Algoritmos de venta aplicados a Ecommerce

por

Diego Fernández Gil

Tesis presentada en conformidad con los requisitos del
Máster en economía, finanzas y computación.

Universidad de Huelva

Universidad Internacional de Andalucía

Noviembre de 2017

Algoritmos de ventas aplicado a Ecommerce

Diego Fernández Gil

Master en económia finanzas y computación especialidad Economía

Dr. Manuel Emilio Gegúndez Arias

2017

Abstract

This document reflects the research and application work developed for an online sales company that has the commercialization of technological products whose final objective is the basis of the sales bases of the cross and up sale systems. Through this type of techniques, the double objective of providing a greater personalized experience for the costumer and improving the company's turnover through the intelligent discovery of the catalogue can be achieved. For this, algorithms programmed in Python language have been used. All this requires a programming language that can be consulted in the field of data mining to establish a specific methodology for this case and to be able to develop efficient and functional applications. In addition to this, you must go on with a selection criteria for the company to establish the elements that can be used, the way in which the script can be implemented and the frequency of execution, locally or in the cloud. and of course, fixing a type of measure with which to check the effectiveness of the applications.

Keywords: Up selling, cross selling, Apriori, support, confidence, lift, ETL.

Resumen

El presente documento refleja el trabajo de investigación y aplicación desarrollado para una empresa de venta online dedicada a la comercialización de productos tecnológicos cuyo objetivo último es sentar las bases de lo que será los sistemas de venta cruzada y up selling. Mediante este tipo de técnicas se pretende lograr el doble objetivo de brindar una mayor experiencia personalizada para el cliente y mejorar la facturación de la empresa mediante el descubrimiento inteligente del catalogo. Para ello se han utilizado algoritmos programados en lenguaje Python. Todo ello no solamente requiere del manejo de un lenguaje de programación sino que se debe acudir al campo de la mineria de datos para establecer una metodología específica para este caso y poder desarrollar unas aplicaciones eficientes y funcionales. Además de esto, se debe continuar con unos criterios discrecionales por parte de la empresa para establecer que elementos son los que se prefiere mostrar, la manera en la que se va a implementar el script ya sea la frecuencia de ejecución, en local o en la nube y por supuesto el fijar algún tipo de medida con la que comprobar la efectividad de las aplicaciones.

Palabras Clave: Up selling, cross selling, Apriori, soporte, confianza, lift, ETL.

Índice

1. Introducción **4**

2. Antecedentes **5**

3. Conceptos **6**

4. Medidas **8**

4.1. Soporte 8

4.2. Confianza 9

4.3. *Lift* 10

5. Marco teórico **11**

6. Ejemplo práctico algoritmo **13**

7. Aplicación del algoritmo **17**

7.1. Preprocesado 17

7.2. Postprocesado 19

8. Aplicación **21**

9. Trabajos futuros **22**

10. Conclusiones **23**

Bibliografía **23**

11. Anexo **25**

11.1. Código 25

1. Introducción

El eje central de este trabajo gira en torno a una solución técnica que cada vez toma más importancia en los negocios con este tipo de características y se enmarca dentro de las denominadas técnicas de venta. En este texto se desarrollan en concreto las llamadas ventas de productos relacionados o más comúnmente conocidas en el sector como cross-selling y up-selling y su misión no es otra que aumentar el valor medio y la rentabilidad del "carrito de la compra".

Estos conceptos aunque en principio puedan parecer desconocidos, si uno compra en cualquier página dedicada al comercio online, más de una vez habrá sido "víctima" de este tipo de técnicas de venta. Cuando nosotros vamos a una página dedicada a la comercialización de productos y buscamos algún artículo, es muy probable que en las parte donde más se centre nuestra atención como por ejemplo la descripción del producto o la foto del mismo se muestre una serie de artículos relacionados que o bien complementa ese artículo (cross-selling) o bien nos ofrece productos similares pero con mejores características conllevando a la par la mayoría de las veces un mayor precio (up-selling). Estás técnicas aunque en principio puedan parecer similares, a la hora de desarrollar una aplicación son totalmente diferentes y su implementación dependerá en gran medida tanto de la cantidad de datos disponibles como la estructura de los mismos. Una vez identificados, a continuación, se describe de forma general ambos conceptos:

- *Cross-selling*: también conocido como el multiplicador de ventas o la venta cruzada, se fundamenta en ofrecer al comprador de una tienda online varios productos complementarios, es decir, artículos que se pueden utilizar para mejorar o añadir algunas características del producto elegido.
- *Up-selling*: que consiste en mostrar al cliente productos similares o más rentables para el propio comercio (no tienen por qué ser más caros). En otras palabras, podemos decir que la tienda intentará persuadir a los clientes para que elijan productos en los que tiene más margen de beneficio o simplemente aquellos artículos los cuales hay que reducir el stock en almacén con más celeridad debido a varias situaciones como por ejemplo podría ser la compra de cantidades por encima de lo normal para aprovecharse de algún tipo de descuento por rappel, etc.

2. Antecedentes

A la hora de abordar las dos soluciones se pueden diferenciar por sus características 2 casos totalmente contrarios si nos enfocamos en el desarrollo de sus aplicaciones. Por el lado del up-selling existe escasa literatura establecida para la creación de un algoritmo más alla de técnicas de visualización, ya que llevando el problema al caso más simple solo se trataría de un algoritmo de ordenación. Queda más bien a elección de la empresa el fijar los criterios por el que se va a regir lo que se muestra al usuario. En este caso se ha optado por similitud con la marca, una horquilla de precio y otro campo más llamado relevancia el cual posee cada artículo y lleva una formula interna desarrollada por la empresa que tiene en cuenta diferentes factores de notoriedad como podrían ser las opiniones de los usuarios sobre ese artículo, las veces que se está viendo o comprando recientemente o si tiene algún tipo de descuento.

No obstante para los algoritmos de cross-selling (o venta cruzada) el problema radica principalmente en que cuando existe una cantidad de artículos demasiado extensa y a esto, se le añade una rotación y catalogación de productos muy activa como es el caso de una empresa dedicada a la comercialización de productos tecnológicos, se hace laborioso y costoso dedicar recursos a buscar y enlazar artículos que se puedan vender conjuntamente. La solución de este problema no solamente tiene que encontrar estas asociaciones anteriormente mencionadas, sino que además debe de poder hacerse automáticamente y con la mayor frecuencia posible para que todos los artículos activos tengan asociados sus productos accesorios correspondientes.

En esta línea existen precedentes que se citarán en las siguientes secciones y que han intentado solventar esta problemática mediante la vía del descubrimiento de patrones frecuentes. No obstante, cabe destacar que recientemente están surgiendo nuevos enfoques para intentar solucionar este tipo de problemas como por ejemplo la minería de texto o el uso de bases de datos orientadas a grafos.

La elección de la metodología correcta debe de tener en cuenta diferentes aspectos a la hora de abordar una solución como por ejemplo los recursos disponibles o el coste de oportunidad que puede llevar la implantación en términos de tiempo-beneficio e incluso coste-beneficio.

En este trabajo en particular se ha optado por el análisis del histórico de compras para el descubrimiento de patrones frecuentes debido a la información disponible, que favorece el uso de este tipo de técnicas además de su productividad medido en tiempo de coste ya que su ejecución puede hacerse mediante los medios disponibles por la empresa. Además de la literatura existente la cual sentó las bases de este campo de la minería de datos y que en la siguiente sección introduce los conceptos más extendidos para este marco.

