REPASO DE CONCEPTOS PROBABILÍSTICOS Y TÉCNICAS NO SUPERVISADAS

ÍNDICE:

[**INTRODUCCIÓN Y CONCEPTOS PROBABILÍSTICOS BÁSICOS** 3](#_Toc166525741)

[Principios de la Teoría de la Probabilidad 3](#_Toc166525742)

[El Teorema de Bayes I: Conceptos básicos 8](#_Toc166525743)

[El Teorema de Bayes II 11](#_Toc166525744)

[**1.- MODELOS DE ASOCIACIÓN: BASKET MARKET ANALYSIS** 15](#_Toc166525745)

[1.1 Conceptos Básicos del modelo 16](#_Toc166525746)

[1.2 Descripción del algoritmo 18](#_Toc166525747)

[1.3 El modelo *BMA*. Aplicaciones y ejemplos. 19](#_Toc166525748)

[**2.- INTRODUCCIÓN A LAS REDES BAYESIANAS** 22](#_Toc166525749)

[2.1 Conceptos básicos del modelo 23](#_Toc166525750)

[2.2 Construcción de Redes Bayesianas 27](#_Toc166525751)

[2.3 Inferencia estadística en redes bayesianas 31](#_Toc166525752)

[Inferencia exacta en cadenas 32](#_Toc166525753)

[Inferencia con Evidencia Incierta 34](#_Toc166525754)

[**3.- ENTRENAMIENTO NO SUPERVISADO I: INTRODUCCIÓN AL ANÁLISIS CLUSTER** 36](#_Toc166525755)

[3.1. Preliminanes a las técnicas de clustering 36](#_Toc166525756)

[3.2. Clustering de individuos 39](#_Toc166525757)

[Algoritmo k-means 40](#_Toc166525758)

[Algoritmo de Ward 44](#_Toc166525759)

[3.3. Clustering de variables 45](#_Toc166525760)

[**4.- ENTRENAMIENTO NO SUPERVISADO II: MÁS SOBRE ANÁLISIS CLUSTER** 47](#_Toc166525761)

[4.1. El Algoritmo t-SNE 47](#_Toc166525762)

[4.2. El Algoritmo UMAP: Evolución del t-SNE 50](#_Toc166525763)

[4.3. El Algoritmo DBSCAN 51](#_Toc166525764)

[4.4. El Algoritmo HDBSCAN 54](#_Toc166525765)

[**5.- DETECCIÓN DE ANOMALÍAS** 54](#_Toc166525766)

[**BIBLIOGRAFÍA** 58](#_Toc166525767)

# **INTRODUCCIÓN Y CONCEPTOS PROBABILÍSTICOS BÁSICOS**

Se comienza esta primera parte del curso con las técnicas de corte no supervisado, se hace por tanto un recorrido de los principales algoritmos que caben ser considerados en áreas de analítica avanzada, mencionándose por un lado los más representativos: *BMA* o *k-means* pero sin olvidar a otros que tienen cierta novedad y que aunque tienen mayor complejidad, pueden dar lugar a resultados que pueden ayudar a entender mejor el comportamiento de ciertos patrones de infomración como es el caso de *UMAP* o los de tipo *DBSCAN*.

Así pues, dada la constitución de algunos de los algoritmos que aquí se mencionan y por aplicaciones posteriores, se cree conveniente realizar un breve repaso a algunos conceptos básicos de la *teoría de la probabilidad*.

Sin pretender ser exhaustivos se toman sólo algunos elementos básidos de la mecionada teoría que por otro lado deberían ser de sobra conocidos por los alumnos que se acercan a realizar una formación específica en materia de *Data Sciencie* en general, por tanto esta sesión asume alumnos o lectores de diversa procedencia formativa y aquí se aclaran una serie de cuestiones comunes a la vez que mínimas de las cuáles partir a nivel técnico – matemático recomendándose la lectura de cualquier libro o manual de probabilidad que sin duda complementará lo aquí desarrollado.

## Principios de la Teoría de la Probabilidad

La *teoría de la probabilidad* comienza con el concepto de *sigma-álgebra*, aunque el nombre pueda resultar un tanto exótico, de lo que se trata es de describir un entorno abstracto, un universo de individuos totalmente general donde “todo cabe” y unas reglas de juego para las operaciones a realizar entre los individuos de dicho universo.

Antes de entrar en la definición de *sigma-álgebra* conviene tener en cuenta que los individuos pueden agruparse en colecciones que se denominan *conjuntos* y por tanto lo que interesa saber 2 cosas:

* ¿Qué ocurre cuando se mezclan distintos *conjuntos*?
* ¿Cómo puede medirse su tamaño respecto al *universo* de partida?

El concepto de *sigma-álgebra* surge para dar solución a la primera cuestión y para garantizar que los *universos* que son entelequias generales sobre un determinado campo de estudio de interés van a tener cierto sentido. Aunque a continuación se dará la definición de lo que es una *sigma-álgebra* en general en los estudios que se realizan en la práctica no se recurre a un nivel tan sumamente básico y se presupone que estas propiedades se cumplen

Definición de *sigma-algebra*:

Una colección *F* de subconjuntos de Ω se denomina σ-álgebra en el par si satisface las siguientes condiciones:

1. *F*
2. Si *F* entonces *F*
3. Si *F* entonces *F*

\_\_\_\_\_

Ejemplo 01: Una *sigma-álgebra* muy real

¿Y qué puede ser una *sigma-álgebra* de las que nos podemos encontrar? Supongamos que estamos en una compañía que en un momento del tiempo desea analizar los clientes que se van a dar de baja. En este caso, el *universo* sería todos los clientes a una determinada fecha que están de alta. Ahora hay que dar una estructura de conjuntos útil y con sentido al objetivo de análisis, en este caso, dada la anterior fecha en concreto se puede considerar a los clientes que tras 6 meses, se han dado de baja de la compañía. En este caso, cabría considerar los siguientes sub-conjuntos:

* El conjunto vacío sin ningún elemento
* El conjunto total con todos los elementos de la población
* Un conjunto con los *individuos* que están de alta durante todo el período
* Un conjunto con los *individuos* que causan baja durante el período

Nótese que estos 4 *conjuntos* cumplen perfectamente las propiedades enunciadas anteriormente

Otro ejemplo que se enuncia con frecuencia en los manuales de probabilidad es el referido a la tirada de un dado de 6 caras igualmente equilibradas. Se define el universo, como el formado por los siguientes elementos:

En este caso se pueden considerar varias familias de sub-conjuntos como las siguientes:

1. Considérese la siguiente familia *F*  Se puede verificar que *F* es σ-álgebra

1. Sin embargo, no sería σ-álgebra la siguiente familia *F*  ya que no se verifica la propiedad ii)

\_\_\_\_\_

Por tanto hay que notar que en los experimentos que se hacen en la realidad por un lado está la población o el *universo* en su totalidad y por otro estarán los *conjuntos* o también denominados *sucesos* que tienen lugar dentro de dicho *universo* y que en puridad deberían ser enumerados todos, *σ-álgebra* nótese que no es más que un catálogo de *sucesos* o *conjuntos* a considerar en un determinado *universo*, aunque en general dicho catálogo no suele ser explicitado claramente en mayoría de las aplicaciones que aquí y en la realidad se consideran.

Por otro lado, la segunda cuestión que se planteó anteriormente tiene que ver con la medida del tamaño relativo de los *sucesos* respecto al *universo* del que proceden y esto es importante, porque nos va a dar una idea de la importancia de dicho *suceso* o *conjunto*, por tanto es necesario un instrumento de medida que es el que se conoce como *probabilidad* con esta medida se va a poder medir ocurrencias a través de los tamaños relativos de los subconjuntos catalogados en la *σ-álgebra*. Por tanto ¿Cómo va a ser posible medir ocurrencias? ¿Cuántas posibles medidas van a existir? ¿Cómo van a a ser dichas medidas de ocurrencias?

Definición de *medida* *de* *probabilidad*:

Una medida de probabilidad *P* sobre el par , donde *F* es una *σ-álgebra* es una función que verifica las siguientes propiedades:

1. Dominio de definición: *P* : *F*
2. *P*
3. *P*
4. Si es una colección de subconjuntos disjuntos de *F*, se verifica que:

Con lo que a la terna se la denominará *espacio de probabilidad*

\_\_\_\_\_

Como se observa, la anterior definición responde a algunas de las cuestiones anteriores, por un lado, se tiene que la medida será siempre positiva y acotada entre 0 y 1, además debe ser aditiva bajo uniones de conjuntos disjuntos aunque el número de estos sea infinito, si los conjuntos tienen intersección nula, entonces la probabilidad de la unión de esos conjuntos es la suma de la probabilidades, lo cual es muy natural pensarlo, así pues hay muchas medidas de probabilidad, tantas como aquellas para las que dado un par *universo* - *σ-álgebra* respeten las anteriores condiciones tal y como se observa en los siguientes ejemplos

Ejemplo 02: Asociando medidas de probabilidad

En el *ejemplo 01* se estaba tratando el caso de una compañía a la que se le había asociado una *σ-álgebra* a un *universo* de clientes formado por el siguiente catálogo de subconjuntos:

* El conjunto vacío sin ningún elemento
* El conjunto total con todos los elementos de la población
* Un conjunto con los *individuos* que están de alta durante todo el período
* Un conjunto con los *individuos* que causan baja durante el período

Con estos datos, es factible definir una medida de probabilidad del siguiente modo haciéndola explícita sobre cada uno de los subconjuntos anteriores:

*P*

*P*

*P*

*P*

Nótese que los conjuntos y son disjuntos y que su unión forman el total por tanto debe pasar en este caso que:

*P* y *P*

Por tanto no cualquier asociación de valores a los subconjuntos de la *σ-álgebra* es factible.

Por otro lado, aquí se ha partido que se conocen los valores probabilísticos de los que se van a dar de baja, pero en general esto sólo será conocido a posteriori y no a priori siendo justamente la búsqueda de este conocimiento a priori lo que toda empresa busca ¿Cuántos clientes se me van a dar de baja el próximo año? Y en otros entornos ¿Cuántos clientes captaré el próximo año? Para responder por tanto a una pregunta del tipo ¿Está creciendo o cayendo mi negocio? Muchos de los modelos que estudiaremos más adelante van a pretender estimar estos valores en sí.

Claramente si se hiciera la siguiente asociación de valores a la *medida de probabilidad*, entonces, *P* no sería *medida de probabilidad*:

*P*

*P*

*P*

*P*

Ya que: *P*016 y como *P* que debería valer 1, se tendría que no se cumpliría la propiedad *iii* enunciada en la definición anterior

\_\_\_\_\_

De la anterior definición, se puede enunciar una serie de propiedades que van a ayudar tanto a entender mejor qué es un *espacio de probabilidad*, como los modelos y aplicaciones que se irán desarrollando más adelante.

Propiedad de la unión de 2 conjuntos: Para 2 *conjuntos* o *sucesos* cualesquiera (con intersección vacía o no) en un determinado universo se verifica lo siguiente:

Nótese que si los *conjuntos* fuesen disjuntos, entonces la intersección es nula y por tanto se estaría en el punto iv de la definición anterior (aunque restringuida sólo para 2 *conjuntos*). Visualmente lo anterior quedaría claro con el siguiente diagrama:

Diagrama, Diagrama de Venn

Descripción generada automáticamente

\_\_\_\_\_

Propiedad del complementario: Dado un *conjunto* cualquiera en un universo, se verifica que:

Es decir, si es el conjunto complementario de , se tiene que ambos conjuntos forman lo que se denomina una *partición del universo* en este caso en 2 partes (justo lo que se había mostrado en los ejemplos 1 y 2 anteriores) por tanto es obvio lo anterior

\_\_\_\_\_

Propiedad de la unión de n *conjuntos*: Finalmente se enuncia una fórmula totalmente general para cuando se quiere unir una colección de *suscesos* de tamaño finito *n*:

Esta propiedad responde al efecto de evitar contar las intersecciones más de una vez y aunque puede pensarse en ella, no va a ser necesario memorizarla porque vaya a surgir recurrentemente en las aplicaciones como sucede con las otras 2, aunque sí se cree conveniente pensar en cómo está construida.

\_\_\_\_\_

Al final, tal y como se observa, una *medida de* *probabilidad* no es más que una medida del tamaño relativo de unos conjuntos (de elementos que pueden ser clientes, transacciones, …) respecto a un *universo* cuya medida es 1, por tanto, si bajo la anterior gráfica se afirma por ejemplo que *P*(*A*) = 0’30, lo que se está diciendo realmente es que el tamaño relativo de *A*, respecto al universo es de un 30% del espacio disponible.

