de contexto formal y concepto, y a las implicaciones entre atributos de un contexto formal.

2.1.1 Contextos formales y concepto formal.

Dentro de FCA, el contexto formal es la unidad básica de representación del conocimiento. En su libro "*Formal Concept Analysis*", Ganter, B. y Wille, R. (1999) definen un contexto formal como una terna K = (G,M,I) donde "G" es un conjunto de objetos, "M" es un conjunto de atributos e "I" es la relación existente entre "G" y "M".

Paises	NW	UNP	CT	G8	EU	UN
USA	×	×		×		×
Alemania				×	×	×
Francia	×	×		×	×	×
Reino Un.	×	×		×	×	×
Turquía						×
Qatar			×			×
Italia			×	×	×	×

Figura 1- Ejemplo de contexto formal. Extraído de Aranda (2011) pág. 26.

A su vez, postulan que para que un par (A,B) dentro de un contexto formal K = (G,M,I) sea un concepto formal verificar:

$$A \subseteq G,\ B \subseteq M$$

$$A' = B,\ B' = A.$$

Si K = (A,B) es un concepto formal, dentro de él se pueden identificar dos partes. Por un lado se encuentra la extensión, formada por el conjunto de objetos "A" que lo componen. Y por el otro lado se puede observar la intensión, formada por el conjunto de atributos "B".

2.1.2 Implicación de atributos.

Los contextos formales también se pueden definir como un conjunto de implicaciones entre los atributos del mismo. Tales que las implicaciones puedan ser extraídas del contexto y el contexto pueda ser reconstruido a partir de las implicaciones.

Para definir bien la implicación de atributos y todo lo que ello conlleva debido a su complejidad se va a extraer los conceptos básicos de la tesis doctoral del profesor Aranda (2011). Si el lector quiere profundizar en el tema se le recomienda acudir al libro *"Formal Concept Analysis"* (Ganter, B. y Wille, R. (1999). El profesor Aranda, utiliza siete definiciones para explicar la implicación de atributos pero en este trabajo solo se van a usar las que más inciden en el mismo.

Definición 1.8 (Implicación entre atributos) *Sea $\mathbb{C} = (O, A, I)$ un contexto formal. Una implicación de atributos es un par de conjuntos $L, R \subseteq A$, y normalmente escritos $L \to R$. Una implicación $L \to R$ es válida en $\mathbb{C}$ si para todo objeto de $\mathbb{C}$ que tiene todos los atributos de L también tienen todos los atributos de R. A todas las implicaciones extraídas de un contexto $\mathbb{C}$ las denotaremos con el nombre de $Imp(\mathbb{C})$.*

Figura 2- Definición profesor Aranda. Extraído Aranda (2011) pág. 29

1 { } ⟹ UN
2 NW, UN ⟹ UNP, G8
3 UNP, UN ⟹ NW, G8
4 CT, G8, UN ⟹ EU
5 EU, UN ⟹ G8

Figura 3- Conjuntos de implicaciones. Extraído Aranda (2011) pág. 30

El conjunto de implicaciones que aparece en la Figura 2 proviene del contexto formal de la Figura 1, y por ejemplo la implicación 4 se lee como, todos los país que son miembro temporal (CT), miembro del G8 (G8), y miembro de las Naciones Unidas (UN) pertenecen a la Unión Europea (EU).

Una vez obtenido las implicaciones se debe de hablar de la consecuencia lógica y para ello se utilizan las reglas de Armstrong. El profesor Aranda (2011), las define como,

Definición 1.14 *Las reglas de Armstrong para implicaciones son:*

$$\frac{}{X \to X}\ \textit{(Identidad)} \qquad \frac{X \to Y}{X \cup Z \to Y}\ \textit{(Extensión)} \qquad \frac{X \to Y \quad Y \cup Z \to W}{X \cup Z \to W}\ \textit{(Substitución)}$$

con $W, X, Y, Z \subseteq A$.

Figura 4- Definición reglas de Armstrong. Extraído Aranda (2011) pág. 31

El último concepto que se debe abordar para entender la teoría en la que se apoya este trabajo son las reglas de asociación. Las reglas de asociación son implicaciones que satisfacen un conjunto de objetos del contexto. (Aranda, 2010). Dentro de las reglas de asociación se pueden

identificar dos partes, la primera es el soporte, número de objetos que cumplen las reglas, y la segunda es la confianza, que es el porcentaje de objetos que cumplen la premisa y las conclusiones.

2.2 Los sistemas de recomendaciones y sus tipos.

Los sistemas de recomendación son sistemas inteligentes que ofrecen al usuario unas sugerencias personalizadas sobre algún elemento en concreto. Melville y Sindhwani (2010) en su libro *"Recommender Systems: Encyclopedia of Machine Learning"* afirman que, "El objetivo de los sistemas de recomendación (SR) es generar recomendaciones válidas de elementos para un conjunto de usuarios a los que le puedan interesar".

Para llevar a cabo estas recomendaciones, los sistemas de recomendación utilizan la información obtenida del usuario, es decir, sus preferencias y sus comportamientos pasados en relación a los ítems y el propio sistema recomendador. En función al tipo de información mencionada, se clasifican los tipos de sistemas de recomendación. Para la clasificación, se va a utilizar parte de la expuesta por Charu C Aggarwa (2016), en su libro *"Recommender Systems: The Textbook":*

- Sistema de recomendación basado en filtro colaborativo: Basan la recomendación en un grupo de personas que le gusta lo mismo que al sujeto al que se le va a realizar la recomendación.