3. Conceptos

Como se ha comentado anteriormente la asociación es una clase de problema de minería de datos que trata de encontrar patrones frecuentes a lo largo de un conjunto de datos. El termino reglas de asociación son atribuidas al trabajo de Agrawal, Srikant et al (1994) [1] en su trabajo: *Mining asociation rules between sets of ítems in large databases*. En él, los autores realizarón una aplicación práctica junto a la empresa americana IBM donde analizaban información de ventas al por menor. Mediante ella se descubrían las relaciones entre los datos recopilados a gran escala por los sistemas de terminales de punto de venta de supermercados. Los datos consistían en colecciones de transacciones, también conocidas como bases de datos transaccionales, donde cada transacción expresa que productos compró un cliente. Asi, los responsables del establecimiento, una vez conocidas estas asociaciones distibuian estrategicamente los productos en los puntos de venta con el fin de aumentar la probabilidad de compra de ciertos productos, por ejemplo poniendo dos productos que se compran muy a menudo juntos como pueden ser lácteos con cereales lo más separado posible para que el cliente recorriera el supermercado y visualizara más productos. O situar dos productos en la misma estantería para estimular su compra conjunta. Este criterio se establecía en base a la información proporcionada por las compras de los individuos y la frecuencia con la que se repetían los artículos a lo largo del conjunto de datos analizado, por lo que a la tarea de descubrir reglas de asociación en transacciones comerciales se denominó "análisis de la cesta de compra" *(market basket analysis)*.

Formalmente:

- Sea $I = \{i_1, i_2, i_3, \ldots, i_{n-1}, i_n\}$ un conjunto de ítems, en este caso todos los artículos que se han vendido en el período de análisis establecido y que todavía estén activos.
- Sea $T = \{t_1, t_2, t_3, \ldots, t_{m-1}, t_m\}$ un conjunto de transacciones donde cada elemento "t" perteneciente a T se corresponde con un conjunto de ítems presentes en I, tal que $t \subseteq I$. Aquí representaría cada carrito de la compra realizado y vendría representado por t_m.
- Conjunto de datos transaccional: Un conjunto transaccional se puede representar por una matriz donde cada fila representa una transacción diferente, y a su vez en cada una de ellas representa los artículos que se contienen:

$$\left|\begin{array}{c|cccccc} T_1 & i_{11} & i_{12} & i_{13} & i_{14} & \ldots & i_{1n} \\ T_2 & i_{21} & i_{22} & i_{23} & i_{24} & \ldots & i_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ T_m & i_{m1} & i_{m2} & i_{m3} & i_{m4} & \ldots & i_{mn} \end{array}\right|$$

$$i_{ij} = \begin{cases} 1 & \text{Sí el item j está presente en la transacción i} \\ 0 & \text{en otro caso} \end{cases}$$

- Asimismo, en la formalización llevada a cabo por Agrawal, Srikant et al (1991) [1], considera una regla de asociación como una implicación donde $A \Rightarrow B$ donde A y B son un conjunto de items que a su vez es subconjunto de "t", o sea $A, B \subseteq t$, o lo que es lo mismo, el conjunto de ítems (o ítem) que pertenece al conjunto A (de ahora en adelante antecedente), en una compra implica la existencia de los ítems (o ítem) que pertenecen al conjunto B(consecuente), donde se puede deducir facilmente que: $A \subseteq I$ y $B \subseteq I$. Es importante señalar que la intersección de A y B es $A \cap B = \phi$, un conjunto vacio, o lo que es lo mismo, si un ítem o conjunto de ítems se encuentran en A, este o estos no pueden pertenecer a B ya que un item no puede ser consecuente de sí mismo.

Como se ha podido observar, se ha diferenciado explícitamente ambos lados de la regla de asociación para determinar un orden lógico y así favorecer una lectura comprensible de la regla. El lado izquierdo (o antecedente) se asimilaría como el ítem principal y para el conjunto del lado derecho (consecuente) como bien indica su nombre la consecuencia de haber elegido A. Para nuestro ejemplo imaginemos una persona que elige un Smartphone. Este sería nuestro conjunto principal o A. Si analizamos el conjunto de transacciones, en la mayoría de ellas se observarían ciertos patrones que se repiten con bastante frecuencia ya que se hace muy común que un individuo compre algunos objetos para complementar su Smartphone como una funda protectora para proteger la pantalla y una memoria extraíble para ampliar su capacidad de almacenamiento, por lo que muy probablemente un consecuente del antecedente Smartphone sea carcasa, tarjeta SD o como hemos ilustrado anteriormente:

$$\{A\} \Rightarrow \{B\} : \{\text{Smartphone}\} \Rightarrow \{\text{Carcasa, Tarjeta SD}\}$$

De este modo una regla de asociación puede ser considerada una relación lógica de implicación. No obstante, a la hora del análisis pueden surgir diferentes casuísticas que vicien el algoritmo y no se obtengan reglas de asociación validas, por lo que se hace necesario utilizar otro tipo de medidas con el fin de garantizar la fiabilidad de una regla y poder discriminar para cada conjunto de ítems cuales son las mejores. Para ello existen diferentes reglas de decisión que han ido aumentando conforme se ha profundizado más en este campo y a continuación pasaremos a definir las usadas en este trabajo.

4. Medidas

Uno de los principales problemas que existen y que evidentemente se han manifestado en este trabajo, cuando se trabaja con este tipo de técnicas y se dispone de un conjunto de artículos tan elevados junto a un gran volumen de transacciones es la enorme cantidad de reglas que pueden surgir. Es por ello que se hace necesario establecer unos criterios que nos aporten información de algún modo sobre la fiabilidad de las reglas generadas y no recomendar artículos que tengan poco o nada que ver. En esta línea se ha investigado mucho sobtre las métricas que se deben usar y cuáles son las mejores según para qué y por supuesto la discusión de las mismas daría para otro trabajo completo. Algunos de los trabajos más interesantes respecto a ello se pueden encontrar en Hilderman y Hamilton (2001) [7] y Berzal et al (2001) [8].

Cabe destacar que en este trabajo las medidas utilizadas como parámetros de entrada por los algoritmos empleados para estos análisis son: soporte, confianza y *lift*.

4.1. Soporte

El soporte es una medida que contabiliza la frecuencia en la cual los términos de una regla de asociación se encuentran en los datos, es decir, el número de transacciones en las cuales los ítems presentes en una regla ocurren juntos en los datos en relación con el número total de transacciones. Evidentemente con esta medida no existe un criterio claro con el que trabajar ya que su resultado depende en gran medida del tamaño del conjunto de datos. Este tipo de medida se utiliza más como parámetro de entrada para establecer que artículos se tienen en cuenta a la hora de realizar el análisis y así aligerar el coste de computo. Un ejemplo sería en el caso hipotético de tener un conjunto de datos con 100 transacciones y fijar el soporte en 0.10 con lo cual, los articulos que no superaran esa frecuencia no pasarían a formar parte del análisis. Para nuestro caso, el soporte se ha fijado en un umbral mínimo para tener en cuenta todos los artículos registrados y no perder detalle alguno.