Finalmente se define una *medida de probabilidad* que será muy utilizada para el caso de *universos* finitos que será el objetio de muchas aplicaciones finales

Definición de *probabilidad* en poblaciones finitas:

Dado un universo con *n* *individuos*, la función que asocia a cada individuo un valor de probabilidad 1 / *n* resulta ser una medida de probabilidad que para cualquier *σ-álgebra* definida en dicho universo va a ser equivalente, para cada uno de los *conjuntos* o *sucesos* de ésta como:

\_\_\_\_\_

## El Teorema de Bayes I: Conceptos básicos

Uno de los grandes conceptos que surgió allá a mediados del siglo XVIII fue sin duda el resultado que se conocería como *Teorema de Bayes*, enunciado por Thomas Bayes (1701-1761) durante la década de 1740 en el libro titulado *An Essay Towards Solving a Problem in the Doctrine of Chances*. Para entender mejor el resultado conviene dotar de un poco de contexto al resultado.

Pues resulta que Thomas Bayes, era el mayor de 7 hermanos de una familia presbiteriana, donde su padre, Josua Bayes, llegó a ser uno de los primeros ministros presbiterianos, por tanto la religión tendría una posición central tanto en Josua como en su hijo. En este sentido, resulta curioso que fuera un sacerdote quién estableciera algunas de las reglas del azar más importantes precisamente al tratar de dar solución a una polémica abierta sobre dicha década y que no era ni más ni menos de si era posible o no establecer conclusiones racionales relativas a la existencia de Dios, es decir, Thomas Bayes trataba de “medir” si era más probable la existencia, frente a la no existencia de un ser supremo a partir de evidencias empíricas. En el ensayo citado anteriormente aparecen los siguientes conceptos destacables que se comentan más adelante como son:

* Los conceptos de *independencia* y *probabilidad condicional*
* La distinción de las *probabiilidades a priori* y a *posteriori*

Por tanto, según ichard Price (1723-1791) editor de esta póstuma obra de Bayes, era la la tesis de la existencia de Dios lo que marcaba la obra de Bayes, siendo por tanto pco probable que el propio Bayes tuviese conciencia del desarrollo moderno de lo que más adelante se conocería como *estadística bayesiana* y sus implicaciones en diversas ramas científicas “desnudada” de toda componente religiosa.

Para tener un adecuado acercamiento a la *estadística bayesiana* necesaria en varios de los modelos que se comentan más adelante, deben darse una serie definiciones bastante sencillas e intuitivas como las siguientes:

Definición de *probabilidad* *condicional* *probabilidad*:

La *probabilidad condicional* de un *suceso* *A* condicionado a otro *suceso B* se obtiene del siguiente modo:

\_\_\_\_\_

La idea intuitiva de esta definición se tiene directamente del siguiente diagrama:

Imagen que contiene Aplicación

Descripción generada automáticamente

Es decir, en este caso se quiere pasar de un *universo* a un único *suceso* importando lo que exista en éste. Por tanto el *suceso* que condiciona, el *B* se va a convertir en el nuevo *universo* y de *A* sólo va a interesar el tamaño relativo que quede dentro de *B*, es decir, la intersección dejando de importar todo lo que esté fuera de ésta. También es fácil intuir que en caso de conjuntos disjuntos, la *probabilidad condicional* será siempre 0

Definición de *sucesos* *independientes*:

Un *suceso* *A* se dice independiente de otro *B* si se tiene que:

\_\_\_\_\_

En la definición anterior, lo que se está diciendo es que la probabilidad de *B* no modifica para nada la de *A*, y además se puede demostrar que en el caso de sucesos independientes:

Es decir, en el caso de que exista independencia de sucesos, la probabilidad de la intersección va a ser igual al producto de probabilidades, pero esto como puede observarse no va a ser lo habitual (nótese el caso de 2 *conjuntos disjuntos* ambos con *probabilidad* no nula, no cumplen la anterior relación y además no son *independientes* ya que no cumplen la anterior definición), en general, la probabilidad de la intersección no es el producto de probabilidades, aunque este supuesto sea la base de algunos modelos que se comentan más adelante como es el caso del del Naïves Bayes. En este sentido, para el caso de *universos* con un número finito de individuos se puede dar la siguiente definición:

Definición de *probabilidad* *condicional* *en univeros finitos*:

La *probabilidad condicional* de un *suceso* *A* condicionado a otro *suceso B* se obtiene del siguiente modo:

\_\_\_\_\_

Ejemplo 03: Cosas de familia

Una familia tiene 2 hijos, ¿Cuál es la probabilidad de que los 2 sean niños, dado que al menos uno de ellos es niño?

Para resolver el problema conviene dibujar un lo que se conoce como un *árbol de decisión*, dibujándose todas las posibilidades, en este caso, el modo más sencillo de tener una representación del posible universo es usando un intuitivo y simple diagrama de árbol tal como sigue:

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Como se observa, y dado que el problema no dice nada acerca del orden de nacimiento, sino que sólo que uno de ellos es niño, el dibujo permite dilucidad cuál es el *universo*, que básicamente es señalar todos los caminos donde hay un niño quedando lo anterior como:

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Y por ende, cuáles son los casos probables que es la respuesta a la pregunta planteada::

Nótese además que los *sucesos* no son *indiependientes* ya que:

\_\_\_\_\_

## El Teorema de Bayes II

Vistas las definiciones anteriores, se realiza un acercamiento al resultado conocido como *Teorema de Bayes*.

Para ello se va a hacer uso de la simulación que puede encontrarse en:

*ejemplos/programa01.R*

Que va a ayudar a responder a lo que se conoce como el *Problema de Monty Hall* o *Problema de las 3 puertas*.

Definición *problema* *de* *las 3 puertas*:

Se está en un concurso donde hay 3 puertas: A, B y C; detrás de sólo una de ellas hay un coche mientras que detrás de las otras hay una cabra, se escoge una puerta al azar (donde se cree que está el coche), y en ese momento el presentador abre una de las otras 2 puertas d donde no está el premio, lo que plantea el problema es: ¿Debe cambiarse de puerta?

\_\_\_\_\_

En el código *programa01.R* anteriormente citado, si se abre, entre las líneas 13 y 60 se define la siguiente función en *R*:

montyhall <- function(estrategia='quedarse', N=1000, print\_games=TRUE)

Que va permitir realizar, en este cas unas 1000 simulaciones dada una estrategia elegida por el concursante, de modo que:

1. Si el concursante decide quedarse con su elección entonces hará uso en la línea 67 de: montyhall(estrategia="quedarse")
2. Si el concursante decide cambiar su elección entonces hará uso en la línea 68 de: montyhall(estrategia="cambiar")
3. O bien el concursante puede aleatoriamente elegir quedarse con su elección inicial o cambiar, en cuyo caso elegiría en la línea 69: montyhall(estrategia="quedarse")

Los resultados obtenidos es que al final aproximadamente la *probabilidad* de ganar en *i* es alrededore de 1/3, la de *ii* es de 2/3 y la de *iii* es aproximadamente 1/2. Es decir, si el concursante cambia su elección, lo más probable es que se lleve el premio incluso con aproximadamente un 66’66%, mientras que si no cambia, lo más probable es que lo pierda con un 33’33%.

Lo que se hace en el resto del código si se ejecuta, es repetir las simulaciones anteriores 100 veces (la simulación de la simulación), dibujándose un histograma y obteniéndose su media y desviación típica, generándose los siguiente resultados:

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Figura 1: Resultado de repetir 100 veces cada una de las 1000 simulaciones por estrategia**

**Fuente: Elaboración propia**

Obsérvese que los datos son bastante estables respecto a los valores 0’33, 0’66 y 0.50 no existiendo gandes perturbaciones y es que de hecho, con los coceptos que se conocen hasta ahora, se puede demostrar que la probabilidad aumenta de hecho hasta 2/3 si se cambia de mano, para ello obsérvese el siguiente diagrama explicativo del problema:

**Diagrama

Descripción generada automáticamente**

**Figura 1: Representación del problema Monty-Hall**

Fuente: <https://www.estadisticaparatodos.es/taller/montyhall/montyhall.html>

La clave de este problema, usando los conceptos que ya se conocen, está en plantear los siguientes sucesos:

El jugador selecciona inicialmente, una puerta donde está el coche

El jugador selecciona inicialmente, una puerta donde hay una cabra

El jugador gana el coche

A partir de aquí se puede plantear la siguiente fórmula:

Dado que los sucesos *A* y *B* dividen al universo en 2 partes disjuntas y además la unión de las partes, forma el total al ser sucesos complementarios, se puede aplicar las definiciones que se conocen que relacionan la intersección de sucesos con el concepto de *probabilidad condicionada* llegándose a que:

Ahora se conocen todos los datos, porque por lo pronto, la probabilidad de ganar, dado que se seleccionó inicialmente la puerta donde estaba el coche, sería 0, por tanto, y por otro lado la probabilidad de ganar dado que se seleccionó inicialmente una puerta con la cabra sería 1, por tanto, , por tanto, ahora si resulta claro que:

Y dado que inicialmente el concursante no tiene información ninguna, claramente y , nótese el cambio cualitativo, del instante en el que el concursante no tiene información ninguna, al instante en el que el concursante ya goza de alguna información, en este último caso, lo que hay que hacer claramente es cambiar, la probabilidad crece al 66.66%, ¡Una barbaridad en un juego de azar de estas características! Y si no cambias, sólo ganarás el 33.33% de las veces aproximadamente.

Una vez visto lo anterior se está en disposición de enunciar los 2 resultados centrales de la estadística bayesiana

*Teorema de Probabilidad Total*

Dada una colección *subconjuntos* (o *sucesos* en un determinado *universo*) disjuntos, finita y cuya unión constituye el *universo*, entonces para cualquier *suceso* *B*, se cumple:

\_\_\_\_\_

Intuitivamente lo que viene a decir el resultado anterior es que si un *universo* se divide en partes que no se cortan y que uniéndolas (como un puzle) forma el *universo*, entonces, la *probabilidad* de cualquier *subconjunto* se puede ver como la suma de probabilidades de cada parte que cae (o interseca) en cada una de las piezas que forma dicho puzle ya que de hecho lo que sucede es que, la anterior expresión, es equivalente a:

Teorema de Bayes

Dada una sucesión de *subconjuntos* (o *sucesos*) disjuntos, finita y cuya unión constituye el *universo*, entonces para cualquier *suceso* *B*, se cumple que:

\_\_\_\_\_

La demostración si se quisiera hacer es sencilla con los conceptos que se tienen y haciendo uso del teorema anterior, aunque en tiempos de Bayes, dicha demostración era más compleja.

Lo interesante del resultado es el modo de pensar al que incita la *fórmula bayes* anterior, según la cual se ofrece un método para medir la *probabilidad* de que un determinado *suceso* de una partición sea la causa que de lugar a una *consecuencia* según los datos que realmente se observan

Ejemplo 04: Averiguando las causas, un típico problema de bayesiano

Tres máquinas producen respectivamente el 50%, 30% y 20% de las piezas de una determinada fábrica. Los porcentajes de fabricación defectuosa por máquina quedan repartidos respectivamente según las siguientes tasas: 2%, 4% y 5%. Se toma una pieza al azar y resulta ser defectuosa ¿Qué máquina tiene mayor probabilidad de haberla producido?

Lo importante en estos problemas es elegir los sucesos con los que actuar, en este caso se consideran los siguientes:

La pieza es fabricada por la primera máquina

La pieza es fabricada por la segunda máquina

La pieza es fabricada por la tercera máquina

La pieza es defectuosa

La pieza no es defectuosa

Lo anterior queda aún mejor expresado si se “planta” un árbol que muestre las interrelaciones (nótese que estos árboles son totalmente similares a los que se verán más adelante, sólo que no se aplica un algoritmo para deducir los puntos de corte al venir estos ya determinados).

Gráfico, Diagrama

Descripción generada automáticamente

Para responder a la pregunta planteada, se sabe que la pieza es defectuosa por tanto, el suceso que condiciona es *D*, por lo que se quiere calcular el máximo de entre las 3 probabilidades siguientes: ; y . Para averiguar la causa más probable, se necesita re-expresar el problema de modo que se tengan datos, siguiendo el orden de la casualidad del problema y es donde interviene el Teorema de Bayes y el Teorema de Probabilidad Total (que se aplican uno tras otro), donde se tiene que:

Por lo que:

Y por tanto si se comenzara una investigación para averiguar la procedencia del defecto, debería empezarse por *M2*

# **1.- MODELOS DE ASOCIACIÓN: BASKET MARKET ANALYSIS**

La primera técnica *no supervisada* que aquí se analiza es el *Basket Market Analysis* que desde ahora se denomina *BMA*.