- Sistema de recomendación basado en contenido: Basan la recomendación en el análisis del contenido de los ítems (descrito por sus características o atributos) y en el perfil de usuario (que realiza un seguimiento de los elementos seleccionados anteriormente).

- Sistema de recomendación híbridos: Basan la recomendación en la combinación de los dos anteriores para mejorar la precisión de la recomendación.

En este trabajo se ha utilizado el sistema de recomendación basado en filtro colaborativo debido a que los datos con los que se trabaja se amoldan mejor a este tipo de sistemas.

3 Experimentación: La construcción de un sistema de recomendación.

Para llevar a cabo esta puesta en práctica de un sistema de recomendación de filtrado colaborativo se ha decidido seguir una hoja de ruta de cinco pasos (ver figura 1).

Figura 5- Hoja de Ruta. Elaboración Propia

3.1 Recogida de datos.

Los datos utilizados para este sistema de recomendación han sido proporcionados por una empresa, que por deseo expreso prefiere mantenerse en el anonimato. La empresa solicitó el diseño de un sistema de recomendación que les permitiera hacer ofertas personalizadas a su clientela actual.

3.2 Tratamiento y Procesado de datos.

El tratamiento y procesado de datos, segunda etapa de nuestra hoja de ruta, han sido realizados mediante scripts en Python. Esta fase es fundamental para que este sistema de recomendación pudiera ser creado. Esto se debe a que pone solución al principal problema que se detectó, que para una misma variable o categoría se había introducido los datos de manera distinta. Por lo que se procedió a realizar una serie de scripts, que se dividieron en tres grupos en función de las tablas proporcionadas por la empresa, para solucionar dicho problema.

3.2.1 Primer grupo: Clientes.

En este grupo se encuentra la información de los clientes proporcionada por la empresa y se compone de nueve scripts que serán explicados a continuación y adjuntados en el Anexo 1.

- Despliegue tabla “Clientes”: Este scripts se encarga de cargar los datos a la base de datos desde un fichero de texto plano a SqLiteStudio. Dicha base está compuesta por 17 variables de las cuales, 13 son propias de la misma base y las 4 restantes han sido creadas para ser utilizadas en la homogenización de los campos. Esta base se compone de 31.120 registros.

- Homogenización de variables: La homogenización se ha realizado en cinco scripts. Estos se encargan de homogenizar las variables NIF, apellidos, nombres, teléfono y teléfono2 e introducirlas en las variables xnif, xapellido, xnombres y xteléfono anteriormente creadas. Para llevar a cabo esta homogenización se han puesto ciertos criterios como

pueden ser quitar letras, espacios y distintos signos que fueron identificados con anterioridad como pueden ser guiones o puntos.

Ejemplo homogenización NIF:

490.21.34.11R → 490213411

- Copia tabla clientes: Este script se encarga de realizar una copia de la tabla de los clientes una vez realizada la homogenización para poder mantener los registros intactos mientras se realizan las manipulaciones pertinentes para los siguientes scripts.

- Clientes tienda: A la hora de realizar el sistema de recomendación resultó interesante sacar a los clientes que habían comprado en la tienda, ya que producían ruido dentro del sistema porque no se les puede aplicar la recomendación. Para ello se creó un script, el cual los introduce a todos en una nueva tabla, con objeto de tener constancia de ellos, y los elimina de la tabla original interior.

- Clientes repetidos: Este script se encarga de buscar NIF repetidos en la base de datos, los selecciona y los introduce en una tabla nueva para utilizarlos en otros scripts, y los borra de la tabla que fueron seleccionados.

- Idnueva: Una vez localizados los clientes repetidos, los clientes que compraron en la tienda y los clientes que solo compraron una vez se procedió a través de este script a proporcionarles una nueva ID teniendo en cuenta el tipo de cliente y así asignarle un único ID.

3.2.2 Segundo grupo: Factura.

Este grupo gestiona las facturas de cada cliente, se compone de tres scripts detallados a continuación y adjuntados en el Anexo 2.

- Despliegue tabla “Facturas”: Es el script encargado de cargar la tabla facturas en la base de datos. Esta tabla se compone de 21 variables, siendo las más relevantes el “cdfactura” y el “cdcliente”. Esta tabla contiene 33.176 registros.

- Tabla ID: Se encarga de crear una tabla nueva con la ID antigua y la nueva para poder utilizarla en el último scripts.

- Añadir ID nueva: Con este script se inserta la variable “Idnueva” en la tabla facturas para poder identificar la factura asociada a cada cliente y posteriormente poder saber que productos compraron.

3.2.3 Tercer grupo: Productos.

Este grupo recoge la información referente a los productos comprados por cada cliente, se compone de tres scripts detallados a continuación y adjuntados en el Anexo 3.

- Despliegue tabla “Productos”: El script de este apartado lleva a cabo el volcado de la tabla facturas en la base de datos. Esta tabla se compone de 9 variables, siendo las más relevantes el “cdfactura” y la “referencia”. Esta tabla contiene 75.195 registros.

- Productos devueltos: Con este script se pretende borrar los productos devueltos por los clientes para evitar ruido o falsas recomendaciones. Para ello se eliminan aquellos registros con la variable cantidad negativa.