4.2. Confianza

La medida de confianza se refiere a un valor de correspondencia entre los ítems que componen una regla, es decir, la medida denota el porcentaje de transacciones que contienen conjuntamente el termino antecedente y el término consecuente en relación al número de transacciones que contienen la parte antecedente. La confianza puede ser obtenida como se denota a continuación:

$$\text{conf}(A \Rightarrow B) = \frac{\text{Soporte(A,B)}}{Soporte(A)}$$

Como se puede observar, desde un punto probabilístico, la confianza no es más que la probabilidad condicional que nos indica la probabilidad de que ocurra un evento A, sabiendo que ha ocurrido B. Algebraicamente:

$$\text{P}(A|B) = \frac{P(A \cap B)}{P(B)}$$

De modo que si tenemos un conjunto de 50 transacciones donde el item i_1 ha aparecido 6 veces, y por el otro lado tenemos el item i_2 que aparece 4 veces y en tres ocasiones conjuntamente con i_1, tenemos:

- Total transacciones = 50
- Soporte(A) = $\frac{6}{50} = 0.12$
- Soporte(B) = $\frac{4}{50} = 0.08$
- Soporte(A,B) = $\frac{3}{50} = 0.06$
- Conf($A \Rightarrow B$) = $\frac{0,06}{0,12} = 0.5$

Es decir, de las veces en las que se eligió A en un 50 % de las transacciones aparecía B.

4.3. *Lift*

El lift, es una medida utilizada para evaluar el grado de dependencia de los términos de una regla, para ello se basa en la definición de sucesos independientes. Recordemos que en teoría de probabilidades, se dice que dos sucesos aleatorios son independientes entre sí cuando la probabilidad de cada uno de ellos no está influida por el hecho de que otro suceso ocurra o no, es decir, cuando ambos sucesos no están relacionados.

Como definición formal sabemos que: dos sucesos son independientes si la probabilidad de que ocurran ambos simultáneamente es igual al producto de las probabilidades de que ocurra cada uno de ellos, es decir, si $P(A \cap B) = P(A) * P(B)$

Por lo tanto, si definimos el *lift* como:

$$\frac{\mathrm{P(A \cap B)}}{P(A) * P(B)}$$

Concluyendo que:

- Si *lift* > 1, muy probablemente los sucesos son dependientes el uno del otro
- Si *lift* = 1, los sucesos posiblemente sean independientes
- Si *lift* < 1, los sucesos posiblemente sean excluyentes

Resumiendo las medidas a utilizar quedarían como:

Soporte	**Confianza**	***Lift***
P(A ∩ B)	$\frac{\mathrm{Soporte(A,B)}}{Soporte(A)}$	$\frac{\mathrm{P(A \cap B)}}{P(A) * P(B)}$

5. Marco teórico

Existe una rica y extensa literatura en el campo de la minería de datos y más concretamente en el descubrimiento de patrones frecuentes. Se puede ver una revisión de ellos en Hidber (1999)[6]. Estás técnicas tratan de hallar asociaciones entre dos entidades diferentes que se repiten continuamente dentro de un determinado conjunto de datos. Dentro de ellos destacan dos algoritmos específicos para este caso:

- Algoritmo Apriori: propuesto por Agrawal, Srikant et al (1994)[1], se basa en la obtención de los denominados conjuntos de ítems frecuentes, los cuales son aquellos que satisfacen un requisito fijado previamente que establece que se debe de cumplir un mínimo de repeticiones donde ambos ítems aparezcan juntos en el conjunto de datos para ser tomado en cuenta como una asociación.

  Su funcionamiento representado en una secuencia de flujo quedaría como:

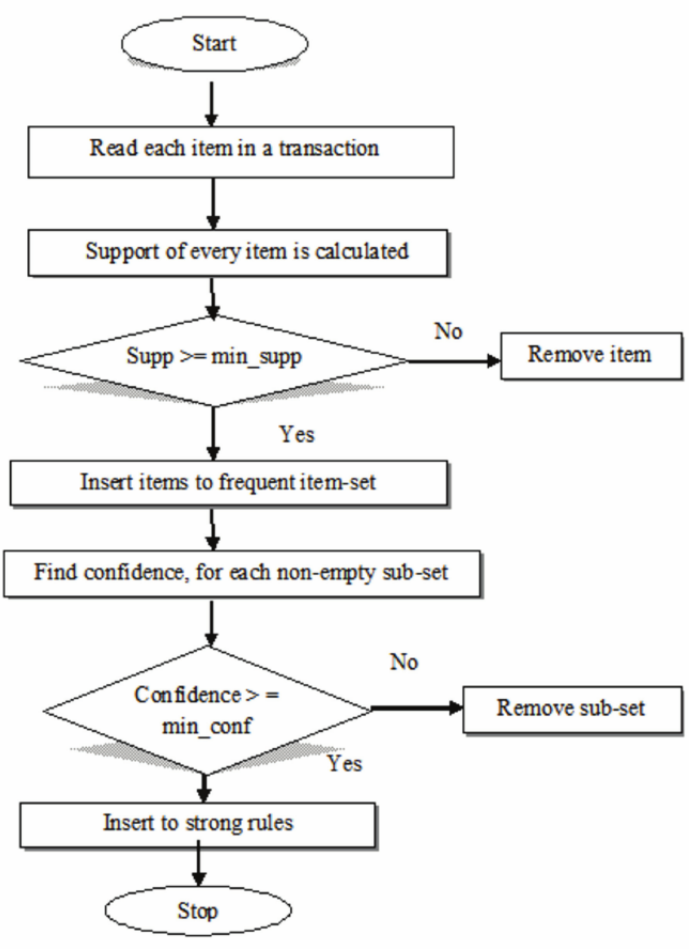

Figura 1: Secuencia de flujo algoritmo apriori [9]

- Algoritmo FP-Growth: planteado por Han, Pei et al (2000)[5] Se basa en la construcción de un conjunto de datos diferente llamado FP-Tree donde se almacena la información de los ítems frecuentes de manera compacta. Para cada conjunto de ítems frecuentes se ordena de más probable a menos probables junto a un contador y se van creando ramificaciones en ese conjunto de datos obteniendo un almacenamiento más eficiente. Después de obtener todos los conjuntos de ítems frecuentes, el algoritmo obtiene las reglas de asociación de manera análoga a la del algoritmo Apriori.

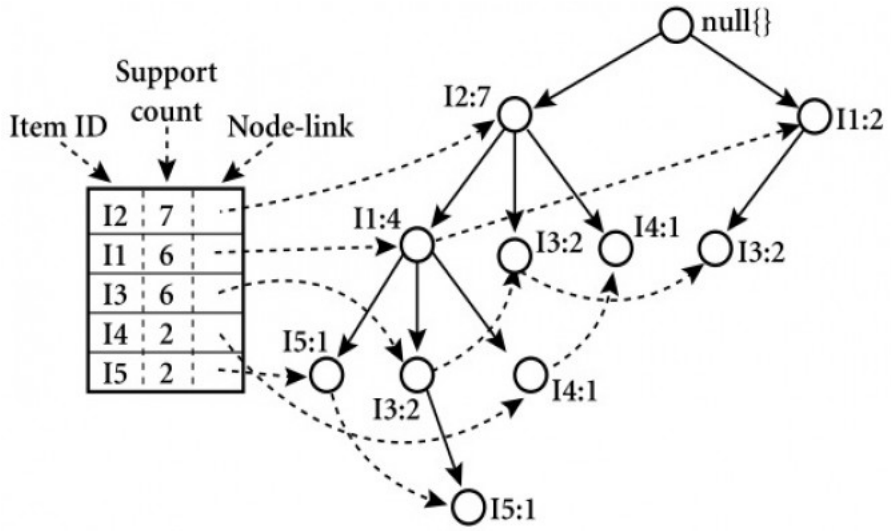

Figura 2: Intuición gráfica FP-Tree [4]

En este caso, se ha optado por Apriori por dos razones principales. La primera debido a que tiene un uso más extendido, conllevando así un desarrollo mucho más eficiente y depurado en las librerías de los distintos lenguajes de programación. La segunda razón y que se relaciona con la eficiencia y el coste de computación ya que al necesitar encontrar solamente relaciones 1:1 es decir solo un par de valores únicos donde uno sería el antecedente y otro el consecuente, el coste es significativamente menor. Para una comparación más detallada entre ambos véase Alfaro y Solano (2017) [4]

6. Ejemplo práctico algoritmo

A continuación se presenta un ejemplo sencillo para entender de una manera más concisa la forma de trabajar del algoritmo y además enfatizar en los conceptos presentados anteriormente como las distintas medidas de la sección de conceptos.