El *BMA* proviene, como su nombre indica del análisis de las cestas de la compra que se hacen en supermercados, por lo que en esencia lo que se busca es ver qué productos se compran conjuntamente por parte de los clientes, de modo que posteriormente se puedan aplicar distintas estrategias de mercadotecnia para aumentar la venta, en cierto modo podría incluso usarse como recomendador de productos “si usted compra leche. ¿comprará también galletas?”. Si se aterriza a un entorno más puramente banca – seguros, la cesta de la compra cambiaría por la cartera de productos que los clientes adquiere[[1]](#footnote-1) conjuntamente estando aquí más próxima al caso de un recomendador de productos sencillo “si un cliente adquiere una cuenta nómina y un crédito al consumo para la compra de un coche ¿Tendrá mayor o menor propensión a adquirir un producto hipotecario?” O bien, dado que ha adquirido dichos productos ¿Cuál es el siguiente producto que va a adquirir? Es esta última línea la que se describe en el siguiente ejemplo que se puede consultar aquí si se desea más detalle:

<https://www.spssanalyticspartner.com/blog-realigning-analysis-in-banking-the-case-of-market-basket-analysis/>

## Conceptos Básicos del modelo

En el *BMA*, se parte de un *universo* de ítems *I* con distintos productos (que serán los individuos o elementos mínimos de la población), Por tanto hay que distinguir 2 conjuntos, el de los *items* y el de las *transacciones* que se realizan con dichos *items* y que en el caso de un supermercado vendrá representado por elecciones realizadas en el ticket de compra:

Se pretende pues formar *reglas de asociación* a partir de dos subconjuntos de items verificando .

Definición de *regla de asociación*:

Es una implicación definida del siguiente modo:

Donde parte izquierda de la regla se denomina *antecedente* y la derecha *consecuente*

\_\_\_\_\_

Definición de *support de un suceso* *X*:

Proporción de transacciones que contienen los ítems que componen el antecedente

\_\_\_\_\_

Definición de *confidence* de una *regla de asociación*:

Proporción de las transacciones donde dándose el antecedente se tiene el consecuente, de entre todas las transacciones del *antecedente*. Es por tanto, un concepto similar a una probabilidad condicionada

\_\_\_\_\_

Definición de *lift* de una *regla de asociación*:

Mejora del *support* *antecedente* - *consecuente* sobre un supuesto de independencia de la regla. Para considerar una *regla* como tal se requiere un valor mayor que 1 y en tal caso cabe suponer evidencia de relación (es decir regla) resulta más fuerte que suponer dependencia que independencia

\_\_\_\_\_

Definición de *leverage* de una *regla de asociación*:

Es una medida que está entre -1 y 1 su interpretación es análoga al de la *lift* sólo que se obtiene por diferencias y suele resultar más sencilla de interpretar, en este caso un valor es próximo a 0 se tendría indepencia:

\_\_\_\_\_

Definición de *conviction* de una *regla de asociación*:

Es una medida que puede estar entre 1 e infinito y trata de medir si la regla tuvo lugar por mero azar o no, en este caso formulación es la siguiente:

A mayor *conviction* menor probabilidad de asociación debida al azar. Resultando en este sentido también que la interpretación de este valor es similar al de la *lift*.

\_\_\_\_\_

Como se observa, hay un problema claro de Big-Data subyacente cuando ya que el número de *transacciones* posibles va a resultar por lo general elevado, por lo que los distintos softwares que dan solución a estos problemas utilizan distintos formatos para su tratamiento como los siguientes:

-Definición de *matriz de incidencia*: donde las filas son las transacciones y las columnas los ítems que pueden encontrarse en éstas de modo que se tendrá un 1 si el ítem está y un 0 en caso contrario. Esta estructura tiene la ventaja de la interpretabilidad, pero la desventaja de que en algunos casos puede resultar demasiada pesada

\_\_\_\_\_

-Definición de *matriz de dispersa*: son un tipo de matrices con orientación de columnas en vez de filas. Estas matrices almacenan etiquetas y gestionan el correspondiente mapeo a lo que sería una matriz de incidencia, aunque resulte más difícil su interpretación, permiten un modo de almacenamiento más simplificado y en ocasiones hasta más rápido en tiempos de computación

\_\_\_\_\_

Tabla

Descripción generada automáticamente con confianza media

**Figura 2: *Matriz de incidencia* vs *matriz dispersa* de un *BMA* tradicional**

**Fuente: Elaboración propia**

## Descripción del algoritmo

El algoritmo que se describe a continuación ofrece una serie de pautas para que puedan estimarse para un *universo* y unas *transacciones* realizadas sobre el primero, tanto las asociaciones de productos más importantes, como la longitud y por supuesto algunos grados de ocurrencia de determinadas *reglas* en base a los coceptos comentados anteriormente. Para llegar a dar unas caracterizaciones como las que se han apuntado, debe tenerse en cuenta que el número de posibles *reglas de asociación* a construir en un determinado par *universo* – *transacciones* puede llegar a resultar muy elevado:

Ver: <https://elvex.ugr.es/decsai/intelligent/slides/dm/d2%20association.pdf>

Un algoritmo que resulta válido y que viene en las correspondientes librerías de *R* o *python* resulta ser el denominado *algoritmo a priori* propuesto por *Agrawal* y *Srikant* en 1994 y que consta de los siguientes pasos

-Primero se establece un nivel mínimo de *support* que cabe denominar *U*

*-*Se comienza definiendo el siguiente conjunto de transacciones unitarias (items) con *suppor* mayor a *U*

Con *j* recorriendo los items desde 1 hasta *n*

-Paso *k* Para este paso debe existir un conjunto no vacío *Lk-1* cuyos elementos son transacciones de longitud *k – 1* y soporte al menos *U*. El conjunto *Lk* se construye del siguiente modo:

i) De entre los *items* existentes en las *transacciones* de *Lk-1* se crean todas las posibles transacciones de tamaño exactamente *k*

ii) Si no existe ninguna transacción de longitud *k* y *support* mayor o igual a *U*, el algoritmo finaliza, en caso contrario se ejecuta un paso *k* + 1

-Fin del algoritmo: Una vez, aplicado el paso anterior *k* veces y no se encuentre ninguna transacción de *support* mayor o igual a *U* las reglas a considerar serán las de orden 1, 2, … *k –* 1 (en caso de que de finalización en *k*)

Como se observa la idea es tratar de reducir el universo de construcción de reglas a unas pocas que ofrezcan además cierto grado de ocurrencia en términos de *support* o incluso elementos adicionales que en algunas funciones se pueden considerar como el *confidence* o/y la *lift*. Algunas alternativas a este algoritmo que no se desarrollan aquí, es el denominado *Eclat* (*equivalence class transformation*) que puede resultar más rápido aunque en consuma más memoria y el *FP-growth* (*frequent patterns*)

## El modelo *BMA*. Aplicaciones y ejemplos.

Con el anterior *algoritmo* se genera el modelo que en este caso consiste en una colección de *transacciones* de distinta longitud (de 1 a *k*) donde además se podrían obtener expresiones del tipo *antecedente* – *consecuente* como se observa en las aplicaciones.

A continuación, se desgrana un código realizado en Python donde se utilizan y particularizan cada uno de los conceptos anteriores tratándoles de dar un sentido de cohesión.

Ejemplo 05: desarrollo de un *BMA* tradicional en *python*

Con el código en *python* denominado *ejemplos/programa02.py* se despliega un primer ejemplo con datos realistas y que puede reproducirse sin mucho problema con tal de instalar la librería *mlxtend* que es la librería báisca para el *BMA* en python aparte de las otras librerías estándar que en dicho código se considera y que se usan para otros cometidos. Este ejemplo se extrae y se modifica a partir de la siguiente fuente que puede consultarse si se desea más información:

<https://pbpython.com/market-basket-analysis.html>

Para este ejemplo se hace uso del entorno de programación en python denominado *spyder*, mostrándose a continuación algunos resultados relevantes que se amplían y se detallan en la correspondiente clase:

- Primera visión de los datos: Siempre es conveniente mirar un poco lo que se tiene, en este caso si se ejecuta hasta el *df.head()* de la línea 14 se obtiene un breve resumen del dataset a analizar:

Imagen que contiene tabla, grande, computadora, agua

Descripción generada automáticamente

- Es habitual, antes de aplicar el algoritmo hacer una selección y preparación de la información en esta primera parte se seleccionan sólo los datos de Francia se construye una *matriz de incidencias* que será lo que se introduzca en la función que ejecutará el *algoritmo* para obtener el posterior modelo, cosa que se logra si se ejecuta hasta la línea 49 la cual genera una visión esquematizada de la *matriz de incidencias*, que tiene unas 392 *transacciones* pero 1562 columnas, tantas como *items* (o productos) del *universo*. Para llegar hasta aquí hay que ejecutar hasta la línea 49, nótese la matriz que se genera en el área de *Spyder* denominado *Variable Explorer*. Mostrándose la siguiente visión parcial de la citada matriz

Aplicación

Descripción generada automáticamente con confianza baja

- Una vez preparados los datos, se aplica el *algoritmo a priori* a la anterior *matriz de incidencias* con la siguiente línea:

frequent\_itemsets = apriori(basket\_sets, min\_support=0.07, use\_colnames=True)

Nótese que simplemente se exige un *support* de 0.07 y que para entrenar el algoritmo anteriormente descrito sólo ha bastado una única línea

- Una vez *entrenado* el *algoritmo* se obtiene el *modelo* que en este caso son todas las reglas de asociación de orden 1, 2, … Para poder observar algunas cabe aplicar algunos filtros al modelo como:

strong\_rules = rules[ (rules['lift'] >= 6) &

(rules['confidence'] >= 0.8)

De modo que si se llega a la línea 78 se obtiene la siguiente salida:

Texto

Descripción generada automáticamente

Nótese que sólo aparecen elementos con las *lift* y *confidence* exigidas, por lo que un modo de incrementar ventas sería colocar juntos el item *ALARM CLOCK BAKELIKE RED* con el *ALARM CLOCK BAKELIKE GREEN*, quizás el tener la “pareja” declina mucho en la compra al cliente, nótese además que en este caso el orden de la cesta es 2 y que aparece tanto la implicación directa como la inversa con resultados bastante elevados de *lift* y *confidence* por lo que parece, al menos en el mercado francés de este producto un hecho a tener muy en cuenta en una estrategia de venta - marketing

-Finalmente con ligeros cambios se puede hacer lo mismo de antes para el caso de Alemania y analizar los diferentes resultados

\_\_\_\_\_

Ejercicio 01: alguien nos deja un “pufo”

Vuestra empresa contrató un consultor para la realización de un proyecto de tipo *BMA*. Cuando llega el momento de la entrega, os dejan un código R y no en Python como habíais pedido junto con unos datos de demo, pero en el código que os dejan hay además algunos bugs, que impiden su correcta ejecución.

El consultor os comenta que en los requerimientos no estaba claro si se quería *R* o *python* porque está escrito sencillamente que preferiblemente el desarrollo debería estar en *python*, pero no se indicaba su obligación, por lo que su apaño requería una ampliación del proyecto por unas 2 semanas adicionales para apañar el código y convertirlo en *python*. Además no se hace responsable de los *bugs* porque dice que en una demo todo funcionaba perfectamente, que requeriría más tiempo de desarrollo …

Vosotros sois los que habéis seguido el proyecto y vuestra empresa os indica que no hay más presupuesto para consultoría. ¿Qué hacéis? ¿Seríais capaces de arrancar el código denominado *ejercicio01.R* a partir del código *R* proporcionado y una muestra de datos? Además, vuestro responsable os pide que saquéis unos resultados y los interpretéis para la próxima Town Hall donde se hará una presentación conjunta de todos los proyectos. ¿Podríais además interpretar los resultados finales?