- Generador de contextos: Este script genera un archivo en texto plano (.cxt), el cual recoge los contextos, que será utilizado por el programa “FCA_Solo_Reglas” para crear las reglas necesarias para poder hacer el sistema de recomendaciones.

Un ejemplo de estructura del archivo es:

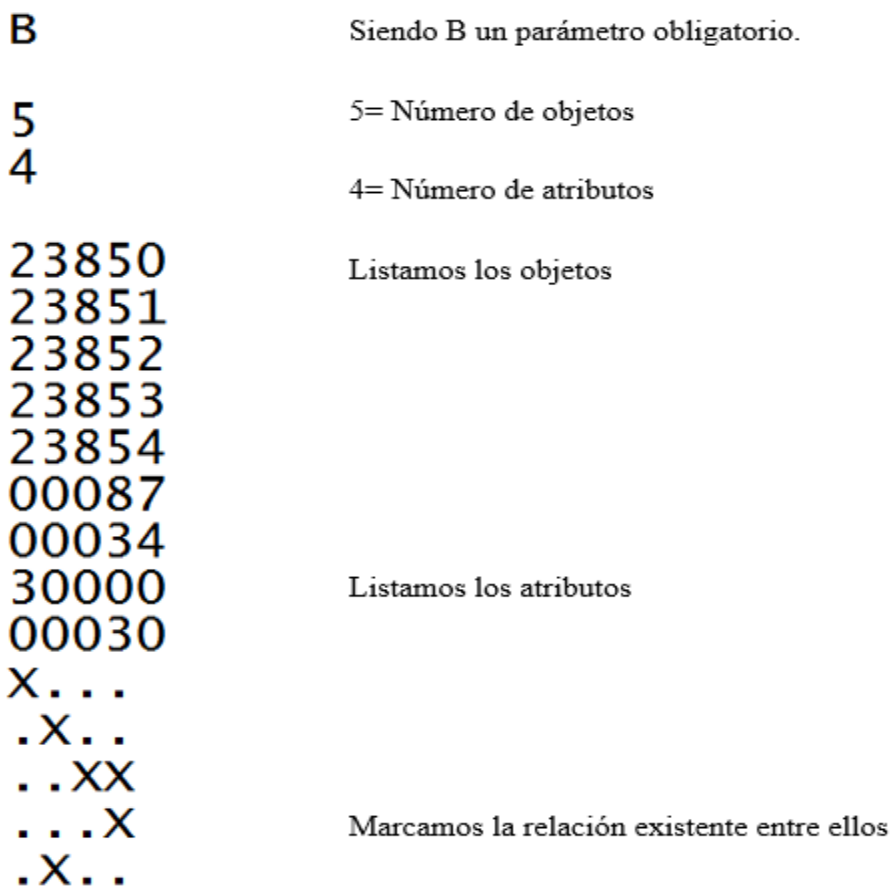

Figura 6- Ejemplo estructura archivo ".cxt". Elaboración propia

3.3 Entrenamiento

La etapa de entrenamiento ha sido dividida en dos fases. Por un lado, en la primera fase se establecen los criterios bases para diseñar el sistema de recomendaciones, es decir, se construye el entorno de trabajo. Y por el otro lado, en la segunda fase se procede a la ejecución de dicho sistema y al testeo del mismo.

Primero se va a proceder a explicar cómo se ha construido el entorno de trabajo del sistema de recomendación. Este entorno ha sido desarrollado a través del programa creado en Java y que se ejecuta desde línea de comandos llamado "FCA_Solo_Reglas.jar". Dicho programa proporciona las reglas de asociación que posteriormente se usarán para la creación del sistema de recomendaciones. La creación de reglas de dicho programa requiere la introducción del archivo generado en el script "Generador de contextos" en extensión (.cxt).

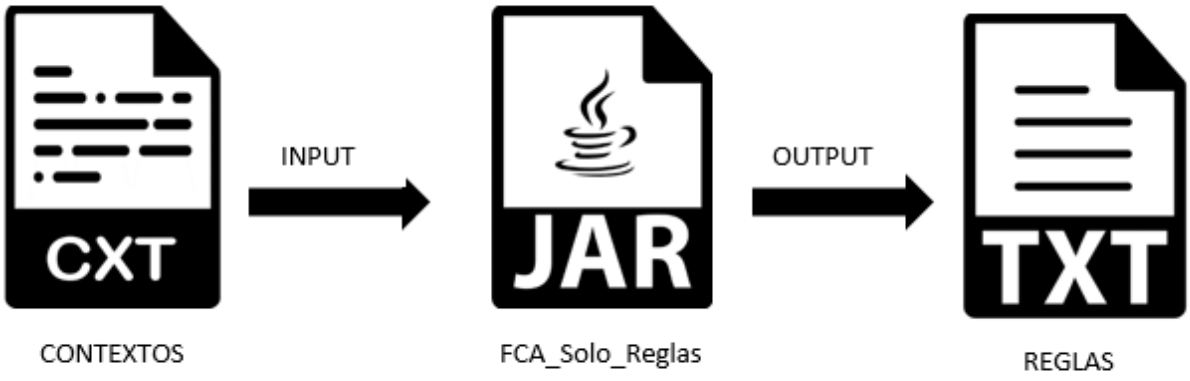

Figura 7- Proceso creación reglas de asociación. Elaboración propia

Una vez obtenidas las reglas de asociación se puede proceder a la segunda parte: la creación y el testeo del sistema de recomendación. La creación del sistema de recomendación se ha realizado a través de un script (véase Anexo 4), llamado sugerencias, en Python al cual se le proporcionan dos ficheros de texto para su ejecución y como resultado, devuelve otro fichero con las recomendaciones. De los dos ficheros que hay que proporcionarles uno debe contener los productos comprados por los clientes y el otro debe de ser el fichero creado por "FCA_Solo_Reglas" con las reglas de asociación.