Tenemos una registro de compras donde cada ID de compra pertenece a una compra distinta y en cada una de ellas se encuentra una selección de items elegidos por el comprador.

Compras	
ID	**Productos**
1	Portátil, Alfombrilla, Ratón
2	Portátil, Ratón
3	Tablet, Smartphone
4	Portátil, Tablet
5	Portátil, Smartphone, Ratón, Alfombrilla

Si creamos una matriz que represente el conjunto de transacciones y a su vez los articulos en cada transacción obtenemos la siguiente matriz:

- Productos de compra :

  i_1 = portátil, i_2 = alfombrilla, i_3 = ratón, i_4 = Tablet, i_5 = Smartphone

  Transacciones:

  $T_1 = \{i_1, i_2, i_3\}$

  $T_2 = \{i_1, i_3\}$

  $T_3 = \{i_4, i_5\}$

  $T_4 = \{i_1, i_4\}$

  $T_5 = \{i_1, i_2, i_3, i_5\}$

Artículos / Transacciones	i_1	i_2	i_3	i_4	i_5
T_1	1	1	1	0	0
T_2	1	0	1	0	0
T_3	0	0	0	1	1
T_4	1	0	0	1	0
T_5	1	1	1	0	1

La forma de generar las reglas de asociación consta de dos pasos:

- Generación de combinaciones frecuentes: Cuyo objetivo es encontrar aquellos conjuntos que sean frecuentes en la base de datos. Para determinar la frecuencia se establece cierto umbral con el cual determinaremos con que frecuencia se debe dar un conjunto de items para considerarlo una combinación frecuente, un umbral que anteriormente hemos nombrado como soporte.
- Generación de reglas: A partir de los conjuntos frecuentes no encargamos de generar las reglas, para ello nos apoyaremos en las otras dos medidas descritas anteriormente: confianza y lift.

El primer paso es generar las combinaciones frecuentes,para ello por cada item diferente en el conjunto de transacciones se cuenta la frecuencia con la que aparece y se le asigna un soporte que no es más que esa frecuencia sobre el total de transacciones

Artículo	**Frecuencia**	**Soporte**
Portátil	4	0.8
Alfombrilla	2	0.4
Ratón	3	0.6
Tablet	2	0.4
Smartphone	2	0.4

El siguiente paso una vez determinado el conjunto de artículos será obtener los posibles candidatos, para ello, por cada artículo que aparece en el conjunto de transacciones se revisa las veces que aparece con los otros artículos, obteniendose así una frecuencia de la regla.

A partir de estas dos tablas ya se pueden obtener las distintas medidas que se pueden usar para modificar el coste computacional del algoritmo ya que por ejemplo si establecemos un soporte mínimo para los artículos eliminariamos posibles candidatos y reduciríamos las posibles combinaciones.

Conjuntos	**Frecuencia**	**Soporte**	**confianza**	***lift***
Portátil, Alfombrilla	2	0.4	0.5	1.25
Alfombrilla, Portátil	2	0.4	1	1.25
Portátil, Ratón	3	0.6	0.75	1.25
Ratón, Portátil	3	0.6	1	1.25
Portátil, Tablet	1	0.2	0.25	0.625
Tablet, Portatil	1	0.2	0.25	0.625
Portátil, Smartphone	1	0.2	0.25	0.625
Smartphone, Portátil	1	0.2	0.5	0.625
Alfombrilla, Ratón	2	0.4	1	1.666
Ratón, Alfombrilla	2	0.4	0.333	0.8333
Alfombrilla, Tabet	0	0	0	0
Tabet, Alfombrilla	0	0	0	0
Alfombrilla, Smartphone	1	0.2	0.5	1.25
Smartphone, Alfombrilla	1	0.2	0.5	1.25
Ratón, Tabet	0	0	0	0
Tablet, Ratón	0	0	0	0
Ratón, Smartphone	1	0.2	0.333	0.83333
Smartphone, Ratón	1	0.2	0.5	0.83333
Tablet, Smartphone	1	0.2	0.5	1.25
Smartphone, Tablet	1	0.2	0.5	1.25

Como se puede ver, el algoritmo comprueba uno por uno las posibles combinaciones que se pueden obtener del conjunto de transacciones en ambas direcciones, esto puede generar diferentes problemas a la hora de tener en cuenta el resultado del mismo:

- Primeramente el coste computacional que tiene añadir más artículos al conjunto de transacciones ya que el número de reglas va creciendo conforme la regla de cálculo combinatorio donde se tiene en cuenta el orden de los elementos y no puede seleccionarse un elemento más de una vez: $n^2 - n$ donde n es el número de articulos distintos. Para nuestro caso sería $5^2 - 5$ que obtendría 20 combinaciones diferentes posibles como refleja la tabla inmediatamente anterior. Si representamos gráficamente, vemos que tenemos un incremento cuadrático conforme aumentamos el total de artículos disponibles:

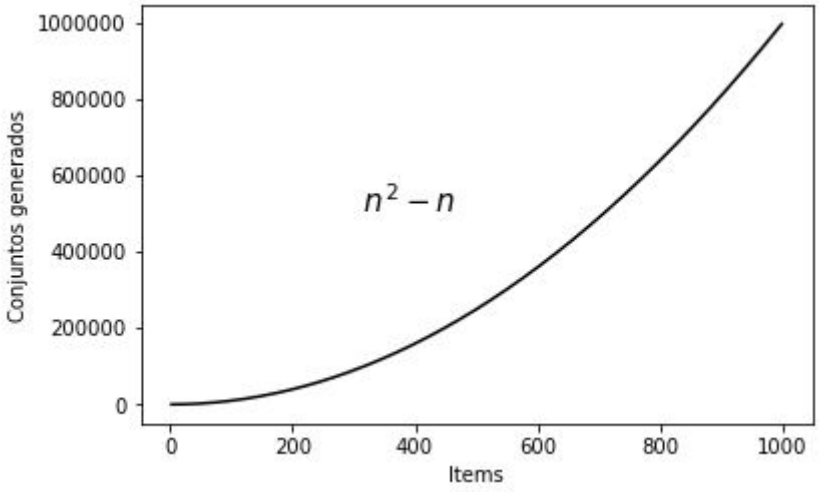

Figura 4: Reglas generadas por Artículos
[9]

Este problema es más de carácter técnico ya que su solución depende en gran medida del equipo disponible para poder ejecutar el algoritmo.

- El segundo problema tiene que ver con las reglas generadas. Como hemos visto en el ejemplo anterior la regla puede apuntar para ambos lados a pesar de que claramente un artículo no sea el causante de otro incluso una vez hayamos descartado relaciones que no consideremos reglas (lift < 1). Es decir, aparece un problema de doble causalidad que no distingue cual puede ser el verdadero artículo principal y el accesorio. Para solucionar este problema, se requiere de creatividad por parte del experto y una posible solución para este caso en concreto se explica en la siguiente sección.