Nota: Nótese que el fichero de pruebas *Groceries.csv* viene dado como una *matriz dispersa*, en este caso, *R* dispone de la función *read.transations()* de la librería *arules* que permite su adecuada lectura para pasarlo posteriormente a análisis

\_\_\_\_\_

Ejercicio 02: Hagámos un *MBA* de modo manual

Se ofrece la siguiente tabla de transacciones y se pide responder a una serie de cuestiones:

Tabla

Descripción generada automáticamente

1. ¿Cuántas cestas de la compra podría haber con los items aquí considerados?
2. Si se impone una regla del tipo min\_sup = 2 créese una tabla con dichos ítems y sus supports correspondientes de una única combinación
3. Realícese lo mismo que en b) pero con cestas de 2 ítems
4. Realícese lo mismo que en c) pero con cestas de 3 ítems ¿Hay alguna cesta de 4 o más ítems que cumpla la condición min\_sup?
5. Estímese para cada posible combinación de las cestas en c), su nivel de *confidence*
6. Si se establece un nivel mínimo de *confidence* del 50% ¿Qué posibles combinaciones reglas de asociación se formarían?

# **2.- INTRODUCCIÓN A LAS REDES BAYESIANAS**

Las redes bayesianas son modelos gráficos de razonamiento bajo incertidumbre, tuvieron gran fama a finales de los 80 como elementos clave en la construcción de lo que se denominaban *Sistemas Expertos*. Estos modelos se componen de una serie de nodos (o vértices) que representan variables (discretas o continuas) y los arcos o aristas entre ellos que son las direcciones de las conexiones (o relaciones) entre ellos.

De nuevo se está ante modelos donde no se supone una distribución particular entre las observaciones y además en algunos casos, con “*juicios expertos*” podrás ser más que suficiente para modelizar ya que interesará dar credibiliad a determinadas relaciones causales entre un determinado conjunto de variables.

## 2.1 Conceptos básicos del modelo

Los modelos que se van a construir van a ser de tipo *grafo* muy visuales y entre sus elementos principales caben encontrar los siguientes, muchos de ellos propios de la *Teoría de Grafos*:

Definición de *nodo*:

Es un elemento de un conjunto denominado “*conjunto de nodos*”que es el *conjunto* de *variables aleatorias* que se consideran en el análisis y que podrían representarse como:

\_\_\_\_\_

Definición de *aristas* o *arcos* o *links*:

Son la representación de la dependencia entre los *nodos* o *variables*, por lo que una dependencia entre 2 *nodos* se representa por una línea (o *grafo dirigido*) como el siguiente:

\_\_\_\_\_

Definición de *grafo directo* o *dirigido o digrafo*:

Es un tipo de grafo en el que las aristas tienen un sentido definido (punta de la flecha de la definición anterior)

\_\_\_\_\_

Definición de *grafo no cíclico*:

Es un tipo de grafo en el que dado un *nodo*, exista un *camino* (o sucesión de *nodos* intermedios) que comiencen y acaban en dicho *nodo*

\_\_\_\_\_

Definición de *Red Bayesiana*:

Este un modelo tipo *grafo dirigido, no cíclico* y *conexo* cuyas aristas se representan mediante *probabilidades condicionadas* cumpliendo la *propiedad de Markov*. Una *Red Bayesiana* es una estructura gráfica que permite representar y razonar acerca de un dominio incierto. Por lo que asociado a una *Red Bayesiana*, existe siempre un *grafo bayesiano* a veces denominado *árbol de interrelaciones bayesianas*, que representa a la red o al modelo

\_\_\_\_\_

Nota: Más adelante por razones de expocisión se comenta la *propiedad de Markov* cuyo conocimiento no es indispensable para el siguiente ejemplo de una *Red Bayesiana*

Ejemplo 06: expresión del *grafo bayesiano* o *árbol de interrelaciones bayesianas*. Un paciente impaciente

Business Case: Un paciente ha sufrido una falta de aliento (disnea) y visita al doctor, preocupado porque pueda padecer cáncer de hígado. El doctor sabe que otras enfermedades como la tuberculosis o la bronquitis pueden ser también posibles causas de lo anterior, además sabe que otra información puede ser relevante como si el paciente fuma o no fuma y el tipo de contaminación atmosférica al que dicho paciente ha sido expuesto. Finalmente, un análisis de rayos X indicaría si se pudiera dar o bien una Tuberculosis o bien un Cáncer con una determinada probabilidad. Así pues, toda la información a considerar, respecto a los nodos cabe ser considerada en la siguiente tabla:

|  |  |  |
| --- | --- | --- |
| Nombre del Nodo | Tipo | Valores |
| Contaminación | Binaria | Baja, Alta |
| Fumador | Booleana | V, F |
| Cáncer | Booleana | V, F |
| Disnea | Booleana | V, F |
| Rayos – X | Binaria | Pos, Neg |

Paso 1: Definición de los *nodos* y de sus posibles valores a tomar por éstos:

Paso 2: Elección de la Estructura Topológica de la Red. Se establece el grafo

Diagrama

Descripción generada automáticamente

Del grafo anterio se observa que:

* Existe una relación de tipo Parent-Child
* Existen unas *probabilidades condicionadas* ya dadas
* También se observan las *probabilidades totales* de los *nodos raíz* de la red que expresan las *probabilidades a priori* (un 90% de la población está expuesta a un baja contaminación y un 30% de la población son fumadores)

Con lo anterior se ha establecido una primera *red bayesiana* y se ha ofrecido un *modelo* tipo *grafo*, lo que viene a continuación es ¿Cómo se va a poder operar con el anterior modelo y las tablas de probabilidades previamente dadas?

\_\_\_\_\_

Como se observará en los ejemplos, en las *redes bayesianas* hay 2 tipos de razonamientos:

* Razonamiento de Diagnóstico: A partir de los síntomas (o consecuencias) se pretende encontrar las causas
* Razonamiento Predictivo: A partir de evidencias a priori, deducir las posibles consecuencias

Pero antes se define el elemento clave que va a permitir operar con facilidad a una modelización como la mostrada en el anterior ejemplo.

*Propiedad de Markov*:

No existen dependencias directas, si éstas no están expresadas mediante arcos (directos), entonces no existe dependencia

\_\_\_\_\_

Es decir, la anterior propiedad implica que, en el ejemplo anterior, se tendría que no existe una relación entre el hecho de ser *Smoker* y que padecer *Dyspnoea* si no se considera el nodo denominado *Cancer* es decir, tiene que existir un *arco* directo y por tanto el ser  *Smoker* sí tiene incidencia en pader *Cancer* y éste a su vez en hacer que se presente síntomas de *Dyspnoea*.

Un hecho derivado de la anterior propiedad es lo que se conoce como la *independencia condicional*.

Definición de *independencia condicional en una red bayesiana*:

Es la que proviene de aplicar la condición de Markov y la que permite operar con las probabilidades conjuntas. Viene a indicar que los valores un nodo particular, depende sólo de los nodos *parents* o superiores directos tal y como se observa en el siguiente diagrama.

Diagrama

Descripción generada automáticamente

\_\_\_\_\_

Definición de *probabilidad conjunta de una red*:

Es la función conjunta de probabilidades de la red.

Esta función mediante el uso del *teorema de Bayes*, cumple la conocida *regla de la cadena* que:

\_\_\_\_\_

Ejemplo 07: Obteniendo un valor de la *probabilidad conjunta* del “paciente impaciente”

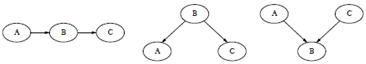
Habitualmente, tras los análisis a realizar y tras un determinado test, un paciente responderá a algunas cuestiones y por tanto, no se va a requerir por lo generar el cálculo de la *probabilidad conjunta* de toda la red en sus numerosas opciones, pero para mostrar el caso de un cálculo en concreto, supóngase que se desea a saber, en base a los datos de los que se dispone, cuál es la probabilidad de que un paciente escogido al azar dé positivo en Rayos X, tenga Dipsnea, tenga Cáncer, viva en una zona de baja polución y sea fumador

Mediante el uso de la *propiedad de Markov* e *independencia condicional*, según el ejemplo 06

Es decir, sería muy raro que se diese toda esa combinación de factores simultánea

\_\_\_\_\_

Si se retoma por tanto el concepto de *independencia condicional en redes bayesiana*,los principales tipos de relaciones a considerar cuando se dispone de 3 *nodos* serían los siguientes:



Siendo el primer caso el más sencillo de tratar como se ha visto antes, mientras que en los otros 2 las *probabilidades condicionadas* bien se reparten o bien se concentran sobre un determinado *nodo*.

Definición de *creencia* y *evidencia*:

Es la *probabilidad a posteriori* (hipótesis) en base (*condicionada*) a las *evidencias* y se denota por:

Las *evidencias* van a ser valores que podrán ser específicos o conocidos en términos absolutos (las más frecuentes) o incluso pueden ser de tipo virtual (o probabilístico)

\_\_\_\_\_

## 2.2 Construcción de Redes Bayesianas

Con lo visto hasta ahora, se tienen los elementos para construir y analizar *redes bayesianas* aunque el cálculo sería tedioso, de modo que para la construcción del modelo como el uso de algoritmos que nos permitan inferir *probabilidades* que es el objetivo último del modelo, hay paquetes de software específico, aunque los 2 más utilizados en Data Sciencie como son R y Python. Aquí se consideran el uso de los 2 anteriores aunque el uso de paquetes específico permite más versatilidad y potenciacia de cálculo en un problema cuya complejidad es de tipo NP.

Como primer algoritmo constructivo para generar posibles modelo de *redes bayesianas* se encuentra el *algoritmo de Pearl*

Definición de *algoritmo Pearl*:

1. Escoger un conjunto de variables aleatorias relevantes que describan el dominio
2. Elegir una ordenación entre dichas variables
3. Si existen variable sin elegir:
4. Añadir la próxima variable a la red
5. Añadir arcos a dicha variables desde un conjunto minimal de nodos que satisfagan la propiedad de Markov
6. Definir las Tablas de Probabilidad Condicional, de las variables child en función de sus parents

\_\_\_\_\_

Lo anterior son las pautas genrales que debe cumplir un modelo de este tipo y aunque pueda haber muchas posibilidades, muchas serán acotadas por la lógica en sí del problema a analizar y la disponibilidad o no de las adecuadas *tablas de probabilidad condionadas* que son el “core” del modelo a usar, así pues:

- Una red dependerá de la ordenación inicialmente asignada a los nodos

- En base a lo anterior, distintas ordenaciones pueden dar lugar a redes con y sin sentido con respecto a la investigación considerada

- Las redes deben de construirse del modo más compacto posible, es decir, aunque las redes sean iguales, la representación puede complicar su apariencia como se observa a continuación:

Diagrama

Descripción generada automáticamente

**Figura 3 Representación de la misma red bayesiana: compacta vs no compacta**

Fuente: Korl, K; Nichols, A. 2011

Así pues antes de practicar un ejemplo con *R* donde se observará la cantidad de manualidades que nos podemos ahorar, se van a dar un conjunto de definiciones adicionales que pueden ser útiles para entender mejor estos modelos de *grafos* como son las que se enuncián a continuación.

Definición de *camino*:

Dados 2 *conjuntos de nodos* se dice que existe un camino entre ellos si dada una estructura de red, se puede encontrar *nodos* intermedios y adyacentes conectados entre sí por *arcos* uniendo algún miembro del primer conjunto con alguno del segundo. Además, debe cumplirse que en dicha secuencia ningún nodo aparezca repetido

\_\_\_\_\_

Definición de *camino* *bloqueado*:

Dado un conjunto de nodos , un camino está bloqueado si hay un nodo en el camino (el que bloquea) para el que se cumple al menos una de las siguientes propiedades:

1. está en y existe un *arco* en el *camino* que llega y sale de dicho *nodo*
2. están en y todos los arcos salen de él
3. Ni ni ningún descendiente de él está en está y todos los arcos se dirigen hacia

\_\_\_\_\_

Definición de *camino d-separador*:

un conjunto de nodos d-separa a otros dos conjuntos de nodos denominados e si dicho conjunto bloquea cualquier camino que pueda construirse entre dichos nodos con lo que en este caso se dirá que los nodos e son condicionalmente independientes dado

Diagrama

Descripción generada automáticamente

\_\_\_\_\_

Por tanto llegado a este punto ¿Qué se pretende hacer con una *red bayesiana*? El objetivo principal va a ser el de hacer *inferencia* y por tanto se trata de calcular las *probabilidades a posteriori* o lo que es lo mismo, llevar a cabo el procedimiento de actualización de creencias.

Aunque aquí se analizan redes muy pequeñas, hay que tener en cuenta que en casos en los que dichas redes resulten ser grandes, el problema teóricamente se convierte en los denominados problemas NP-duros y resultarán complejos resolver en términos computacionales así pues, elementos como la *compacidad de la red*, *reducción de la conectividad*, *reducción de los bucles no-direccionales* y la *localización de evidencia* y de los *nodos query* podrían ser claves para que los problemas modelizados mediante esta técnica sean abordables.