Después de esta visión a grosso modo del funcionamiento del sistema, se va a proceder a profundizar en los componentes del script. El script se compone de cuatro funciones que al ejecutarlas en conjunto ofrecen la sugerencia o sugerencias pertinentes. La primera función que se puede ver en el script se llama "analizar", y se encarga de parsear las reglas de asociación para obtener los datos que se necesitan para el sistema. La segunda función, "cargar_reglas", abre el fichero con las reglas de asociación, lo lee línea a línea y le ejecuta la función "analizar" guardando los resultados en una lista de Python. La tercera función perteneciente al script, "cargar_compras", abre el fichero de las compras proporcionado al script y lo lee línea a línea. Para cada línea que lee, separa los distintos productos comprados por cada cliente, introduciéndolo en una lista de Python. La cuarta y última función utiliza las listas resultantes de las funciones "cargar_reglas" y "cargar_compras" para verificar si la nueva compra está dentro de un concepto existente y hacer la sugerencia de objetos a ofrecer para complementar dicha compra.

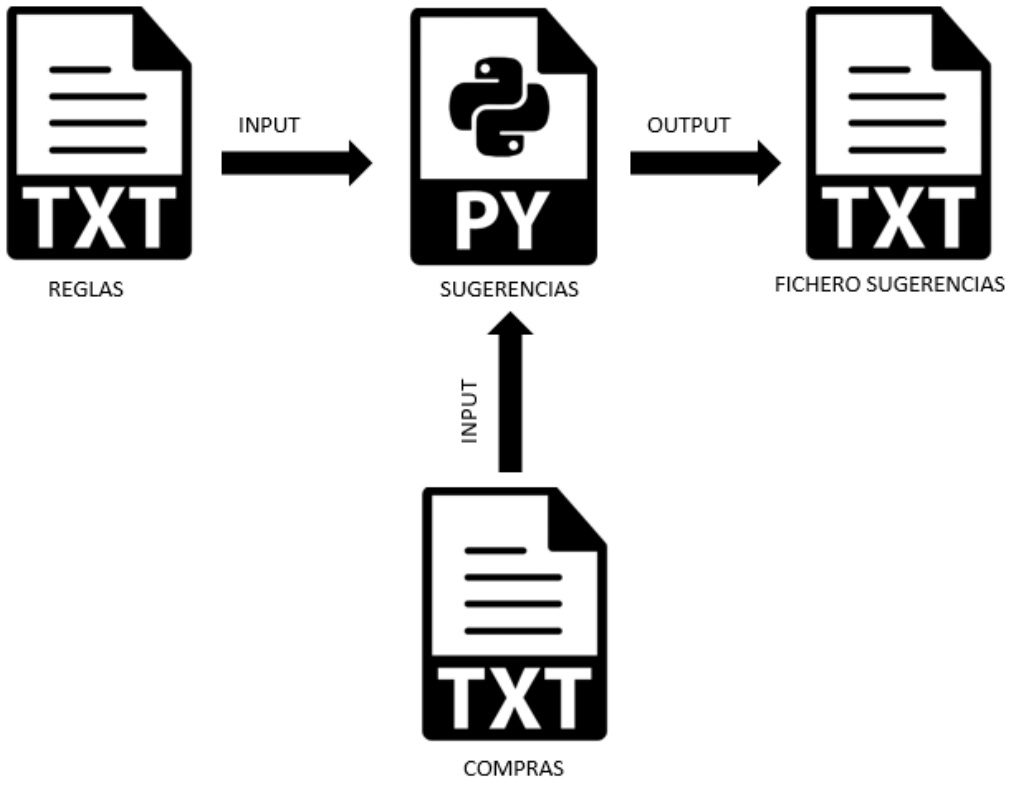

Figura 8 - Esquema funcionamiento script sugerencias. Elaboración propia

3.4 Visualización e interpretación de los datos.

La visualización e interpretación de las sugerencias se realizan a través del archivo de texto que genera nuestro sistema de recomendaciones, ya que el sistema no tiene una interfaz gráfica. El fichero muestra tanto las sugerencias a hacer a cada comprador como el grado de confianza de dicha sugerencia.

Las sugerencias muestran el número de referencia del producto, ya que si se llegará a implementar el sistema iría conectado a un sistema de integración de datos que tiene la empresa, el cual procesaría las referencias, haría la petición al servicio que hospeda la familia de productos y devolvería el nombre o familia del producto, el cual se mostraría directamente al cliente.

```
['00034', '00044']      ===>    [('30000', 100), ('30000', 25), ('00057', 16), ('00060', 16), ('10055', 8)]
['30000']       ===>    [('00030', 66), ('00034', 33), ('00057', 16), ('00060', 16), ('10055', 8)]
['00030']       ===>    [('30000', 100), ('30000', 25), ('00057', 16), ('00060', 16), ('10055', 8)]
```

Figura 9- Fichero de sugerencias. Elaboración propia

4 Ejemplo de ejecución del sistema de recomendación.

Este ejemplo se va a realizar únicamente sobre el sistema de recomendación, es decir, todo el proceso de procesado y tratamiento de datos ha sido realizado previamente pero no se mostrará.