- Y por último y no menos importante también se observa que pueden salir algunas relaciones que poco o nada tienen que ver, como el caso Alfombrilla, Smartphone. Reglas que pueden darse por mera casualidad debido a que existen pocas transacciones de un objeto y las pocas que se tienen se han viciado debido alguna característica externa como en este caso que por ejemplo un cliente compró un teléfono móvil y aprovecho para comprar una alfombrilla para un ratón que ya tenía en casa. Para este caso también se ha propuesto una posible solución.

Conjuntos	**Frecuencia**	**Soporte**	**confianza**	***lift***
Portátil, Alfombrilla	2	0.4	0.5	1.25
Alfombrilla, Portátil	2	0.4	1	1.25
Portátil, Ratón	3	0.6	0.75	1.25
Ratón, Portátil	3	0.6	1	1.25
Alfombrilla, Smartphone	1	0.2	0.5	1.25
Smartphone, Alfombrilla	1	0.2	0.5	1.25
Tablet, Smartphone	1	0.2	0.5	1.25
Smartphone, Tablet	1	0.2	0.5	1.25

De este análisis, si se eliminan todas aquellas reglas donde el *lift* sea menor que 1, nos queda:

- $\{\text{Portátil}\} \Rightarrow \{\text{Alfombrilla}\}$
- $\{\text{Alfombrilla}\} \Rightarrow \{\text{Portátil}\}$
- $\{\text{Portátil}\} \Rightarrow \{\text{Ratón}\}$
- $\{\text{Ratón}\} \Rightarrow \{\text{Portátil}\}$
- $\{\text{Alfombrilla}\} \Rightarrow \{\text{Smartphone}\}$
- $\{\text{Smartphone}\} \Rightarrow \{\text{Alfombrilla}\}$
- $\{\text{Tablet}\} \Rightarrow \{\text{Smartphone}\}$
- $\{\text{Smartphone}\} \Rightarrow \{\text{Tablet}\}$

7. Aplicación del algoritmo

Este apartado se centrará en explicar el desarrollo y las conclusiones obtenidas, más concretamente los problemas que se han tenido que abordar para que el resultado del algoritmo satisfaga en la mayor medida posible las peticiones de los diferentes departamentos con los que se ha trabajado. En primer lugar hay que remarcar la diferencia para cada algoritmo desarrollado ya que para la solución aplicada al upselling apenas ha surgido contrariedad alguna, no siendo de igual manera para el algoritmo de venta cruzada donde se ha requerido de un preprocesado y un postprocesado de datos además de aplicar las reglas de asociaciones lo cual se explica a continuación detalladamente.

7.1. Preprocesado

Con el fin de mejorar la eficiencia del algoritmo se tienen que establecer ciertos criterios a la hora de seleccionar la información a analizar ya que cuanto más grande sea ésta, mayor será el coste computacional y más recursos se deberán destinar para la ejecución del código pudiendo generar problemas de concurrencia. El script que contiene el código se debe ejecutar diariamente descargando los diferentes datos de la base de datos de la empresa (histórico de compras, artículos, rastrillo), ejecutar los cálculos necesarios y actualizar una tabla en la base de datos donde se almacena el resultado final. Para aligerar el tiempo de respuesta y evitar sobrecarga de información se filtran los datos para solucionar los distintos problemas que se han comentado en la seccion anterior. Podemos dividir la ejecución del algoritmo en tres partes principales:

1. Descartar todas las líneas de las compras que no sean artículos (gastos de envio, canon digital, gastos de embalaje, de montaje...).

2. Eliminar las compras monolínea ya que no aportan información alguna al algoritmo.

3. Limitar las compras que queremos analizar ya que una compra con un número excesivo de líneas corre el peligro de haber sido realizada por un distribuidor que pide a discreción de distintos clientes propios viciando así los resultados del algoritmo.

Quedando una conjunto de datos de la siguiente forma[1]

ID Compra	**ID artículo**
1	150
1	240
1	370
2	187
2	242
2	398
2	547
2	1200
3	47
3	23
⋮	⋮
T_n	89
T_n	23
T_n	894
T_n	5020

Una vez tenemos nuestra tabla de transacciones la usamos de input para el algoritmo y esté nos muestra un resultado parecido al de la tabla construida en el ejemplo práctico. A ella le añadimos la información de la base de datos como la familia a la que pertenece, su precio y su stock calculado como el stock en almacen, restandole los pedidos para ese articulo y sumándole los pendientes de entrada (compras) y exigiendo que solo mantenga aquellas reglas con un lift superior a 1.1[2].

[1] Este resultado pertenece a la parte de código llamada preprocesado adjunta en el anexo.

[2] Este resultado pertenece a la parte de código llamada Algoritmo Apriori en python adjunta en el anexo.

Antecedente	Familia	Precio	Consecuente	Familia	Precio	Stock
150	Portátiles	300 €	272	Ratones	8 €	10
240	Smartphones	300 €	598	Carcasas	15 €	12
370	Televisores	700 €	1580	Soportes Tv	40 €	5
187	Smartphones	200 €	871	Carcasas	14 €	50
242	Sobremesa	800 €	68	Monitores	120 €	17
398	Consolas	400 €	10	Juegos	60 €	2
547	Smartphones	100 €	451	Carcasas	5 €	25
1200	Portatiles	200 €	314	Alfombrillas	9 €	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮
47	Televisores	200 €	847	Soportes TV	70 €	20

7.2. Postprocesado

Como se explico anteriormente, el resultado de aplicar Apriori no sería del todo inteligente si se dejará tal y como está ya que como se ha visto existen diversos problemas a la hora de establecer relaciones entre artículos que el experto debe descubrir, tratar y reparar en la medida de lo posible y de una manera provechosa para su objetivo final. En esta sección, se intenta mejorar las reglas de asociación para eliminar todas las asosiaciones indeseables. A continuación, se relata como se han intentado corregir los problemas descritos en la sección anterior[3].

1. Asociaciones con familias incompatibles. Este problema se da muy a menudo debido a compras conjuntas que compran 2 artículos muy diferentes o que uno no es accesorio de otro y se registran pocas transacciones de los mismos, por ejemplo si solo se han comprado una vez como en el ejemplo de la alfombrilla y el smartphone. Para evitar este tipo de asociaciones por cada familia, el algoritmo agrupa y cuenta por familias los consecuentes con la familia principal y limitando las reglas a aquellas cuyo consecuente pertenece a una de las familias que más se venden con este artículo.

   Por ejemplo, si se estudia la familia de portátiles se obtendría una tabla similar a la siguiente, donde se agrupan las reglas y se distingue por familia del consecuente obteniendo de esta manera los casos más comunes.

[3] Este resultado pertenece a la parte de código llamada postprocesado adjunta en el anexo.

Familia	Total
Ratones	1 272
Alfombrillas	998
Disco duro externo	897
Impresoras	342
Conectividad	68
Consolas	10
Televisores	8
Smartphones	6
⋮	⋮
Sobremesa	1

Por lo tanto, en el conjunto de reglas generadas para portátiles, solo se mantendrían aquellos pares que tuvieran de consecuente Ratones, Alfombrillas, Disco duro externo, Impresoras o conectividad y se eliminarían todas las demás. Aplicando un análisis más exhaustivo se está utilizando el comportamiento general de los clientes para poder depurar los datos, es decir, son los mismos clientes los que con sus compras dictaminan cuales son las familias que más se compran junto a otra familia y está información es utilizada para solventar el problema de asociaciones incompatibles.