Ejemplo 08: El turista asiático

En esta ocasión se va a hacer uso de *python* y en concreto del código *programa03.py* para construir una *red bayesiana* basada en la típica *Red Asia* y a partir de las *probabilidades condicionadas* dadas en el paper original *Daly, Rónán & Shen, Qiang & Aitken, Stuart. (2011)*:

Tabla

Descripción generada automáticamente

En este ejemplo, un turista ha visitado un país asiático y llega con una serie de síntomas al hospital, en correspondiente código del *ejemplos/programa03.py,* se observa como se introduce fácilmente cada una de las probabilidades totales y condicionales en cada uno de los nodos que forma la rede bayesiana, en su construcción se siguieron los siguientes pasos:

Paso 1. Instalación de las librerías necesarias de modo básico, dichas librerías son: *pybbn* para crear la *rede bayesiana* junto a su matemáitca y cálculos asociado y para su representación, estaría la librería *networkx*

Paso 2. Se introducen las tablas de probabilidades anteriores, estas probabilidades son los supuestos a priori anteriormente comentados y se supone que, tras nueva información, todo irá cambiando, pero en este primer ejemplo, sólo se tratan estos supuestos que se toman como referencia del siguiente modo:

Texto

Descripción generada automáticamente

Paso 3. Se crea la arquitectura de red con las relaciones entre los distintos nodos:

Texto

Descripción generada automáticamente

Paso 4. Finalmente se representa la red:

Diagrama

Descripción generada automáticamente

Observación nótese cómo Python permite controlar todos los elementos tanto de arquitectura como de representación final de la red de un modo muy natural, dando los pasos razonables para su adecuada representación.

\_\_\_\_\_

## 2.3 Inferencia estadística en redes bayesianas

Una de las principales aplicaciones de las *redes bayesianas* van a ser las predicciones o más bien las *inferencias* que caben realizar con ellas, en este sentido los principales tipos de *inferencias* son las siguientes:

-La *inferencia exacta en cadenas*

-La *inferencia exacta en poli-árboles*

-La *inferencia con evidencia incierta*

-La *inferencia exacta en redes multi-conectadas*

-La *inferencia causal*

Aquí se analizará el primer caso por ser el más sencillo y habitual y se comentará brevemente el tercero porque en ocasiones puede ser interesante añadir un punto de incertidumble a la ocurrencia de un hecho bajo este enfoque.

### Inferencia exacta en cadenas

En el caso más sencillo lo que se tiene es una cadena formado por 2 nodos del tipo En este caso puede haber 2 posibilidades:

* Que exista evidencia del *nodo* superior en cuyo caso:
* Que exiata evidencia del *nodo* hijo o inferior, en cuyo caso:

En este caso es la *probabilidad a priori* mientras es la *verosimilitud* con:

Una vez se tiene el caso sencillo controlado, se puede aplicar el método para el caso de una red de tres nodos: donde en este caso se van actualizando las probabilidades desde el *nodo* origen hasta el final

* Si existe evidencia del Nodo Padre:
* Si existe evidencia del Nodo Hoja:

Ejemplo 09: El turista puede estar enfermo.

Claramente realizar los anteriores cálculos a mano, sería complejo y es bastante posible que se cometa algún error, en este sentido los paquetes informáticos *python* y *R* ofrecen la solución una vez creada la *rede bayesiana*, así pues si se toma el anterior ejemplo, dada la información disponible cabe estimar la probabilidad de que un individuo de la población padezca de dypsnea

Para ello es necesario calcular las *probabilidades marginales* a partir de:

|  |  |
| --- | --- |
|  |  |

Es decir, que bajo esta red, la probabilidad de tener dyspnea es de un 38.81%

Sin embargo, lo normal no es que se tenga que realizar una *inferencia* de *arriba* abajo sino que lo que suele ocurrir es que se tienen, tras unas pruebas médicas o un cuestionarios unas *evidencias* de la ocurrencia de alguno de los *nodos* y lo que se va a necesitar es tener respuesta de cómo se comportan los otros, sí pues cabe plantearse la siguinete pregunta: ¿Cuál es la probabilidad de que se tenga dyspnea dado que sabe que smoke = ‘Yes’ (con probabilidad 1)? Que se puede resolver del siguiente modo.

|  |  |
| --- | --- |
|  |  |

Donde se observa que en este caso, dicha probabilidad aumenta como cabía esperar

Además, quizás más interesante pueda ser la siguiente pregunta en la que se trata de averiguar la ocurrencia de una variable, en principio no observable ¿Cuál es la probabilidad de que tenga bronquitis dado que se es fumador y se padece de dypsnea?

Texto

Descripción generada automáticamente

Donde se observa que la probabilidad de padecer bronquitis aumenta hasta algo más del 87% mientras que, en el estado inicial, dicha probabilidad era de algo más del 44% cuando no se tenía ningún tipo de conocimiento.

### Inferencia con Evidencia Incierta

La noción de *evidencia incierta*, también conocida como *virtual* o *probabilística*, indica un grado de ocurrencia que no va a ser del 100%. Para ello lo que se hace es considerar una nueva distribución sobre el nodo en sí donde va a existir la incertidumbre, así pues, si para un nodo simple , va a existir una incertidumbre (o una credibilidad) acerca de la veracidad de su información y que se denota por , esta se representa como en la siguiente figura (ejemplo de una credibilidad del 80% de que un hecho haya ocurrido de verdad):

Diagrama

Descripción generada automáticamente

Es decir, lo que se haría en una red, es introducir un nuevo nodo con su correspondiente tabla de *probabilidades condicionales CPT*. En los algoritmos de paso de mensajes, ya comentados, las uniones con respecto a los *nodos virtuales* se expresan como mensajes que se transmiten sólo en una única dirección, el mensaje va en dirección child-parent tal y como se muestra en el siguiente ejemplo

Un reloj digital

Descripción generada automáticamente con confianza media

Ejercicio 03: ¿Pagará o no pagará?

Se quiere crear un modelo de tipo casual, donde a partir de relaciones de ocurrencia de distintas variables, se tenga una idea de la transmisión de la probabilidad de ocurrencia de un impago, o bien:

Diagrama

Descripción generada automáticamente

Con las siguientes tablas de *probabilidades condicionadas*:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | edadElevada = S | 60% | | |  |  | | --- | --- | | sexo = H | 70% | | |  |  | | --- | --- | | economia = P | 60% | |

|  |  |  |
| --- | --- | --- |
| Ahorro | | |
| edadElevada = S | trabajo = S | 80% |
| edadElevada = S | trabajo = N | 40% |
| edadElevada = N | trabajo = S | 50% |
| edadElevada = N | trabajo = N | 30% |

|  |  |  |
| --- | --- | --- |
| Trabajo | | |
| edadElevada = S | economia = P | 30% |
| edadElevada = S | economia = N | 10% |
| edadElevada = N | economia = P | 80% |
| edadElevada = N | economia = N | 40% |

|  |  |
| --- | --- |
| vivienda | |
| economia = P | 70% |
| economia = N | 40% |

|  |  |  |
| --- | --- | --- |
| dificultad | | |
| ahorro = S | trabajo = S | 1% |
| ahorro = N | trabajo = S | 5% |
| ahorro = S | trabajo = N | 40% |
| ahorro = N | trabajo = N | 80% |

|  |  |  |
| --- | --- | --- |
| paga | | |
| dificultad = S | vivienda = P | 60% |
| dificultad = N | vivienda = P | 98% |
| dificultad = S | vivienda = N | 10% |
| dificultad = N | vivienda = N | 95% |

Nota: Véase el doc de la carpeta *ejercicio03*

# **3.- ENTRENAMIENTO NO SUPERVISADO I: INTRODUCCIÓN AL ANÁLISIS CLUSTER**

Como paradigma del *análisis no supervisado*, se encuentran las *técnicas de clustering*. El objetivo aquí es tratar de llevar asociaciones entre elementos de una población en base a valores de todas o parte de sus características sin que exista a priori un conjunto de etiquetas asociadas a las observaciones de modo análogo a como se hizo anteriormente, pero en este caso se trata de hacer agrupaciones directas sin construir directamente la causalidad, que vendría dada en todo caso tras una interpretación de los resultados obtenidos, lo cual no resultará siempre sencillo.

Existen como 3 grandes conceptos sobre los que hacer *clustering* y que son los siguientes:

* *Clustering de individuos*
* *Clustering de variables*
* *Clustering de series temporales*

En esta sesión se tratan los 2 primeros, dejándose el tercero para el tema relativo a *series temporales*.

## 3.1. Preliminanes a las técnicas de clustering

Para entender bien lo que viene a continuación, merece destacarse algunos elementos matemáticos adicionales. Así pues, en *análisis cluster* el elemento esencial que subyace por debajo de todos los algoritmos es el concepto de *distancia*, lo cual tiene sus implicaciones como se destaca a continuación:

* Usar *distancia*s en un algoritmo, implica lentitud en éste cuando la población a clusterizar es elevada, ya que se requiere construir matrices simétricas siendo las operaciones a realizar de orden cuadrático.
* Implica que las variables deberían de estar debidamente *homogeneizadas* o *normalizadas*, ya que al final, al existir una diferencia entre coordenadas en los cálculos de las distancias, se corre el riesgo de sesgar hacia una u otra componente

Definición de distancia (en espacios métricos -completos-):

Dado un *espacio* (o *población* de individuos) se define como *distancia* a una aplicación *d* que verifica las siguientes propiedades:

\_\_\_\_\_

En estadística se usan *distancias* de tipo T2 que son las *aplicaciones* que cumplen las anteriores propiedades considerándose en ocasiones aplicaciones que no son propiamente *distancias* como la *pseudo-distancia* *Kullback-Leibler*.

Ejercicio 04: ¿Son distancias métricas o T2 las siguientes *aplicaciones*?

\_\_\_\_\_

Como se puede observar hay muchas *distancias*, entre ellas cabe destacar la más habitual e intuitiva como es la *distancia euclídea*, que es la que se define sobre los *productos cartesianos* de *espacios multivariantes* y que es la que habitualmente se tiene entre manos en estadística, es decir, en general se trabaja con un individuo que se convierte en vector de *n-variables* y por tanto éste queda *embedido* en un *espacio* tipo es decir, se convierte en un vector con tantas componentes como variables se consideren de dicho individuo. Así pues, en este tipo de *espacios*, la *distancia euclídea* se define tal como:

Otras distancias también utilizadas son la distancia taxi o Manhattan y la de Mahalanobis que se definen respectivamente como sigue:

Nótese el caso especial de la distancia de Mahalanobis. Bajo esta distancia lo interesante es la distancia entre variables teniendo en cuenta su correlación, por lo que en cada caso hay que construir la matriz de covarianzas entre todas las variables, para posteriormente llevar el cálculo pedido.

Ejemplo 10: El tamaño importa.

Se considera el código denominado *ejemplo/programa04.py* donde:

* En primer lugar, se consideran 5 clientes, que se desean agrupar bajo 2 variables, edad e ingresos anuales. Aquí se observa que al construir la *matriz de distancias*, bajo una *distancia* de tipo *euclídea* se obtiene el siguiente resultados:

|  |  |
| --- | --- |
|  |  |

* Bajo este ejemplo, los clientes más cercanos serían el segundo con el cuarto (aunque parece tener más sentido asociar los 2 primeros o los 2 últimos), ya que prácticamente la edad no tiene importancia numérica, sin embargo, si se lleva a cabo una normalización de media 0 y desviación 1, los datos se transforman en:

|  |  |
| --- | --- |
|  |  |

* Quedando ahora como clientes más cercanos los 2 últimos (los mayores) y también como siguiente más cercanos los 2 primeros, lo cual tiene lógica tal y como se observa en los datos a simple vista.
* Por otro lado, si en el ejemplo, se hubiera aplicado directamente la distancia de Mahalanobis (para lo que se requiere un poco más de código, el resultado habría sido semejante:

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

Cuando se introduce la estructura de interdependencia de las variables a través de la matriz de covarianzas, se obtiene “de una tacada” conclusión también en línea con el sentido común de este ejemplo.

En resumen, en todos los algoritmos donde intervienen *distancias*, va a influir tanto el tipo de *distancia* a emplear, como transformaciones en los datos como la *normalización* y por tanto será habitual que en los *análisis clúster* distintos analistas, obtenga con los mismos datos distintas conclusiones, siendo aquí por tanto clave la interpretación de los resultados y no dejar al algoritmo por sí solo a realizar asociaciones que en muchos casos tendrá poco o nulo sentido.