Lo primero que se ha realizado es la selección del marco de trabajo. Para ello se ha ejecutado el script “Generador de contextos”. En este script se ha proporcionado dos fechas para que este seleccione los datos. Los datos seleccionado están comprendidos desde “2017-01-01” hasta el “2017-02-01”. Una vez generados los contextos ejecutamos desde línea de comandos el generador de reglas. Para ello se utiliza el comando “java -jar FCA_Solo_Reglas.jar tabla_recomendación.cxt >reglas.txt”.

Una vez obtenido las reglas de asociación (Figura 6) y nuestro fichero de compra (Figura 7). Se procederá a ejecutar el script sugerencias introduciéndoles dichos archivos. Para ello volveremos

al terminal y utilizaremos el comando “python sugerencias.py reglas.txt compras.txt >ficherosugerencias.txt”. Este guardará en el fichero “ficherosugerencias.txt” las sugerencias o recomendaciones a ofrecer al cliente (Figura 8).

```
4;23;00216;10098
3;20;00057,10067;00009
1;20;00057,10099;10056
1;20;00057,10099;30000,00030
1;20;00057,10266;30000,00030
2;18;10220;00000
1;16;00057,10056;10099
1;16;30000,00032;10272
1;14;00057,00009;10220
1;14;00057,10098;00010
2;13;00057,10067;10227
1;11;00057,10057;10102
1;10;10081;10067
1;10;10081;00050
1;8;10098;00005
1;8;00050;10057
14;8;;00051
1;7;00051;10057
1;7;00051;00216
```

Figura 11- Ejemplo de reglas generadas.

Elaboración propia.

```
10272,10271,10270,00059,00057,00087,00086,00051,00083
10067,10098,10057
00030,00057,00063
00001,10066
```

Figura 10- Ejemplo fichero de compras.

Elaboración propia.

```
['10272', '10271', '10270', '00059', '00087', '000051']        ├===>    [('30000', 100), ('00057', 100), ('00086', 100), ('00051', 100), ('00083', 100)]
['10067', '00034', '10098']     ===>    [('00057', 100), ('30000', 100), ('00057', 62), ('00057', 58), ('00057', 27)]
['00030', '30000']      ===>    [('00057', 36), ('00057', 27), ('00051', 8), ('10243', 1), ('00400', 1)]
['00063', '00057']      ===>    [('10067', 31), ('00051', 8), ('10243', 1), ('00400', 1), ('10264', 1)]
['00001', '10066', '00057']     ===>    [('00000', 35), ('10067', 31), ('30000', 25), ('00030', 25), ('10056', 25)]
```

Figura 12- Ejemplo de sugerencias generadas. Elaboración propia.

El fichero de sugerencias (Figura 8) se puede identificar dos grupos de datos, primero se observa las compras realizadas por el cliente y a continuación detrás de la flecha se puede ver las sugerencias a hacer al cliente con el grado de confianza de aceptación o dicho de otra manera la probabilidad de compra del producto.

Referencias bibliográficas

Aguilar, L. J. (2013). Big Data, Analisis de los grandes volumenes de datos. México: Alfaomega

Aranda-Corral, G. A. and Borrego-Díaz, J. (2010). Reconciling knowledge in social tagging web services. In Proc. 5th Int. Conf. on Hybrid AI Systems (HAIS 2010), volume 6077 of Lecture Notes in Computer Science, pages 383–390.

Aranda-Corral, G.A., Borrego Díaz, J., Galán Páez, J.: Confidence-based reasoning with local temporal formal contexts. In: Cabestany, J., Rojas, I., Joya, G. (eds.) IWANN 2011. LNCS, vol. 6692, pp. 461–468. Springer, Heidelberg (2011). doi: 10.1007/978-3-642-21498-1_58

Armstrong, W. W. (1974). Dependency structures of data base relationships. In IFIP Congress, pages 580–583.

Ganter, B., Wille, R.: Formal Concept Analysis - Mathematical Foundations. Springer, Heidelberg (1999)

Genesereth, M.R., Nilsson, N.J.: Logical Foundations of Artificial Intelligence. Morgan Kaufmann Publishers Inc., San Francisco (1987)

Guigues, J.L., Duquenne, V.: Familles minimales d'implications informatives résultant d'un tableau de données binaires. Mathématiques et Sciences Humaines **95**, 5–18 (1986)

Jäschke, R., Hotho, A., Schmitz, C., Ganter, B., and Stum, G. (2008). Discovering shared conceptualizations in folksonomies. Journal Web Semant., 6:38–53

Martín, D., Rosete, A., Alcalá-Fdez, J., & Herrera, F. (2014). QAR-CIP-NSGA-II: A new multi-objective evolutionary algorithm to mine quantitative association rules. Information Sciences, 258, 1–28

Nonaka, I., Takeuchi, H.: The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation. Oxford Univ. Press, Oxford (1995)

P. du Boucher-Ryan and D. Bridge. Collaborative recommending using formal concept analysis. Knowledge-Based Systems, 19(5):309–315, 2006.

P. Melville and V. Sindhwani, "Recommender Systems," In: C. Sammut and G. Webb, Eds., Encyclopedia of Machine Learning, Springer, Berlin, 2010, pp. 829-838.