2. Bien por diseño, bien por política empresarial se exige que hayan exactamente 4 accesorios por cada artículo principal por motivos de diseño de la página web. Esto es un problema debido a que existen artículos donde no se consigue completar el conjunto de asociaciones o peor aún, no obtenemos ninguna quedando un conjunto vacío. En este caso se ha optado por asociar esos artículos realizando un top de accesorios concreto de esa familia y esa marca y asociarlos hasta obtener el máximo de los 4 accesorios exigido. Por ejemplo, tenemos un nuevo portátil que ha entrado en stock de cierta marca. Este portatil no tiene ningún registro de compra y evidentemente el algoritmo no encontrará ninguna asociación posible. Por lo tanto, para estos casos particulares se detecta la familia a la que pertenece este articulo (en este caso en concreto un portatil), se identifica la marca del mismo y se obtiene una lista con los accesorios ordenados de más vendidos a menos vendidos de esa familia y esa marca en particular. Una vez realizados los pasos, solo queda establecer cuantos artículos cogemos de la lista que en este caso es 4, pero en el caso general es 4-n (donde n es el número de accesorios encontrados para ese caso en particular). Además de esto se le añade otra funcionalidad de filtro por precio, en el cual un articulo no puede ser accesorio si supera un cierto umbral de precio variable (dependiendo del precio del principal).

3. Un último problema es el que origina los artículos rastrillo. Este problema surge cuando un artículo que estaba expuesto para la vista al público sale a la venta. Este se da de alta en la base de datos con un id de artículo diferente al producto original lo cual hace que ese artículo no posea ningún histórico de compras y por lo tanto nunca podrá crearse una asociación. No obstante en la base de datos se registra en otro campo el id del articulo original y este se usa para poder enlazar al artículo rastrillo los mismo accesorios que tiene el artículo original.

8. Aplicación

Como se ha descrito anteriormente, todo este desarrollo no sería valido sin otros pasos complementarios para aprovechar todo el conocimiento revelado por el algoritmo. Para ello, se ha construido lo que en el mundo de la informatica se conoce como una ETL .

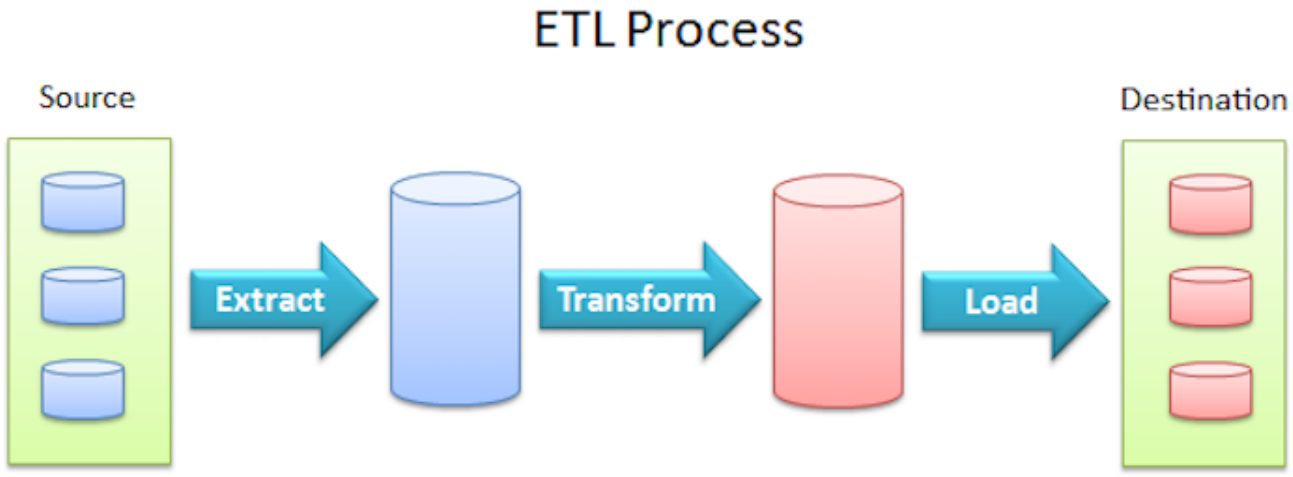

Figura 5: ejemplo general de ETL

Para este caso en particular, el script escrito en lenguaje Python se almacena en un servidor, al mismo se programa para que se ejecute diariamente con las siguientes instrucciones:

1. Primeramente con el mismo motor de Python y la libreria SQLAlchemy se lanza una consulta a la base de datos en lenguaje SQL donde cargamos en distintos dataframes la información necesaria para procesarla.

2. Seguidamente los datos son procesados como se ha explicado en secciones anteriores para crear una tabla donde a cada item le corresponde su accesorio y su familia además de otra columna donde se indica la familia del artículo accesorio. Como se podrá observar cada artículo aparece repetido 4 veces, y como se ha mencionado anteriormente es debido a que solo se necesitan mostrar 4 artículos por id articulo principal

id articulo	id accesorio	id familia
1	12	1
1	14	1
1	13	2
1	22	3
2	44	8
2	45	8
2	82	4
2	61	17
⋮	⋮	⋮
k	111	4

3. Una vez la tabla ha sido calculada, usando otra vez la librería que permite utilizar lenguaje SQL, se actualiza la tabla. Este, sería el punto y final del script de cross-selling y la tabla creada quedaría a disposición del departamento encargado de llevar la pagina web donde se comercializan los productos e insertar en la web una zona donde se puedan visualizar conjuntamente con cada artículo los 4 accesorios que se recogen en la tabla

9. Trabajos futuros

Aunque en un primer momento los resultados parecen positivos, hay que ser conscientes de que es una primera versión que tiene mucho margen de recorrido para probar distintas funcionalidades. En este sentido se debe trabajar junto al departamento de diseño web para probar los distintos tipos de vistas y diseñar una configuración que capte la atención del usuario. Además de ello, el equipo técnico también debe trabajar en la mejora continua del algoritmo sobre la base del mismo y que podrían ser:

- Desplegables de accesorios. Actualmente la presentación de los resultados está limitado a un máximo de 4 accesorios por artículo, no obstante, debajo de la descripción del artículo accesorio se puede añadir un botón que abra un desplegable con la opción de cambiar un accesorio por otro. Por ejemplo si estamos visitando un Smartphone aparecen de accesorios una funda, una carcasa y una tarjeta de almacenamiento. Quizás el usuario prefiera otro tipo de carcasa o una tarjeta de almacenamiento con más capacidad. En vez de recorrer el catalogo y añadirle un trabajo extra al usuario, él mismo podría configurar sus accesorios con el diseño o características que más satisfagan sus preferencias sin la necesidad de moverse de la página del artículo. En este sentido solamente habría que quitar la limitación de 4 accesorios y trabajar en conjunto con diseño web para que en cada slot de accesorio aparecieran desplegados al pulsar el botón artículos de la misma familia.

- Recomendaciones por usuarios. Sería posible dividir a los usuarios según ciertas características como el gasto medio y la frecuencia de compra mediante técnicas de aprendizaje automático como análisis cluster, dividir a los clientes en diferentes grupos y por cada grupo ofrecer una combinación diferente de accesorios. Claro está que para esta personalización el usuario deberá iniciar sesión para poder identificarlo. Incluso se podría utilizar los datos de navegación del cliente para poder ofrecer recomendaciones personalizadas utilizando las técnicas ya mencionadas y la integración de software específico de Big data para el análisis de datos en tiempo real y la creación de una api que comunicara las llamadas a la página web con los datos almacenados.