\_\_\_\_\_

## 3.2. Clustering de individuos

En este caso se desea la asociación de individuos en grupos, existiendop 2 tipos de técnicas de clasificación tal y como se detalla continuación:

Interfaz de usuario gráfica, Diagrama

Descripción generada automáticamente

**Figura 4: Taxonomía de las principales técnicas de clasificación**

Fuente: <https://www.diegocalvo.es/cluster-jerarquicos-y-no-jerarquicos/>

Es importante tener un poco de conciencia sobre el elevado número de agrupaciones posibles a realizar, lo importante por tanto de un *algoritmo* por tanto es que sea capaz de discernir de entre el *número de Stirling de segunda especie* un subconjunto de las posibles de entre las siguientes:

A lo que habría que añadir que en general, el número de grupos es desconocido, por lo que el número de posibilidades es además la suma de *números de Stirling* como los anteriores, con lo que en el caso de *m* observaciones, se tendría que el número total de posibles clasificaciones sería el siguiente:

En esta sesión se analizan 2 técnicas básicas que darán lugar a un número acotado de agrupaciones con elecciones subjetivas por parte del investigador, una de tipo *no jerárquico* como es el algoritmo *k-means* y otro de tipo *jerárquico* como es el *método de Ward* o de varianza mínima.

### Algoritmo k-means

Es un *algoritmo* que pertenece a la familia de los clustering no jerárquicos, por tanto, implica que en la aplicación de este algoritmo hay que elegir un número de grupos *k*, para que posteriormente se haga la agrupación. Además, esta elección es fundamental en la aplicación del algoritmo *k-means* porque básicamente es el punto de partida del algoritmo, a partir del cual se tiene el número de *centroides* tal como sigue:

* Paso 1: Se genera un número de *k* *centroides* elegidos por el usuario en el *espacio* o *población* de individuos
* Paso 2: Se calcula la *distancia* de cada individuo a cada *centroide* y se asigna cada individuo al centroide más cercano
* Paso 3: Se recalculan de nuevo los *centroides* a partir de las asociaciones realizadas en el paso 2, en este caso, como los individuos están asociados a esos *k* *centroides*, se obtiene un *centroide* por grupo
* Paso iterativo: Se aplica 2 y 3 hasta que los centroides estabilicen su posición

El funcionamiento de este *algoritmo* se ve de modo muy sencillo con la siguiente representación en *shiny*.

Ejemplo 11: Agrupando flores de Iris

Una *app shiny* muy sencilla que se puede construir basda en la librería de R *kmeans* y haciendo uso del conjunto, integrado en R.base denominado *iris* es la que viene el siguiente enlace:

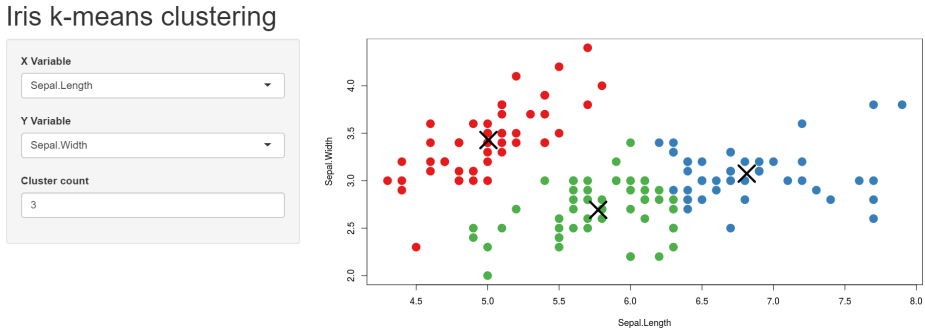
<https://shiny.rstudio.com/gallery/kmeans-example.html>

El código para generar dicha aplicación se puede obtener desde el botón *Get Code*, y en este caso se debe crear una carpeta vacía en nuestro ordenador donde descargar estos 2 ficheros que se denominan *ui.R* y *server.R*. Justarmente en el código denominado *server.R* se tiene el siguiente fragmento:

Texto

Descripción generada automáticamente

Se observa que la *función* en R base existente para llevar a cabo el análisis es la denominada *kmeans()* de modo que para que funcione adecuadamente necesitan al menos 2 elementos, los datos y el número de clúster a realizar, a partir de ahí, todo el resto de salidas que se obtienen son auto-explicativas.



\_\_\_\_\_

Visto lo anterior caben enunciar las siguientes propiedades de los *clusters*:

* Todas las observaciones están asignadas a un *cluster*:
* Las observaciones que pertenecen a un determinado *cluster* no podrán pertenecer a otro distinto, es decir, la intersección de las observaciones de un cluster con otro, es el conjunto vacío:

Quizás el punto más interesante de este algoritmo es el que tiene que ver con la elección del *k* óptimo, siempre dando prioridad a la interpretabilidad de los clusters formados hay 2 tipos de técnicas principales:

* *Método Elbow*: Se mide la suma total de cuadrados interna a los *clusters* en función del número *k* de estos, se suele tener caídas de tipo exponencial y directamente desde el gráfico, a partir de un número, la caída es horizontal y este sería el número de grupos a elegir.
* *Método Average Silhouette*: Similar al método anterior sólo que en vez de minimizar la suma de cuadrados interna, lo que se hace es maximizar la media de los *coeficientes silhouette*. Estos coeficientes, asociados a cada uno de los *k* grupos creados, asigna un valor entre -1 y 1 y compara la similitud de observaciones dentro de un mismo *clúster* frente a la de los demás

Ejemplo 12: Eligiendo el número *k* de grupos por el *Método de la Silhouette*.

En el *ejemplo/programa06.py* se aplica el método a unos datos generados aleatoriamente por la función *make\_blobs()*

|  |  |
| --- | --- |
|  |  |

Aunque la suma de los *sillhouetes* es mayor cuando *k* = 2, parece razonable la opción *k* = 4 que mantiene similares valores y parece que separa bien frente a otras alternativas.

Nótese que, en este caso, hay que combinar el método analítico (*sillhouete*) frente al visual de la agrupación que se observa. En una situación real, además hay que poder interpretar dichas agrupaciones lo cual no siempre será fácil y posible.

\_\_\_\_\_

Ejemplo 13: Agrupando estados de USA por criminalidad

Al tenerse más de 2 variables, los datos no van a poderse visualizar fácilmente, en este caso hay unas 50 observaciones y se aplica el código de *ejemplos/programa07.py* bajo el *método de elbow* generándose el siguiente diagrama::

|  |  |
| --- | --- |
|  |  |

Con este diagrama, se debería elegir entre 4 y 5 grupos. Por otro lado, el análisis *sillhouete* puede dar una respuesta para discernir y completar la decisión final:

|  |  |
| --- | --- |
|  |  |

Finalmente se observa que *k* = 4 es una elección aceptable, por tanto, falta ahora la parte más compleja que es la interpretación de los 4 *cluster* formados en base a las variables escogidas.

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

\_\_\_\_\_

Para concluir esta técnica, cabe indicar que dentro de su tipología hay otras variantes como *k-medoids()* que lo que pretende es elegir un “centroide real”, seleccionando un representante del dataset más que un centro de masa que no tiene que estar ubicado sobre ningún dato en particular, por tanto lo que se minimiza aquí no es una suma cuadrática intra-cluster, sino la suma de las diferencias de cada observación mediante su *medoid* (o dato central), básicamente su uso mediante Python se realizaría con:

from sklearn\_extra.cluster import KMedoids

### Algoritmo de Ward

En los *clustering* de tipo *jerárquico* aparece un gráfico de modo natural como es el *dendograma*. Dicho gráfico va agrupando las observaciones por *niveles de jerarquía*. Estas *jerarquías* se construyen bajo distintos enfoques, bien de modo *aglomerante* (bottom-up) o de modo *divisivo* (top-down)

Una vez visto el dendograma, lo difícil es ¿Dónde se corta para obtener las agrupaciones finales? En el siguiente ejemplo, se observa un caso de aplicación a la agrupación de tiendas en orden al número de clientes y sus ingresos

Ejemplo 16: agrupando tiendas por su comportamiento

En el código *ejemplos/programa08.py* se aplica el el *Algoritmo de Ward*, donde se genera el siguiente diagrama:

Gráfico

Descripción generada automáticamente

Claramente hay muchos posibles puntos de corte, por ejemplo, si se decide “cortar” a nivel de distancia 300, se crearían 3 grupos, pero a 250, el número de grupos crece a 4. Un corte más lógico parece estar a 150, con lo que se formarían 5 grupos con suficientes observaciones y con posibilidad de bajar casi a 100 la distancia:

Una captura de pantalla de un videojuego

Descripción generada automáticamente con confianza media

Nota: Aunque habría que haber normalizado los datos no obstante, al estar el ingreso en miles la escala es parecida, pero se recomienda en todo caso normalizaición de observaciones en este tipo de caso

Dado que se tienen 2 únicas variables para clusterizar, sí se puede observar la separación que ha hecho el algoritmo y se observa en este caso que tiene bastante sentido en cuanto a posicionamiento de las observaciones en los ejes considerados

Gráfico, Gráfico de dispersión, Gráfico de burbujas

Descripción generada automáticamente

\_\_\_\_\_

La ventaja de este tipo de *algoritmos* es que se puede controlar en qué punto se corta y por tanto la decisión sobre el número de *cluster* a elegir se posterga hasta tener una primera impresión visual y por tanto junto con esa decisión, se puede tener una idea del tamaño del *cluster*, lo que permite aplicar cierta heurística en este método, como “buscar zonas donde se reduzca mucho la distancia y el número de observaciones de las agrupaciones sea suficientemente alto”, nótese que la representación bi-dimensional anterior puede aplicarse aunque se esté usando un número muy elevado de variables (siempre que se elijan pares adecuados), sin embargo si hay demasiadas observaciones puede dar lugar a problemas en la selección del número óptimo de clusters ya que la visualización tendrá “demasiado lío” con individuos muy cercanos y no muy separados entre sí como en el ejemplo anterior.

## 3.3. Clustering de variables

Otro gran grupo de *algoritmos* se centran en lo que se conoce como *clustering de variables*, en este caso, dada una *tabla*, en vez de asociar *registros*, de lo que se trata es de asociar *columnas*. Para ello el juego de *distancias* a utilizar es distinto al utilizado anteriormente siendo popular el uso de *correlaciones* que a veces se denominan erróneamente *distancia* aunque más bien serían *pseudodistancias*, ya que entre otras cosas, una *correlación* puede ofrecer valores entre -1 y 1 y por tanto no puede considerarse como tal. Sin embargo, para ello se hace uso a veces del concepto de *r-cuadrado* que en este caso sí se estaría hablando de un número entre 0 y 1 y de un concepto más cercano al de *distancia*.

*Clusterizar variables* es agrupar variables para saber cuáles tienen información parecida y entre otras cosas se evita la *multicolinealidad* que se comentará más adelante al generarse grupos de variables muy diferentes entre sí.

Uno de los algoritmos más conocidos el es el *varclust* popularizado por SAS y viene a funcionar del siguiente modo:

* Paso 1: Todas las variables están en un único cluster
* Paso 2: Bajo algún criterio de selección (que debe elegirse) se elige un cluster para generar una subdividsión que puede estar basada, por ejemplo, en escoger aquel que tenga el menor porcentaje de varianza explicada por sus componentes (variables del cluster)
* Paso 3: El cluster escogido se divide en 2, para ello se encuentra 2 componentes principales mediante rotación obliqua y se se asigna cada variable a la componente rotada con la cual tenga un mayor nivel de r-cuadrado
* Paso iterativo: Se repite 2 y 3 hasta que se llega al número de cluster especificado o bien se establece algún criterio de parada que tenga que ver con reducción de la varianza explicada o con valores en los autovalores

Hoy en día, este algoritmo está disponible en python tal y como se observa en el siguiente ejemplo.