Rodriguez Jimenez, JL. (2017). Extracción de conocimiento usando atributos negativos en el Análisis de Conceptos Formales Aplicaciones en la Ingeniería. Disponible online: https://riuma.uma.es/xmlui/bitstream/handle/10630/15447/TD_RODRIGUEZ_JIMENEZ_Jose_Manuel.pdf

Apéndice 1

- **Despliegue tabla “Clientes”:**

```
import sqlite3
con=sqlite3.connect('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')
with open('C:\Users\enrik\Desktop\TFM\clientes copia.sql','r') as foo:
    count=0
    foo2=foo.read().rsplit(';\n')
    for linea in foo2:
        count+=1
        con.execute(linea)
        print 'Lleva ' +  str(count)  + ' registro introducidos'

con.execute('update crf_clientes set cdcliente = trim(cdcliente)')
con.execute('update crf_clientes set nombre = trim(nombre)')
con.execute('update crf_clientes set apellidos = trim(apellidos)')
con.execute('update crf_clientes set domicilio = trim(domicilio)')
con.execute('update crf_clientes set localidad = trim(localidad)')
con.execute('update crf_clientes set cdprovincia = trim(cdprovincia)')
con.execute('update crf_clientes set codpostal = trim(codpostal)')
con.execute('update crf_clientes set telefono = trim(telefono)')
con.execute('update crf_clientes set telefono2 = trim(telefono2)')
con.execute('update crf_clientes set email = trim(email)')
con.execute('update crf_clientes set web = trim(web)')
con.execute('update crf_clientes set fax = trim(fax)')
con.commit()
con.close()
```

- **Homogenización de variables:**
  - **Apellidos:**

```
import pandas as pd
import sqlite3

con=sqlite3.connect ('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')
datos1=con.execute('select cdcliente from crf_clientes')
id=pd.DataFrame(datos1.fetchall())
for o in id[0]:
   surname=con.execute('select apellidos from crf_clientes where cdcliente=(?)', (o,))
   apellido =surname.fetchone()
   apellido = apellido[0].replace('.', '').replace(',', (' '))
   if  apellido[0][0]== ' ' :
      apellido=apellido[0].replace(' ', '')
   con.execute('update crf_clientes set xapellidos=(?) where cdcliente=(?)', (apellido, o))

con.commit()
con.close()
```

  - **Nombre**

```
import pandas as pd
import sqlite3

con=sqlite3.connect ('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')
datos1=con.execute('select cdcliente from crf_clientes')

id=pd.DataFrame(datos1.fetchall())

for o in id[0]:
```

```
    name=con.execute('select nombre from crf_clientes where cdcliente=(?)',(o,))
    nombre=name.fetchone()
    nombre = nombre[0].replace('.', '').replace(',', (' '))
    if nombre[0][0]== ' ':
        nombre=nombre[0].replace(' ', '')
    con.execute('update crf_clientes set xnombre=(?) where cdcliente=(?)',
(nombre, o))

con.commit()
con.close()
```

- **NIF:**

```
import pandas as pd
import sqlite3

con = sqlite3.connect('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')

datos1=con.execute('select cdcliente from crf_clientes')

cdcliente = pd.DataFrame(datos1.fetchall())

numeros = ["0","1","2","3","4","5","6","7","8","9"]

for registro in cdcliente[0]:
    dni=con.execute('select nif from crf_clientes where cdcliente=(?)',(registro,))
    dni1=dni.fetchone()
    resultado = ''
    for xx in dni1[0]:
        if xx in numeros:
resultado+=xx
if len(resultado)>=5:
    resultado1=resultado[:8]
```

```
            con.execute('update crf_clientes set xnif=(?) where cdcliente=(?)',(resultado1, registro))


con.commit()
con.close()
```

- **Telefóno:**

```
import pandas as pd
import sqlite3

con = sqlite3.connect('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')
datos1=con.execute('select cdcliente from crf_clientes')
con.execute('alter table crf_clientes  add xtelefono2 varchar(15)')
cdcliente =  pd.DataFrame(datos1.fetchall())

numeros = ["0","1","2","3","4","5","6","7","8","9"]

for registro in cdcliente[0]:
    phone= con.execute('select telefono from crf_clientes where cdcliente=(?) and telefono IS NOT NULL',(registro,))
    phone1= phone.fetchone()
    resultado = ''
    for xx in phone1[0]:
        if xx in numeros:
            resultado+=xx
            if len(resultado) >= 9:
                resultado1 = resultado[:9]
                con.execute('update crf_clientes set xtelefono=(?) where cdcliente=(?)', (resultado1, registro))

    phone2=con.execute('select telefono2 from crf_clientes where cdcliente=(?) and telefono2 IS NOT NULL', (registro,))
    phoneaux= phone2.fetchone()
    result = ''
```

```
    for yy in phoneaux[0]:
        if yy in numeros:
            result+=yy
            if len(result) >=9:
                result1= result[:9]
                con.execute('update crf_clientes set xtelefono2=(?) where cdcliente=(?)',
(result1, registro))

    con.commit()
    con.close()
```

- **Copia tabla clientes:**

```
import sqlite3

con=sqlite3.connect('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')

con.execute('drop table if exists `copia_cliente`')
con.execute('create table if not exists  copia_cliente as select * from crf_clientes where
1=0')
con.execute('insert into copia_cliente select * from crf_clientes')

con.commit()
con.close()
```