10. Conclusiones

- Existe una gran variedad de técnicas para dar solución a este tipo de problemas incluso existiendo empresas dedicadas exclusivamente a ello.
- El resultado en bruto del algoritmo no garantiza el exito del mismo ya que debe de existir un postprocesado de datos que deseche en la medida de lo posible resultados poco lógicos.
- Las rutinas de ejecución deben de estar bien depuradas y automatizadas para intentar evitar cualquier tipo de fallo y en última instancia si se produjese tener algún tipo de mecanismo de alerta.
- No existe una solución única, se debe trabajar continuamente en diferentes alternativas y estudiar cual puede ser la mejor configuración para nuestro caso en concreto.
- Hay que escoger una métrica que mida adecuadamente el impacto del algoritmo tanto para monitorizar su rendimiento como para medir posibles cambios en el mismo.

Bibliografía

[1] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[2] Fernando Berzal, Ignacio Blanco, Mar Vila, et al. Measuring the accuracy and interest of association rules: A new framework. *Intelligent Data Analysis*, 6(3):221–235, 2002.

[3] Sergey Brin, Rajeev Motwani, Jeffrey D Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In *ACM SIGMOD Record*, volume 26, pages 255–264. ACM, 1997.

[4] Jimmy Solano Felipe Alfaro. Apriori vs fp-growth for frequent item set mining, 2017.

[5] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM sigmod record*, volume 29, pages 1–12. ACM, 2000.

[6] Christian Hidber. *Online association rule mining*, volume 28. ACM, 1999.

[7] Robert Hilderman and Howard Hamilton. Evaluation of interestingness measures for ranking discovered knowledge. *Advances in Knowledge Discovery and Data Mining*, pages 247–259, 2001.

[8] Ying Liu, Jayaprakash Pisharath, Wei-keng Liao, Gokhan Memik, Alok Choudhary, and Pradeep Dubey. Performance evaluation and characterization of scalable data mining algorithms. In *16th IASTED international conference on parallel and distributed computing and systems (PDCS). MIT, Cambridge*, pages 620–625, 2004.

[9] Mittal Mandeep. Efficient ordering policy for imperfect quality items using association rule mining, 2014.

[10] Jong Soo Park, Philip S Yu, and Ming-Syan Chen. Mining association rules with adjustable accuracy. In *Proceedings of the sixth international conference on Information and knowledge management*, pages 151–160. ACM, 1997.

[11] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right objective measure for association analysis. *Information Systems*, 29(4):293–313, 2004.

[12] Hannu Toivonen et al. Sampling large databases for association rules. In *VLDB*, volume 96, pages 134–145, 1996.

11. Anexo

11.1. Código

Listing 1: Importamos librerías

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


#Cargamos la libreria
import pandas as pd
import matplotlib as mp
import numpy as np
import MySQLdb as sql
import datetime
from sqlalchemy import create_engine
from datetime import timedelta
import rpy2.robjects as robjects
from rpy2.robjects.packages import importr
from rpy2.robjects import pandas2ri
ts=robjects.r('ts')
arules = importr('arules')
```

Listing 2: Preprocesado

```
#traemos los datos de la BBDD
query1 = """ SELECT fecha, id_compras_datos, ca.id_producto
FROM compras """



#articulos
query3= """ SELECT id_articulo, familia , marca, precio, stock
FROM articulos 3"""


#reacondicionados y refurbished
query4= """Select id_articulo, id_articulo_refurbished
FROM articulos_rastrillo """

db_connection = sql.connect(host='#####', port=####, user=######,
                passwd="####",db="#####" )
```

```
db_cursor = db_connection.cursor()

articulos = pd.read_sql(query3, db_connection)
rastrillo = pd.read_sql(query4, db_connection)
df = pd.read_sql(query1, db_connection, index_col = 'id_compras_articulo',
    parse_dates=['fecha'])


db_connection.close()


#ordenamos las compras por id y fecha
df = df.sort_values(['fecha', 'id_compras_datos'], ascending=[0, 0])

#fijamos el periodo de estudio
dias= 1000
fechamin = df['fecha'].max() - timedelta(days=dias)
df = df[df['fecha'] > fechamin]

#quitamos productos que no nos interesan donde desechar es un list object con los id
    #productos
df = df[~df['id_producto'].isin(desechar)] #quitamos del dataframe todas las compras

#quitamos compras monolinea
keep = df.groupby(['id_compras_datos'])['id_producto'].count().reset_index()
keep = keep[(keep["id_producto"] > 1) ]
keep = list(keep["id_compras_datos"])
df = df[df['id_compras_datos'].isin(keep)]

#preparamos los datos de entrada del algoritmo
df2 = df[['id_compras_datos', 'id_producto']]
df2 = df2.values
df2 = pd.DataFrame(df2)
df2.columns = ['InvoiceNo', 'id_producto' ]

contador = []
comparador = df2['InvoiceNo'][0]
n=1
for i in df2['InvoiceNo']:
    if comparador == i:
        contador.append(n)

    else:
        n=n+1
        comparador = i
```

```
        nn=n
        contador.append(nn)

df2['id'] = contador

cestap = df2[['id_producto', 'id']]
```

Listing 3: Algoritmo Apriori en Python

```
pandas2ri.activate()

Cesta = pandas2ri.py2ri(cestap)


rstring= """  function(prod,id,supp,conf)
{
  tr <- as(split(prod,id), "transactions")
  rules <- apriori(tr, parameter = list(supp=supp,conf=conf,minlen=2,maxlen=2), control
      = list(sort=-2,memopt=TRUE))
  return(rules)
}    """


CreaRules=robjects.r(rstring)

rstring= """ function(first.rules)
{
  rulesDF <- as(first.rules, "data.frame")
  rules <- regmatches(rulesDF$rules,gregexpr("{([^}]*)}",rulesDF$rules, perl=T))
  rules <- lapply(rules, function(x) gsub("[{}]","",x))
  mrules <- lapply(rules, function(x) cbind(left=strsplit(x[1],",")[[1]], right = x[2]))
  NavRules <- cbind(rulesDF[rep(1:nrow(rulesDF), sapply(mrules, nrow)),],do.call(rbind,
      mrules))
  NavRules <- NavRules[,c(5,6,2,3,4)]
  names(NavRules) <- c("antecedants", "consequents", "support", "confidence", "lift")
  return(NavRules)
} """

FromRulesToDataFrame =robjects.r(rstring)

rules = CreaRules(Cesta[0],Cesta[1],0.00001,0.01)
R = FromRulesToDataFrame(rules)

R_df = pandas2ri.ri2py(R)

R_df['antecedants'] = pd.to_numeric(R_df['antecedants'])
```

```
R_df['consequents'] = pd.to_numeric(R_df['consequents'])
R_df = R_df.sort_values(by = ['antecedants','lift'], ascending = [False, False])
R_df = R_df[R_df['lift'] > 1.1]
```

Listing 4: Funciones