Ejemplo 15: agrupando variables vinícolas

En el código *ejemplos/programa09.py* se lleva a cabo la aplicación del *algoritmo* *varclust* donde se discuten e interpretan algunas de sus salidas aplicadas al conocido dataset denominado *wine* con unas 11 variables, el problema tras llevar la agrupación es ¿Y cómo se aplica finalmente el algoritmo? ¿Se escoge una variable por grupo o una combinación de variables? ¿Qué variable escoger?

|  |  |
| --- | --- |
| Texto, Chat o mensaje de texto  Descripción generada automáticamente | Texto  Descripción generada automáticamente |

Nótese que lo único que la parametrización de la función *VarClusHi()* ha sido muy sencilla y tan sólo ha necesitado que se le indicase el dataset y un criterio de parada

\_\_\_\_\_

Ejemplo 16: comprensión de imágenes

Una aplicación que muestra que el algoritmo *k-means* es capaz de encontrar las agrupaciones adecuadas es la mostrada en el código *ejemplos/programa10.py* donde se toman todos los píxeles que componen una imagen y lo que se hace es generar, una serie de “pixel centroides” donde se asocian todos los demás. Así pues en la ejecución del código se genera la siguiente transición:

Imagen que contiene fruta, alimentos

Descripción generada automáticamente

De una foto original, la de la izquierda tomada del planeta Marte y cuyo código viene de un ejemplo que Andrew Ng explica en su curso de entrenamiento no supervisado

\_\_\_\_\_

# **4.- ENTRENAMIENTO NO SUPERVISADO II: MÁS SOBRE ANÁLISIS CLUSTER**

Mientras que los anteriores *algoritmos* de corte *no supervisado* caben ser denominados como los “clásicos” o habituales, actualmente se tiene a disposición *algoritmos* de corte más avanzado donde ya se ha creado desde hace bastante tiempo paquetes de software adecuados a partir fundamentalmente del año 2010, tanto en R como en Python y donde la potencia actual de los ordenadores permite tanto su aplicación, como el de tenerlo dentro de “la caja de herramientas” del DS, cuando se afronta un problema de este tipo.

En los siguientes *algoritmos* a enumerar se distingue entre los que se aplican a variables y permiten generar *proyecciones* en espacios de menor número de variables como las técnicas tipo *PCA* y aquellos que tratan de agrupar individuos o elementos de una determinada población de modo análogo a como se ha visto antes.

## 4.1. El Algoritmo t-SNE

Una técnica alternativa a las anteriormente citada es la denominada *t-SNE* o *t-Distributed Stochastic Neigbor Embedding*, cuya aplicación más directa tiene que ver con la reducción de la dimensionalidad. Bajo esta alternativa, a diferencia de la técnica clásica *PCA* (utilizada por primera vez por Fisher hace casi más de 100 años), se trata de encontrar relaciones no lineales entre variables, partiendo de distribuciones probabilísticas e interrelaciones expresadas en términos de grafos, de modo que el *t-SNE* es capaz de encontrar estructuras subyacentes en los datos y en ocasiones ofrecer reducciones de dimensionalidad más significativas que las obtenidas simplemente mediante aplicación del *PCA*.

El *t-SNE* trata de encontrar una representación *m-dimensional* (de dimensión menor que el problema de partida) de los datos y donde se preserven distancias o la estructuras posicionales *n-dimensionales* entre los individuos del *espacio* o *población* de partida lo máximo posible. Esta técnica comienza con una representación aleatoria *m-dimensional* (aquí se supone *m* = 2) de cada punto y lo que trata de hacer es mantener, en el *espacio proyectado*, asociaciones de puntos que estén próximos en el espacio original de características. En este caso se hace uso del concepto de *similaridad* entre puntos a través de la *probabilidad condicionada* de una gausiana que responde a la siguiente fórmula:

Bajo la anterior fórmula se mide “como de *similar* es un punto respecto a otro ”. Los pasos para tomar esta medida son:

* Se elije un punto cualquiera
* Se mide la *distancia euclídea* hasta otro punto (que cabe de denominar )
* La *distancia* anterior tiene que ser proporcional a una *gaussiana* centrada en
* El paso anterior se repite con el resto de los puntos
* Finalmente se distribuyen todos bajo la correspondiente gausiana

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | |

**Figura 5 Proceso de construcción de las *probabilidades condicionadas* en el *t-SNE***

**Source:** [**https://towardsdatascience.com/t-sne-clearly-explained-d84c537f53a**](https://towardsdatascience.com/t-sne-clearly-explained-d84c537f53a)

* Dada la anterior construcción se tendrá que:

Este valor se supone así

* Por tanto, se define la *similitud* en el *espacio* de salida de dimensión *N* como la siguiente cantidad:

En este sentido un parámetro que se considera en el *algoritimo* es el de *perplexidad* que viene a ser como el número de *puntos vecinos* a considerar alrededor de uno dado en cada iteración, en vez de considerar todos los *individuos* de la *población* a partir de este valor se denermina las *desviaciones típicas* que serán mayores, cuantos más *puntos de perplexidad*. Su valor suele estar entre 5 y 50, viniendo en python asignado por defecto en 30 y que suele ser fijo de modo que:

Por otro lado, hay que *proyectar* las anteriores *similitudes* en un *espacio* de dimensión menor, en este sentido, en dicho espacio debe haber el mismo número de puntos distribuidos aleatoriamente pero bajo una adecuada *ditribución de probabilidad* que por diversos motivos será una de tipo *t-student* con 1 grado de libertad que entre otras ventajas tiene que no necesita de un parámetro de *desviación típica*:

Finalmente, la estimación de todos los parámetros se realiza mediante el uso del *algoritmo del gradiente descendente* donde se usa la *suma* o *divergencia de Kullbar-Leibler*

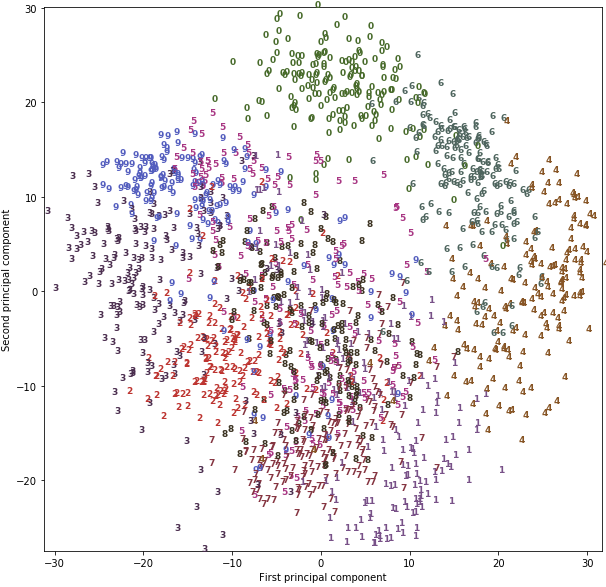
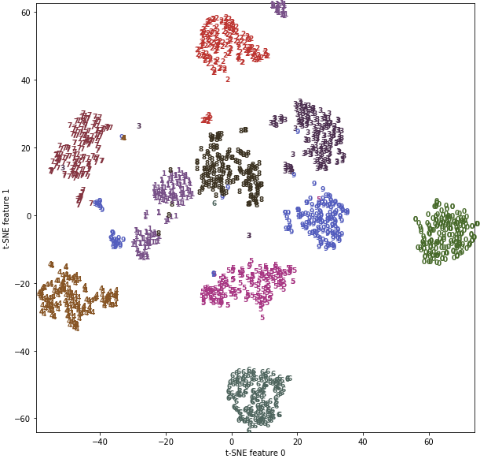
Y para enlazar estos conceptos lo que se pretende es minimizar mediante un *algoritmo* iterativo tipo *gradiente descendente*, lo que se conoce como la *suma de Kullback-Leibler*, que es un concepto que proviene de la teoría de la Información de Shannon y que se expresa tal como sigue:

A pesar de la aparente complejidad matemática la única idea realmente novedosa es que se crea una especie de “proyección inteligente” con ejes ortogonales construidos ad-hoc.

Actualmente la versión *t-SNE* en python es bastante óptima y para tema de reconocimiento de imágenes y de escritura puede ofrecer excelentes resultados como alternativa a otros métodos.

Ejemplo 16: Reconocimiento de escritura manual

En el código *ejemplos/programa11.py* se observa, como en problemas donde hay patrones claros, el *t-sne* tiene una enorme eficiencia:

Nótese que no se le ha dado al *algoritmo* información sobre el etiquetado, se han introducido las escalas del pixelado por grupo, y ambos *algoritmos* han actuado encontrando patrones. En el caso del *t-SNE* encuentra 10 “manchas” que se refieren a 10 *clusters* bastantes separados entre sí, mientras que el *PCA* no consigue separar más que a lo sumo en 2 con cierta confusión.

Pot otro lado, hay que notar que en ambos *algoritmos* se han creado 2 ejes en los que representar los puntos, dichos ejes son las salidas de ambos *algoritmos* de modo que se reducen las variables de entrada en sólo 2 dimensiones que permiten dibujar las anteriores manchas notándose que mientra el *PCA* es determinista, el *t-SNE* no lo es y si no se fija la semilla de aleatorización, la próxima vez que se ejecute, dará luar a manchas con formas distintas aunque seguramente con análogo poder de separación.

\_\_\_\_\_

## 4.2. El Algoritmo UMAP: Evolución del t-SNE

Una técnica alternativa a al *t-SNE* es la denominada *UMAP*, esta técnica hace uso de elementos de topología como la *Teoría de Complejos Simpliciales* que viene a ser como una esquematización de la representación de poliedros n-dimensionales mediante grafos (poliedros estos que pueden representar superficies en dimensión elevada). En la técnica a utilizar se hace uso de grafos con aristas ponderadas mediante la probabilidad de que 2 puntos resulten estar conectados bajo un determinado entorno que tiene un radio, dicho radio que debe ser elegido, de modo que si dicho radio es muy pequeño, se crean clústeres pequeños y muy aislados, mientras que si es excesivo, acabarán por estar todos los puntos bajo un mismo cluster, con lo que se aplica una idea similar al modelo que se ve mas adelante y que se denominada *k-NN*.

Por tanto este *algoritmo* trata de respetar en todo momento la estructura inicial y local de los puntos existentes. De ahí que resulte que sus parámetros más importantes resulten ser el denominado *n\_neigbourhg* y el *min\_dist*, con lo que se trata de hacer un balance de la estructura local de la disposición de los puntos, con la que resulta global y común a todos ellos. Entre las diferencias del *UMAP* y el *t-SNE* cabe considerar que el primero trata de preservar más la estructura original de los datos, mientras que resulta a la vez algo más robusto frente a cambios leves en sus parámetros. En todo caso, aunque en numerosas ocasiones van a ofrecer representaciones similares, debe tenerse en cuenta que tanto *t-SNE* como *UMAP* tienen problemas de interpretación de las formas finales de los datos que ofrece y pueden verse afectados por “ruido” en las observaciones.

Ejemplo 17: Reagrupoando las transcripciones numéricas

Volviendo al ejemplo anterior, el código *ejemplos/programa12.py*, muestra el uso de este *algoritmo* que de nuevo crea formas, quizás mucho mas separadas que el anterior, pero bajo una estructura mucho más extraña, no obstante lo interesante es que efectivamente, pueda realizar la separación de los distintos tipos de datos, que en este caso resultan ser los mismos en número, aunque no en forma

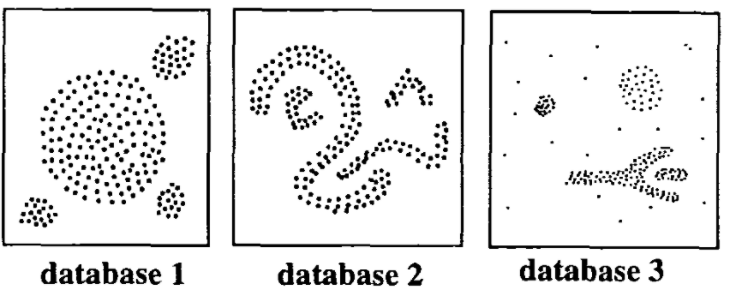
Gráfico, Gráfico de dispersión

Descripción generada automáticamente

\_\_\_\_\_

## 4.3. El Algoritmo DBSCAN

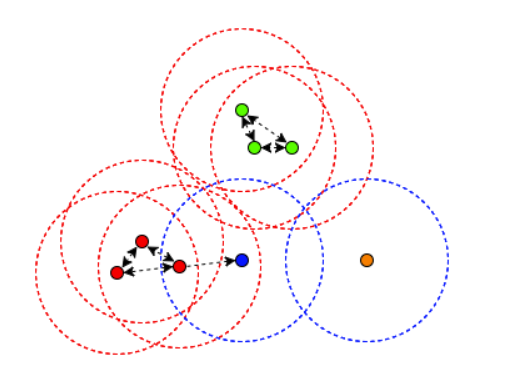
Mientras los anteriores *algoritmos*, estaban muy enfocados a la *reducción de la dimensionalidad*, gracias a lo cual, se conseguía, bajo adecuados ejes, separar observaciones de una determinada población, con estos 2 últimos *algoritmos* que se van a presentar se vuelve de nuevo a la *clusterización* a nivel de individuos donde cabe compararlos con la actuación de los *algoritmos k-means* o *ward*. Así pues, aparece la conocida técnica como *Density-Based Spatial Clustering*, que se desarrolló a mitad de los 90 y es un algoritmo que trata de buscar concentraciones de puntos manteniendo hasta cierto punto su estructura original:



**Figura 7: Asociaciones por aplicación del algoritmo DBSCAN**

Fuente: Ester et al. 1996

Así pues, a diferencia del *k-means* que hace uso de centroides y de entornos alrededor de éstos, el algoritmo interno del *DBSCAN* trata de usar las densidades de los puntos para descubrir su auténtico modo de agrupación y por tanto en algunos casos donde las formas no tengan porqué ser concéntricas, puede permitir obtener agrupaciones con cierto sentido, que por supuesto tendrán que ser posteriormente interpretadas. Una de las ventajas del algoritmo *DBSCAN* es que no es necesario especificarle un número de clústeres pre-determinado, él sólo se encarga de encontrar las formas subyacentes de los datos, sin embargo, esta virtud quizás sea también su principal desventaja ya que puede dar lugar a clústeres erróneos, sin sentido o imposibles de interpretar adecuadamente. Para guiar en la formación de los grupos, dos parámetros resultarán claves como son el número de puntos mínimos a partir de los cuáles se forma un grupo *min\_samples* junto con el radio para unir dichos puntos que se controlan con el parámetro *épsilon*.