- **Clientes tienda:**

```
import sqlite3

con=sqlite3.connect('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')

con.execute('drop table if exists `cliente_tienda`')

con.execute('create table if not exists cliente_tienda as select * from copia_cliente where
1=0')

con.execute("insert into cliente_tienda select * from copia_cliente where xnombre like
'%TIEN%' or xnombre= '%COMP%'  or xapellidos like 'TIEN%' or xnombre like '0%' or
xapellidos like '0%' and (xnif='' or xnif like '0%')" )
```

```
con.execute("delete from crf_clientes where xnombre like '%TIEN%' or xnombre= '%COMP%'  or xapellidos like 'TIEN%' or xnombre like '0%' or xapellidos like '0%' and (xnif='' or xnif like '0%')")
con.commit()
con.close()
```

- **Clientes repetidos:**

```
import sqlite3
import pandas as pd

con=sqlite3.connect('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')

xnif=con.execute('select xnif from copia_cliente')
xnifdt=pd.DataFrame(xnif.fetchall())

xnifdt=xnifdt.rename(columns={0:'nif'})
xnifdt=xnifdt.sort_values('nif')
xnifdt['counts'] = 0
xnifdt['counts'] =xnifdt.groupby(['nif'])['counts'].transform('count')

xnifdt2=xnifdt['counts'] < 18
xnifdt3=xnifdt['counts'] > 1

repetidos=xnifdt[xnifdt2&xnifdt3]
con.execute('drop table if exists `repetidos`')
con.execute('create table if not exists  repetidos as select * from copia_cliente where 1=0')

aux=0
for y in repetidos['nif']:
    if aux!=y:
        con.execute('insert into repetidos select * from copia_cliente where xnif=(?) order by xnif', (y,))
        con.execute("delete from crf_clientes where xnif=(?)", (y,))
```

```
        aux=y
    else:
        next

con.commit()
con.close()
```

- **Idnueva:**

```
import sqlite3
import pandas as pd

con=sqlite3.connect('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')

con.execute('drop table if exists `idnueva`')
con.execute('alter table repetidos add idnueva numeric(15)')
con.execute('create table idnueva as select * from copia_cliente where 1=0')
con.execute('alter table idnueva add idnueva numeric(15)')
con.execute('alter table cliente_tienda add idnueva numeric(15)')
con.execute('alter table crf_clientes add idnueva numeric(15)')

y=''
idnueva=1

datos1=con.execute('select cdcliente from repetidos')
id=pd.DataFrame(datos1.fetchall())


for o in id[0]:
    nif=con.execute('select xnif from repetidos where cdcliente=(?)', (o,))
    nif1=nif.fetchone()
    if y=='':
        y=nif1[0]
    if nif1[0] == y:
```

```
        con.execute('update repetidos set idnueva=(?) where cdcliente=(?)', (idnueva, o))
    elif nif1[0]!= y:
        y = ''
        idnueva+=1
        con.execute('update repetidos set idnueva=(?) where cdcliente=(?)', (idnueva, o))

con.execute('insert into idnueva select * from repetidos')
datos2=con.execute('select cdcliente from cliente_tienda')
idtienda=pd.DataFrame(datos2.fetchall())

idnueva+=1
for t in idtienda[0]:
    tienda=con.execute('select xnombre from cliente_tienda where cdcliente=(?)',(t,))
    shop=tienda.fetchone()
    con.execute('update cliente_tienda set idnueva=(?) where cdcliente=(?)', (idnueva,t))
con.execute('insert into idnueva select * from cliente_tienda')

datos3=con.execute('select cdcliente from crf_clientes')
idclientes= pd.DataFrame(datos3.fetchall())

for w in idclientes[0]:
    idnueva+=1
    cliente=con.execute('select xnombre from crf_clientes where cdcliente=(?)',(w,))
    costumer=cliente.fetchone()
    con.execute('update crf_clientes set idnueva=(?) where cdcliente=(?)', (idnueva, w))
con.execute('insert into idnueva select * from crf_clientes')

con.commit()
con.close()
```

Apéndice 2

- **Despliegue tabla “Facturas”:**

```
import sqlite3
con=sqlite3.connect('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')

with open('C:\Users\enrik\Desktop\TFM\crf_facturas.sql','r') as foo:
    count=0
    foo2=foo.read().rsplit(';\n')
    for linea in foo2:
        count+=1
        con.execute(linea)
        print 'Lleva ' + str(count) + ' registro introducidos'

con.execute('update crf_facturas set cdfactura= trim(cdfactura)')
con.execute('update crf_facturas set fechafactura= trim(fechafactura)')
con.commit()
con.close()
```

- **Tabla ID:**

```
import sqlite3
con=sqlite3.connect('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')
con.execute('drop table if exists `ID`')
con.execute('create table ID as select idnueva, cdcliente from idnueva')
```

- **Añadir nueva ID**:

```
import sqlite3
import pandas as pd
con=sqlite3.connect('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')
#con.execute('alter table crf_facturas add idnueva numeric(15)')
idantigua=con.execute('select cdcliente from ID')
cdcliente=pd.DataFrame(idantigua.fetchall())
```

```
count=0
for id in cdcliente[0]:
    count+=1
    idnew=con.execute('select idnueva from ID where cdcliente=(?)', (id, ))
    idnueva=idnew.fetchone()
    con.execute('update crf_facturas set idnueva=(?) where cdcliente=(?)', (idnueva[0],id))
    print 'Actualizado '+ str(count) + ' registros'
con.commit()
con.close()
```