```
def cont_id(dataset, id_cont):
    """ Funcion que asigna a un dataset un id incremental por cada numero que se repita
        en la columna que le indiquemos cuando ese numero cambio el id se reinicia a 1"""
    dataset = dataset.reset_index(drop=True)
    contador = []
    comparador = dataset[id_cont][0]
    n = 1

    for i in dataset[id_cont]:
        if comparador == i:
            contador.append(n)
            n = n + 1
        else:
            n = 2
            comparador = i
            nn = 1
            contador.append(nn)

    dataset['contador'] = contador
    return dataset


def remplazador(x, y, minimo_accesorios):
    """Esta funcion necesita dos dataframes. Uno con las reglas de asociacion y otro con
        el total de productos. En el dataframe de reglas de asociacion rellena los
        accesorios en orden de mas vendidos si son menor que el numero que elijamos, si
        no esta lo introduce en el dataset y le aniade los accesorios mas vendidos. """

    global relleno, final

    dataframe = x
    acc = x[['antecedants', 'contador']].groupby('antecedants').count()
    final = x[['antecedants', 'consequents', 'contador']].reset_index(drop=True)

    topaccg = dataframe["consequents"].value_counts()
    topaccg = list(topaccg.index.values)

    relleno = pd.DataFrame()
```

```
for i in range(0, len(acc)):
    accesorios = acc.iloc[i].values
    id_marca = list(dataframe['id_marca_x'][dataframe['antecedants'] ==
        acc.index[i]].unique())
    topacc = dataframe["consequents"][(dataframe['id_marca_x'] ==
        id_marca)].value_counts()
    topacc = list(topacc.index.values)
    topacc = topacc

    if accesorios < minimo_accesorios:

        topacc = topacc + topaccg

        antecedente = [acc.index[i]] * len(topacc)
        contador = list(range(accesorios, len(topacc) + accesorios))

        data2 = pd.DataFrame({'antecedants': antecedente, # creamos un dataframe con
            esa lista
                              'consequents': topacc,
                              'contador': contador})
        relleno = relleno.append(data2)



    else:
        continue

for i in range(0, len(y)):

    if y[i] not in acc.index.values:

        id_articulo = y[i]

        id_marca = articulos['id_marca'][articulos['id_producto'] ==
            id_articulo].values
        id_familia = articulos['id_familia'][articulos['id_producto'] ==
            id_articulo].values

        if len(id_marca) > 0:
            id_marca = int(id_marca)
            id_familia = int(id_familia)

            dataframe = x[x['id_familia_x'] == id_familia]

            topaccg = dataframe["consequents"][(dataframe['id_familia_x'] ==
                id_familia)].value_counts() # cogemos el top global de accesorios
```

```
                topaccg = list(topaccg.index.values)

                topacc = dataframe["consequents"][(dataframe['id_marca_x'] ==
                    id_marca)].value_counts() # cogemos los accesorios que mas se venden
                    con esa marca
                topacc = list(topacc.index.values)
                topacc = topacc

                topacc = topacc + topaccg

                antecedente = [y[i]] * len(topacc)
                contador = list(range(1, len(topacc) + 1))

                data2 = pd.DataFrame({'antecedants': antecedente,
                                      'consequents': topacc,
                                      'contador': contador})
                relleno = relleno.append(data2)
            else:
                continue

        vertical_stack = pd.concat([final, relleno], axis=0)
        vertical_stack = vertical_stack.sort_values(['antecedants', 'contador'],
            ascending=[True, True]).reset_index(drop=True)
        vertical_stack = vertical_stack.astype(int)

        vertical_stack = pd.merge( vertical_stack, articulos, left_on= 'antecedants',
            right_on='id_producto', how = 'left' )
        vertical_stack = pd.merge( vertical_stack, articulos, left_on= 'consequents',
            right_on='id_producto', how = 'left' )
        vertical_stack = vertical_stack.dropna(axis=0, how='any')


        return vertical_stack


    def relleno_ref(x, y):
        """Pasa a la funcion el dataframe con los articulos en en rastrillo y los enlaza con
            su articulo orginal
        x=rastrillo
        y= dataframe con las asociaciones
        """

        check = list(y['antecedants'].unique())

        relleno = pd.DataFrame()
```

```
    for i in range(0, len(x)): # coge el portatil del rastrillo
        original = x['id_articulo'].loc[i] # id del articulo de origen
        reacondicionado = x['id_articulo_refurbished'].loc[i] # id del articulo del
            rastrillo
        if original in check: # si el id esta en nuestro dataframe de portatiles aniadele
            al diccionario el
            # id articulo reaciondcionado con sus accesorios originales

            captura = y[y['antecedants'] == original]



            antecedente = [reacondicionado] * len(captura['antecedants'])
            consecuentes = list(captura['consequents'])
            contador = list(captura['contador'])
            familia = list(captura['id_familia_y'])

            data2 = pd.DataFrame({'antecedants': antecedente, # creamos un dataframe con
                esa lista
                                  'consequents': consecuentes,
                                  'contador': contador,
                                 'id_familia_y': familia })

            relleno = relleno.append(data2)



    vertical_stack = pd.concat([y, relleno], ignore_index=True)


    return vertical_stack
```

Listing 5: Postprocesado

```
df3 = pd.merge(R_df, articulos, left_on= 'antecedants', right_on='id_producto', how =
    'left' )
df4 = pd.merge(df3, articulos, left_on= 'consequents', right_on='id_producto', how =
    'left' )




familias = df4['id_familia_x'].unique().tolist()

#completamos los accesorios
```

```
Tabla = pd.DataFrame()
for i in familias:
    df5 = df4[df4['id_familia_x'] == i]
    topfa = df5['id_familia_y'].value_counts()
    keep = topfa.index.tolist()
    keep = keep[0:4] #limita el numero de top familias con el que se puede emparejar
    df5 = df5[df5['id_familia_y'].isin(keep)]
    df5 = cont_id(df5,'antecedants')
    rellename = articulos['id_producto'][articulos['id_familia'].isin([i])].values
    df6 = remplazador(df5, rellename)
    Tabla = Tabla.append(df6, ignore_index=True)

aniadimos los refurbished
df6 = relleno_ref(rastrillo, Final)

#limitamos por el numero de accesorios que queremos
Final = Final[Final['contador'] <= 4]
```

Listing 6: Script upselling

```
articulos = df['id_articulo'].values
articulo_marca = df[['id_articulo', 'id_marca']]

relleno = pd.DataFrame()
for i in range(0 ,len(articulos)):
   id_articulo = articulos[i]

   familia = int(df[df['id_articulo'] == id_articulo ]['id_familia'])
   marca = int(df[df['id_articulo'] == id_articulo ]['id_marca'])
   relevancia = float(df[df['id_articulo'] == id_articulo ]['relevancia'])
   precio = float(df[df['id_articulo'] == id_articulo ]['precio'])

   up_marca = list(df[(df['id_familia'] == familia) & (df['id_articulo'] !=
       id_articulo) & (df['precio'] > precio) & (df['precio'] <= precio*limite_superior)
       &  (df['id_marca'] == marca) & (df['stock'] > 0)].sort_values(['relevancia'],
       ascending=[False])['id_articulo'][0:5].values)

   up = list(df[(df['id_familia'] == familia) & (df['id_articulo'] != id_articulo) &
       (df['precio'] > precio) & (df['precio'] <= precio*limite_superior) &
       (df['id_marca'] != marca) & (df['stock'] > 0)].sort_values(['relevancia'],
       ascending=[False])['id_articulo'][0:5].values)

   total = up_marca + up

   antecedente = [id_articulo] * len(total)
```

```
contador = list(range(1, len(total) + 1))
data2 = pd.DataFrame(
{'antecedants': antecedente,
'consequents': total,
'contador': contador}
)
relleno = relleno.append(data2)
```

Listing 7: Carga a la base de datos

```
engine = create_engine("mysql+mysqldb://user:"+'password'+"@host")
engine.execute(' DELETE FROM nombre_de_la_tabla WHERE contador >= 0 ')
Tabla.to_sql('nombre_de_la_tabla' ,con=engine, if_exists='append')
```