Ejemplo 18: Unas formas extrañas I

En este caso se aplica este *algoritmo* a un “ejemplo de juguete” con datos generados con la función de pytthon *make\_blobs()*, con el código del *ejemplos/programa13.py*, se crea una serie de “manchas de puntos” como los siguientes que se desean agrupar

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Obsérvese que aquí aunque aquí no se elige el número de grupos, debe tenerse muy en cuenta el valor del parámetro *épsilon*, por lo que en problemas que no se puedan graficar adecuadamente los datos, elecciones erróneas puede dar lugar a errores en los grupos finales.

Un método que podría ayudar para determinar dicho parámetro es mediante la técnica *knn* que se comenta más adelante y donde en su versión *no supervisada*, cabe utilizarla para tener una idea de la *distancia* de los puntos en dimensión 2, con lo que se puede generar las *distancias* entre puntos, ordenadamente del siguiente modo (desde lo más cercanos a los más alejados)

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Por lo que el punto de mayor curvatura ofrece un valor *épsilon* = 0.3 que será el empleado junto con el estándar *min\_samples* = 5

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

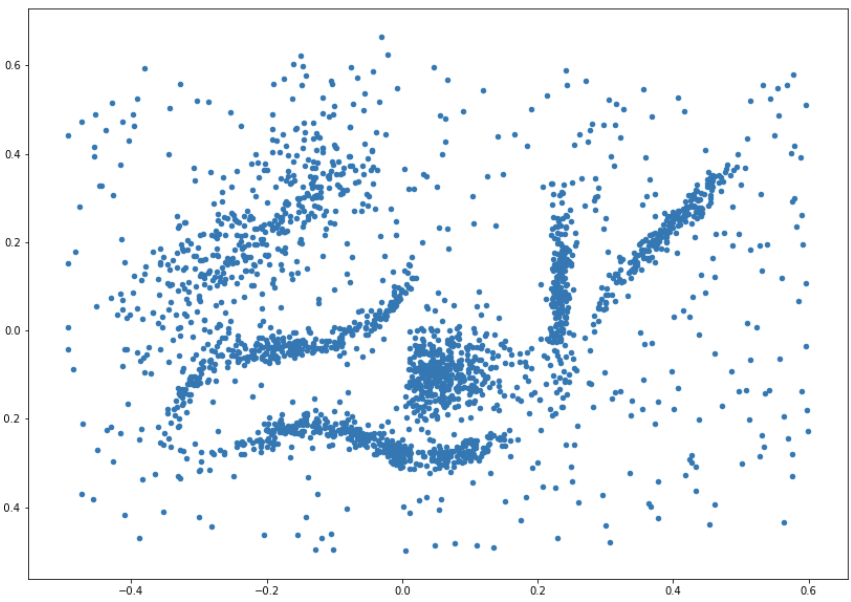
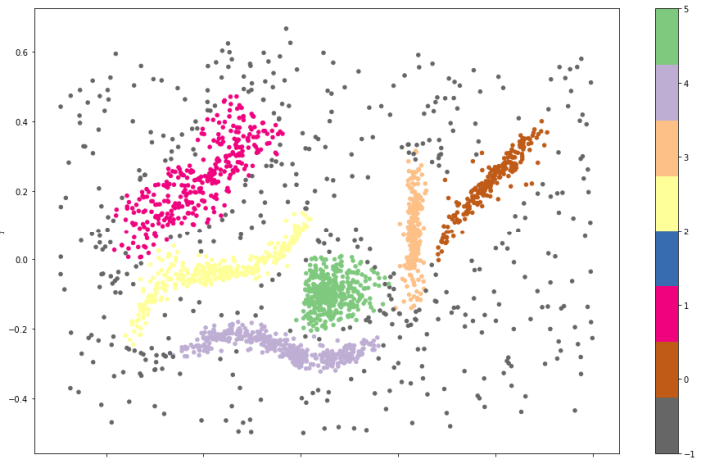
\_\_\_\_\_

## 4.4. El Algoritmo HDBSCAN

Otra alternativa de clasificación similar a *DBSCAN* es el denominado *HDBSCAN*, en este caso la diferencia fundamental con el anterior algoritmo es que, mientras con *DBSCAN* la densidad de puntos que se forma en cada cluster es la misma (o similar), en este caso las densidades pueden variar de *cluster* a *cluster*, al igual que en el caso anterior, existen unas ventajas claras como son la construcción de *clústeres* de observaciones que no tienen que ser esféricos, evita tener que tomar decisiones sobre el número de clústeres y se descarta puntos aislados, no obstante algunos inconvenientes es que adolece del conocido mal “curse of dimensionality” y que debe fijarse una semilla de aleatorización para reproducir resultados, a diferencia de algoritmos como el *k-means*, aquí no se garantiza la convergencia a una solución única final.

Ejemplo 19: Unas formas extrañas II

En el código de *ejemplos/programa14.py* se analiza este *algoritmo* que no se encuentra en la conocida librería *sklearn* pero que se aplicaría de modo exactamente igual al *dbscan*, solo que aquí se toman como parámetros principales *min\_cluster\_size* que tiene que ver con el mínimo número de elementos que formara cada cluster y el *min\_samples* que tiene que ver con el número de elementos a considerar cuando se hace una asociación

# **5.- DETECCIÓN DE ANOMALÍAS**

Las ténicas de *detección de anomalías* es otro tipo de *algoritmos* *no supervisados* en el que básicamente, asumiendo una *distribución* de los datos *normal* (en general *multivariante*), lo que se trata de detectar son los *outliers* a la hipótesis formulada.

En la versión sencilla que se analiza aquí, se va suponer que la estructura subyacentes de los datos, siguen una *normal*, en la cual deben de ser estimados sus parámetros:

* *Media*
* *Desviaciones típicas*
* *Covarianzas* (en el caso *multivariante* con relaciones de dependencia)

Aunque gran parte de lo analizado proviene del curso *usupervised learning* de Andrew Ng se trata de destacar aquí algunos elementos esenciales que permiten ser tratados en multitud de aplicaciones entre las que cabe encontrar:

* Detección de funcionamientos mecánicos irregulares
* Prevención de fraude
* Seguridad de la información
* Análisis de comportamiento de CPUs en centro de datos

El caso a considerar aquí, de datos *normales multivariantes* va a asumir el supuesto de *independencia* de las variables o de *multicolinealidad* nula, lo que lleva a que la *distribución normal* va a tener la siguiente forma funcional en general:

*Vector de medias*:

*Matriz de varianzas covarianzas*:

*Función de densidad de probabilidad*:

Que en el caso particular de vectores de dimensión 2, tendrá un aspecto geométrico tal como el siguiente:

Gráfico, Gráfico de superficie

Descripción generada automáticamente

**Figura Geometría de una normal multivariante**

**Source: Non – supervised training course of Andrew Ng**

Ejemplo 20: Midiendo la latencia y el rendimiento de una red de servidores

En el código *ejemplos/programa15.py*, se observa desde el scratch como funciona el algortimo para el caso de 2 variables. Dado que se pueden dibujar en un plano, se observa una situación tal como sigue, donde se desea que el algoritmo aprenda la regla para detectar los que se comportan de un modo extraño:

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

En este contexto se observa un núcleo duro de observaciones y después otras que están separadas del comportamiento general que son las que se desea ser detectadas.

En este ejemplo se cuenta con 2 datasets adicionales donde se tienen observaciones y donde se han marcado cuáles son outliers denominados *X\_val* e *y\_val* De este modo, a partir de aquí cabe crear un algoritmo que maximice la detección de estos outliers, siendo una elección la medida denominada *F1 – score* que se discutirá más adelante en este curso, así pues con esa medida se determina un *épsilon* o punto de corte que será el que se aplique. Por supuesto se podría haber tomado uno a partir de la inspección visual, pero es más ortodoxo tomar uno basado en observaciones reales.

Así pues, si se consideran outliers aquellos cuyo valor en de su *densidad de probabilidad* de la *normal multivariante* asociada sea menor que dicho *épsilon* = 8.9908e-05 se tiene el siguiente diagrama con la detección asociada:

Gráfico, Diagrama, Esquemático

Descripción generada automáticamente

\_\_\_\_\_

Claramente lo anterior puede extenderse a más de 2 variables, en ese caso debe hacerse uso de la función *GaussianMixture* de *sklearn.mixture* donde se evita tener que hacer una programación en *python* ya que claramente esta implementación resulta suficientemente eficaz.

Ejemplo 21: Midiendo la latencia y el rendimiento de una red de servidores II. El caso multivariante

En el código *ejemplos/programa16.py*, se toma una muestra análoga al caso anterior, per esta vez se consideran varias variables, en este caso se hace uso de la función mencionada anteriormente para dar solución al problema, pero dado el número de variables, se pierde la intuición geométrica sobre el funcionamiento del método y claramente en casos reales, es recomendable llevar a cabo un análisis más intenso para estar seguros de la idoneidad del punto de corte a establecer.

Texto

Descripción generada automáticamente

La parte final del ejemplo es la vuelta al principio pero haciendo uso de esta función que ahorraría el tener que programarlo todo desde el scratch. Es decir, se toma el caso de dimensión 2 pero se hace uso de la función *GaussianMixture()* para obtener un resultado parecido al del anterior ejemplo:

Texto

Descripción generada automáticamente

\_\_\_\_\_

# **BIBLIOGRAFÍA**

**Abramowitz, M.; Stegun, I. A. (1972)** *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* Applied Mathematics Series. 55 (9ª reprint with additional corrections of tenth original printing with corrections)

**Daly, R.; Shen, Q.; Aitken, S. (2011)** *Learning Bayesian networks: Approaches and issues* Knowledge Eng. Review. 26. 99-157. 10.1017/S0269888910000251.

**Hahsler, M; Grün, B; Hornik, K (2005)** *Arules – A Computational Environment for Mining Association Rules and Frequent Item Sets* Journal of Statistical Software, October 2005, Vol 14, Issue 15

**Grimmett, G; Stirzaker, D (2004)** *Probability and Random Process 3ed* Oxford University Press ISBN 0-19-857223-9

**Korl, K; Nichols, A (2011)** *Bayesian Artificial Intelligence 2ed* Ed. CRC Press ISBN 978-1-4398-1591-5

**Scutari, M (2010)** *Learning Bayesian Networks with the bnlearn R Package* Journal of Statistical Software, July 2010, Vol 35, Issue 3

\_\_\_\_\_\_\_\_\_\_\_\_\_

<https://www.cs.upc.edu/~belanche/Docencia/mineria/Practiques/R/arules.pdf>

1. Nótese que el problema conceptualmente sería bastante distinto al tradicional *BMA*. Hay que definir un horizonte temporal durante el cual un cliente realiza “su compra” e introduce en su cesta los productos, que en el caso de un banco puede ser 1 año o más (mientras que en el caso supermercado, la compra se hace cada vez que se va a la tienda). [↑](#footnote-ref-1)