Apéndice 3

- **Despliegue tabla “Productos”:**

```
import sqlite3
con=sqlite3.connect('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')

with open('C:\Users\enrik\Desktop\TFM\crf_lineas.sql','r') as foo:
    count=0
    foo2=foo.read().rsplit(';\n')
    for linea in foo2:
        count+=1
        con.execute(linea)
        print 'Lleva ' + str(count) + ' registro introducidos'
con.execute('update crf_lineas set cdfactura = trim(cdfactura)')
con.execute('update crf_lineas set referencia = trim(referencia)')
con.commit()
con.close()
```

- **Productos devueltos:**

```
import sqlite3
con=sqlite3.connect('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')
con.execute('delete from crf_lineas where cantidad<=-1')
con.commit()
con.close()
```

- **Generador de contextos**:

```
import sqlite3
from datetime import datetime,timedelta

t=open("C:\Users\enrik\Desktop\TFM\\tabla_recomendacion.cxt",'w')

con=sqlite3.connect('C:\Users\enrik\Desktop\TFM\Rupe_DS.db')
con.execute('drop table if exists `recomendacion`')

inicio=raw_input('Introduzca una fecha de inicio Ej: 2002-08-12: ')
fin=raw_input('Introduzca una fecha de fin Ej: 2002-08-12: ')


inicio=datetime.strptime(inicio,"%Y-%m-%d")
fin=datetime.strptime(fin,"%Y-%m-%d")

t.write('B')
t.write('\n')
```

```
t.write('\n')
foo= con.execute('select cdfactura from crf_facturas where fechafactura >(?) and
fechafactura<= (?)', (inicio, fin+ timedelta(days=1)))
fact=foo.fetchall()
facturas=[]
for f in fact:
   facturas.append(f)

t.write(str(len(facturas))+'\n')

cadena = "select distinct referencia FROM crf_lineas,crf_facturas WHERE
crf_lineas.cdfactura = crf_facturas.cdfactura AND crf_facturas.fechafactura and
fechafactura > '%s' and fechafactura<= '%s'" % (inicio, fin)

foo= con.execute(cadena)
lc=foo.fetchall()
referencias=[]
for r in lc:
   referencias.append(r)
t.write(str(len(referencias))+'\n')

t.write('\n')
for f in facturas:
   t.write(str(f[0])+'\n')

for r in referencias:
   t.write(str(r[0])+'\n')

count=0
for f in facturas:
   cadena = "select referencia FROM crf_lineas WHERE cdfactura=%d" % f
   foo = con.execute(cadena)
   lc = foo.fetchall()
   reffactura = []
   for r in lc:
      reffactura.append(r)

   for rr in referencias:
      if rr in reffactura:
         print 1,
         t.write('X')
         count+=1
      else:
         print ".",
         t.write('.')
         count+=1
      if count==len(referencias):
```

```
            t.write('\n')
            count=0
    print
t.close()
```

Apéndice 4

Sistema de recomendación:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import sys


def analizar(regla):
    lista = regla.split(";")
    soporte = int(lista[0].strip())
    confianza = int(lista[1].strip())
    antecedentes = lista[2].strip().split(",")
    consecuentes = lista[3].strip().split(",")
    return (soporte,confianza,antecedentes,consecuentes)

def cargar_reglas(camino):
    resultado = []
    f = open(camino)
    for linea in f:
        resultado.append(analizar(linea))
    f.close()
    return resultado


def cargar_compras(camino):
    resultado = []
    f = open(camino)
    for linea in f:
        resultado.append([cc.strip() for cc in linea.strip().split(",")])
    f.close()
    return resultado

def sugerencias(reglas, compra):
    resultado = []
    for r in reglas:
        cumple = True
        for ant in r[2]:
            if not ant == "":
                cumple = cumple and (ant in compra)
        if cumple:
            for mix in r[3]:
                if not mix in compra:
                    resultado.append((mix,r[1]))
```

```
    resultado.sort(key=lambda a: a[1], reverse=True)
    return resultado[:5]



'''************************************
PROGRAMA PRINCIPAL

************************************ '''
if __name__ == "__main__" :
    if len(sys.argv) <3:
        print "Error: Necesita, al menos, 2 parámetros para ejecutar el programa"
        print "Formato: python sugerencias.py ficheroreglas ficherocompra "
    else:
        reglas = cargar_reglas(sys.argv[1])
        compras = cargar_compras(sys.argv[2])
        for c in compras:
            if len(c) == 1 and c[0] == "":
                print ""
            else:
                if len(c) > 0 :
                    ss = sugerencias(reglas,c)
                    print c, "\t===>\t",ss
                else:
                                print ""

'''************************************
PROGRAMA PRINCIPAL

************************************ '''
if __name__ == "__main__" :
    if len(sys.argv) <3:
        print "Error: Necesita, al menos, 2 parámetros para ejecutar el programa"
        print "Formato: python sugerencias.py ficheroreglas ficherocompra "
    else:
        reglas = cargar_reglas(sys.argv[1])
        compras = cargar_compras(sys.argv[2])
        for c in compras:
            if len(c) == 1 and c[0] == "":
                print ""
            else:
                if len(c) > 0 :
                    ss = sugerencias(reglas,c)
                    print c, "\t===>\t",ss
                else:
                                print ""
```